

آشنای با Java و J2EE

تالیف و ترجمه: مهدی وجدانی

mehdi_vojdani@yahoo.com

این کتاب در ابتدا برای چاپ در نظر گرفته شده بود اما با وجود زیاده خواهی ناشران از نویسندگان جوان به صورت مجانی در اختیار علاقه مندان قرار میگیرد. کلیه حقوق این کتاب محفوظ است.

به نام خدا

عنوان فصل صفحه

۱	مقدمه ای بر J2EE و تفاوت‌های آن با .NET و IDE	۴
۲	مقدمه ای بر زبان Syntax جاوا و نحوه اجرای برنامه ها	۱۰
۳	عناصر واجزاء گرافیکی در جاوا JFC and Swing	۷۶
۴	اتصال به پایگاه داده با JDBC	۱۲۷
۵	کار با JSP و Servlet	۱۴۳
۶	Java XML	۱۷۲
۷	آشنای با EJB	۲۰۸
۸	برنامه نویسی موبایل J2ME	۲۳۶
۹	مراجع	۲۵۵

فصل اول

مقایسه .NET و J2EE

مقدمه

در ابتدا قرار بر وجود این فصل نبوده ولی بنده آنرا بدون تغییر در این جا قرار دادم.

دات نت : یعنی يك بستر براي ایجاد نرم افزار و توسط مایکروسافت ایجاد شده اما تمام دات نت چیزی بیشتر از يك "بستر اجرائی کد و کتابخانه مقدماتی کلاس" نیست . تمام دات نت يك Framework است و يك محیط تولید نرم افزار که برپایه استفاده از DLL و مستندات جهت ایجاد ابزارهای متفرقه تولید نرم افزار و فقط روی ویندوز کار کرده .

J2EE يك استاندارد است که براي پاسخ دادن به يك نیاز نرم افزاري " سازمان مقیاس" چگونه باید با اجزاء نرم افزار رفتار کرد و براي مدیریت طول عمر نرم افزار (Application Lifecycle Management) چکار باید کرد . این استاندارد توسط سان ارائه شده . خیلی ها مبتنی بر این استاندارد نرم افزارهای خودکار سازی ایجاد کرده اند ، سان مایکروسیستمز نیز هم . استاندارد J2EE که چگونه با "زبان جاوا" يك Framework ایجاد کنیم ، چگونه کتابخانه کلاس براي تمام مقاصد بنویسیم ، بانک اطلاعاتی چطور باشد و ... و شرکتهای متعددی بر اساس این استاندارد Application Server های مبتنی بر J2EE ایجاد کرده اند که برخی شون تمام J2EE رو پیاده سازی کرده اند و برخی ها فقط بخشی از آن را

Application Server يك زیر ساخت نرم افزاري است که وظیفه اش Application Lifecycle Management است . یعنی از ابتدای تولد يك نرم افزار "سازمان مقیاس" تا انتهای اتمام تولید ، باید به تمام نیازهای نرم افزاري پاسخ دهد . یعنی اگر برنامه نویس به يك Framework احتیاج داشت ، Application Server يك Framework به او بدهد ، اگر بانک اطلاعاتی خواست ، Application Server يك بانک اطلاعاتی کامل برایش فراهم کند ، اگر وب سرور خواست ، Application Server يك وب سرور تمام عیار به او بدهد ، اگر برای ارسال نامه های الکترونیکی برنامه اش به يك SMTP سرور نیاز داشت ، Application Server يك SMTP سرور به او بدهد ، اگر قرار شد براي احراز هویت از Kerberos استفاده کند يك پیاده سازی کربرایزد از استک TCP/IP در Application server وجود داشته باشد ، اگر خواست داده های کاربری رو از کارتهای هوشمند (smart Card) دریافت کند ، رابطهای لازم و API های مربوطه را از Application Server بگیرد و ... به دیگر بیان Application Server يك محیط Integrated است براي طراحی و تولید و مدیریت و توزیع و کاربرد يك نرم افزار "سازمان مقیاس" .

فرض کنید اگر قرار شد يك گروه نرم افزاري براي بزرگترین سازمان بیمه غیر دولتی امریکا يك راهکار جامع ERP تولید کند (یا بخرد و خصوصی سازی کند براي محیطش) باید به چه بستری اعتماد کنه که مطمئن باشه تمام درخواستهای نرم افزاري " سازمان مقیاس" ش رو میتونه جواب بده و مشخصه های اون ، سازمانش رو به يك نرم افزار خاص ، سخت افزار خاص ، پروتکل خاص و ... محدود نمیکنه ؟ (اصولاً محدودیت در ادبیات آی تی ، سطح اعتماد و قابلیت وثوق - Reliability - رو کاهش میده) اینجاست که يك Application Server لازم میشود . يك Application Server تضمین میکند که از بستر اجرائی کد گرفته تا وب سرور ، از توابع امنیتی گرفته تا بانک اطلاعاتی ، از IDE گرفته تا ابزارهای حمایت از UP (یا Unified Process) و ... در بسته نرم افزاریش وجود دارد .

طبیعی است که زبان جاوا زبان استاندارد توسعه نرم افزارهای مبتنی بر J2EE باشد ، هر چند بر خلاف اظهارات ناشیانه برخی ، J2EE و خصوصاً "بستر اجرای کدش ، به زبان جاوا منحصر نیست . یعنی همونطور که [مثلاً] بستر دات نت قابلیت پذیرش زبانهای مختلف رو داره ، بسترهای مبتنی بر جاوا هم میتونن به سایر زبانهای برنامه نویسی سرویس بده . یعنی براحتی میشه بین جاوا و سایر کتابخانه هایی که توسط سایر زبانهای برنامه نویسی تولید شده ارتباط برقرار کرد . (Java Native Interface) هر چند که مثل دات نت منعطف نیست . شرکت سان تلاش میکند يك Application Server مبتنی بر استاندارد خودش یعنی J2EE تولید کند اما هنوز تکمیل نشده . (سان فعالیت گسترده ای برای توسعه خود جاوا و بهینه سازی منطق J2EE و کلاسهای تولید نرم افزار دارد.

در ایران اغلب اوراکل رو به عنوان يك بانک اطلاعاتی میشناسن در حالیکه بانک اطلاعاتی اوراکل فقط بخشی از اون چیزی است که اوراکل تحت عنوان e-Bussines Suite منتشر کرده . Application Server اوراکل تمام اون چیزهایی که در وصف يك Application Server عرض کردم داره . بطور مختصر و لیست وار در موردش توضیحاتی عرض میکنم تا کمی روشنتر بشه بحث :

۱. يك بانک اطلاعاتی کامل : اوراکل فعلاً تنها بانک اطلاعاتی است که نه تنها نسخه های متعددی برای MainFrame ها داره ، برای تمام بسترهای نرم افزاری و سخت افزاری موجود هم نسخه هایی رو ارائه کرده . بزرگترین بانک اطلاعاتی که این حقیر در جریانش هستم و با اوراکل کار میکنه بانک اطلاعاتی وزارت انرژی ایالات متحده آمریکاست که روی يك MainFrame شرکت IBM اجرا شده . بانک اوراکل يك نسخهء کامل زبان پرس و جوی ساخت یافته یعنی PL/SQL ، يك سوئیت کامل بنام PSP که برای تولید صفحات وب بطور مستقیم از PL/SQL استفاده میکنه است . اوراکل تنها بانک اطلاعاتی است که موتور آن (DB Engine) هم میتونه بصورت توزیع شده و چند بخشی (Clustered) اجرا بشه . حتی میشه بخشی از انجین رو روی يك بستر کوچک وینتل (ویندوز + اینتل) و بخشی دیگر رو روی يك ماشین غول پیکر HP مجهز به HP-UX اجرا کرد . حتی میشه حین سرویس دهی بانک ، بانک رو از يك پلت فرم به پلت فرم دیگه منتقل کرد . (ویژگی های منحصر به فردش رو عرض کردم)

۲. يك بستر اجرای کد نرم افزار : Application Server شرکت اوراکل بطور کامل "بخش نرم افزاری J2EE" یعنی کتابخانه های کلاسش رو پیاده سازی کرده .

JDBC Connectors
JSP Engine
JavaBeans Engine
RMI
JMS
JINI
JMX
JIRO
J2EE CORBA ORB
JXTA
JXML
JCP
JNI
Web Service Implementation
و ...

یعنی هر کسی هر برنامه ای مبتنی بر J2EE نوشته باشه در بستر Application Server اوراکل قابل اجرا و سرویس دهی است . اوراکل J2SE و J2ME رو هم حمایت میکنه (دومی برای تولید برنامه های موبایل برای پورتابل دیوایسها کاربرد داره) . همچنین اوراکل بطور کامل

يك نسخه از Java Smart Card API رو پياد سازي كرده . در حال حاضر جاوا تنها ابزاري است كه ميشه توسط اون تقريباً" براي تمام كارتهاي هوشمند برنامه نوشتن اينكه توسط قابليت فوق الذكر قطعه كدهاي قابل ذخيره سازي در كارتهاي هوشمند هم قابل توليد است . فرض كنيد يك تابع تبديل تاريخ مينويسيد و تابع رو داخل كارت هوشمند قرار ميديد ، هر وقت نرم افزار اون تابع رو صدا زد كارت رو در كارت خوان ميگذاريد و برنامه شما تابع رو روي كارت هوشمند صدا ميزنه و جواب ميگيره بدون اينكه در مورد پياده سازي اش چيزي بدونه .

۳. اوراكل يك وب سرور مخصوص به خود ، همچنين سرورهاي :

POP3
SMTP
FTP
WebDav
Cache Server
Common Internet File system - CIFS
LDAP compatible Directory Service
...

رو بطور كامل پياده سازي كرده . تمام اين سرورهاي نرم افزاري كاملاً" با هم سازگار هستند و براي كار روي يك محيط مبتني بر J2EE بهينه سازي و خصوصي سازي شدن .

۴. اوراكل يك content Management System داره كه قابليت ايجاد پورتال هاي مبتني بر وب روي اينترنت يا اينترانت رو به "نرم افزار" هاي J2EE میده .

۶. اوراكل يك سرويس (يعني نرم افزارهائي + سرويس دهنده هائي) براي ايجاد ويژوال گزارش از بانك اطلاعاتي داره . گزارشها ميتونن طراحي بشن تا از داده ها استفاده كنن و خروجي بدن ، يا يك سرويس توليد گزارش به يك نرم افزار متصل بشه تا در زمان اجرا مولفه هاي گزارش به سرويس گزارش درخواست داده بشن تا گزارش رو طراحي كنه ، به داده متصل كنه و خروجي بده . بهش ميگن Reporting Service

۷. ابزارهاي مديريتي قدرتمند براي كنترل ترانكشهاي بانك اطلاعاتي خارج از محيط بانك (منحصر به فرد) ، كنترل وضعيت اشياء مثلاً" EJB ها و سطح دسترسي آنها ، انتقال سرويسهاي از يك پلت فرم به پلت فرم ديگر بدون توقف روند سرويس دهی ، صف گذاري منطقي و مديريت شده درخواستها و ...

۸. Load Balancer اوراكل كمك ميكنه سرويسها ، بانكهاي اطلاعاتي و سرورها و ساير نرم افزارهاي مبتني بر وب يا شبكه روي يك بستر توزيع شده اجرا بشن و اگر فشار ترافيك روي يك سرور زياد بود ، Load Balancer درخواستها رو به ساير سرورها كه توسط قابليت Replication Service اوراكل بصورت mirror آماده هستند هدايت ميكنه . اين Load Balancer قابليت درك جلسات کاربري (Session) ها يا مثلاً" متغيرهاي سطح برنامه (Application Level Variables) رو داره . يعني اگر شما به كتابخانه ملي سنای امريكا (Powered By Sun) لاگ اين كنيد و در حال انتقال صفحاتي از يك كتاب به دايركتوري شخصي خودتون باشيد و فشار روي سرور بانك اطلاعاتي زياد بشه ، درخواستهاي بعدي شما بصورت خودكار به سرور خلوت تري ارسال ميشن بدون اينكه State-Less بدون محيط به كانال ارتباطي شما لطمه بزنه ، يعني هويت شما و Session شما همچنان معتبر است اما روي يك سرور ديگر (اين منحصر به فرد نيست اما فقط شركت مكرمديا در JRUN كه اون هم يك Application Server نصفه نيمه است چنين چيزي داره كه در مورد اون هم مطلبي عرض ميكنم)

نتيجه اول : اگر شما يك Application Server كامل و قابل اتكاء ميخواهيد بايد بستهء نرم افزاري فوق العاده گران قيمت Oracle 11i - e bussines Suite رو تهيه كنيد كه هر آنچه ذكر شد داخلش موجوده .

نتیجه دوم : برای پاسخ دادن به نیازهای "خیلی بزرگ" و "سازمان مقیاس" که هزینه های میلیونی و میلیاردي برایش در نظر گرفته میشه و خطا در انتخاب معماری یا عدم سازگاري اجزاء نا بخشودني است فقط باید از يك Application Server استفاده کرد که ضمن دارا بودن همه چیز یکجا ، سازگاري ، قابلیت اتکاء و وثوق ، يك شرکت بزرگ و عريض و طويل ازش حمایت کنه . اوراکل یکی از پیشنهادهاي موجوده . در بازار E- bussines دنیا اوراکل قدرتمند ترین فروشنده نرم افزاره ، فقط به عنوان مثال مراجعه کنید به خبري از اوراکل که يك پیشنهاد " نه میلیارد دلاري" برای خرید شرکت People Soft (تولید کننده ERP هاي قدرتمند) داده . لازم به ذکره ۹ میلیارد دلار ، کمی کمتر از نصف بودجه یکسال جمهوری اسلامی ایران است !!

یاد آوردی اول : يك بستر اجراي کد مثل دات نت ، با يك استاندارد برای تولید Application Server يعني J2EE قابل قیاس نیست .

الف. کتابخانه کلاس : دات نت و JDK هر دو کتابخانه هاي قدرتمندي هستند که اغلب نیازهاي پایه برای تولید نرم افزار رو حمایت میکنند . اما برای تولید نرم افزارهاي بزرگ مقیاس هیچکدام کافي نیستند . دات نت چیز دیگری ندارد اما برای جاوا راهکارهاي دیگری هم وجود دارد . کاربري دات نت راحت تر است . پیچیدگی هاي دات نت هم کمتر است . نمودار یادگیری جاوا بسیار کم شیب است . (اگر نمودار عمودي پیشرفت باشه و نمودار افقي زمان) در حالیکه یادگیری دات نت خیلی سریعتر است .

ب. زمان اجرا : زمان اجراي دات نت تقلیدی صرف از زمان اجراي جاوا ست . هیچ بحثي هم درش نیست . حتی کسانی که مثل بنده عقلشون کم باشه و بشینن و IL رو با ByteCode مقایسه کنن درک خواهند کرد که مایکروسافت خلاقیتی از خودش نشون نداده . JIT در هر دو محیط خوب است . سرعت اجراي "برنامه" هاي دات نت از برنامه هاي جاوا کندتر است اگر از JIT استفاده نکند . این حقیقت رو هر کسی با چند آزمایش کوچولو میتونه درک کنه . سیستم Code Caching و JITC دات نت کمک زیادی به افزایش سرعت برنامه ها کرده . جاوا با عمر طولانی اش به ادعای اسکات مک نلی حدود پنجاه بار بهینه سازی شده در حالیکه دات نت هنوز جوونه . به نظر میاد در این يك مقوله باید منتظر آینده شد . اما في الحال وضع دات نت در این راستا خوبه .

ج. اتصالات : دات نت از ریموتینگ ، وب سرویس و کام پلاس حمایت میکنه (بصورت داخلي) . جاوا بجای ریموتینگ چیزی بنام ریموت متد اینووکیشن داره ، وب سرویس رو حمایت میکنه ، CORBA رو حمایت میکنه ، چیزی بنام EJB داره که اشیاء شناور در يك "مخزن سازمانی" هستند که افراد ، سرویسها و نرم افزارها بنا به میزان دسترسی میتونن ازش استفاده کنن . کنترلرهای دات نت هنوز چنین قابلیتني ندارند و دات نت هنوز راهي برای ایجاد يك Object Repository سازمانی ارائه نکرده . اشیاء کام پلاس و محیط MTS ویندوز هم (با اینکه ربطی به دات نت نداره بطور مستقیم) مانند EJB ها منعطف نیستند . EJB ها State-Less نیستند .

د. ارتباط با داده : دات نت چیزی بنام ADO.NET ارائه کرده که راه حلی است منحصر به فرد . جاوا JDBC رو داره که چه در connection Pooling و چه در objecy pooling به خوبی ADO.NET کار میکنه اما ADO.NET فوق العاده امکانات زیادی داره . چون اینجا دات نت کار زیاده لزومی به توضیح نیست . من با تمام وجود به ADO.NET اعتقاد دارم و تصور نمیکنم معادلی داشته باشه (یکسال و خورده ای پیش چند مقاله کامل در مورد ADO.NET در سایت ایران دولوپرز نوشتم که این مطلب رو اونجا هم عرض کردم . یکسال قبل)

ه. امنیت : امنیت در این حوزه رو "من" به سه بخش تقسیم میکنم (تقسیم بندی کاملاً شخصی و تجربی)

۱.ه) حفاظت از متن کد

۲.ه) حفاظت از ارتباطات

۳.۵) حفاظت از خود بستر و حفظ مانائی

در مورد اول هر دو محیط ضعیف هستند . Obfuscator ها نمیتونن به مفهوم واقعی از کد حفاظت کنند و راهکارهای Third party موجود هم بیشتر به طنز شبیه هستند . با داشتن IL یا بابت کد براحتی کد اصلی یا کدی " با قابلیت های" کد اصلی قابل باز-تولید است . پسوردها ، اعداد خصوصی ، کلمه های عبور و ... براحتی قابل بازیافت هستند اگر در متن نرم افزارهای Managed دات نت یا برنامه ها جاوا بکار رفته باشند . اینجا واقعا هیچ ایمنی " نمیتواند" وجود داشته باشد .

در مورد دوم هر دو محیط با Open Standard ها کار میکنند . از SSL گرفته تا Kerberos و ارتباط با Directory Service ها گرفته تا CA . در این مورد تفاوتی وجود ندارد .

در مورد سوم تا حالا مستندی که بر قوت یا ضعف یکی دلالت کنه نخوندم (نمیگم نیست ، نخوندم) و تجربه شخصی و عملی هم ندارم .

في المجموع در حوزه امنیت دو محیط چندان متفاوت نیستند .

و. انتقال : جاوا از MainFrame ها تا کارتهای هوشمند رو حمایت میکنه . دات قراره بزودی بسترهای دیگه رو حمایت کنه . پس اصولاً در این زمینه هیچ رقابتی وجود نداره . جاوا پانزده سال جلو تره . من با مونو (که قراره بشه دات نت روی لینوکس) کار کردم و فعلاً " ناقص و غیر قابل اعتماد " مایکروسافت هم یقیناً تا انتهای ۲۰۰۵ هیچ نسخه ای از دات نت مبتنی بر NIX* ها توزیع نخواهد کرد .

ز. تولید محتوای وب : دات نت ASP.NET رو ارائه کرده . جاوا JSP رو . سرعت پاسخگوئی دات نت در کاربردهای معمولی بالاتره . اما با توجه به محدودیت ویندوز (به عنوان تنها بستر دت نت) برای حمایت از ترافیک و فشار بالا ، اگر کاربردهای خیلی سنگین مد نظر باشه ASP.NET نمیتونه حرفی داشته باشه . موتور JSP هم قابلیت Clustering داره و میشه مجموعه ای از سرورها رو با " یک موتور" راه اندازی کرد . (میدونم به بحث ربطی نداره اما یکبار یکی ازم پرسید چرا مایکروسافت برای MSN و هات میل از فری بی اس دی استفاده میکنه ؟ و نه ویندوز ؟ جواب بنده این بود دلیل هر چیزی هست ربطی به امنیت نداره . سایت خود مایکروسافت با ترافیک بالا و دشمنانی قسم خورده بدون مشکل داره روی ویندوز کار میکنه . اما وقتی قرار باشه بخاطر ترافیک خیلی بالای مسنجر و ایمیل ، از یک ماشین با مثلاً ۳۰ تا پردازنده استفاده بشه تجربه ویندوز چندان موفقیت آمیز نیست ! در حالیکه فری بی اس دی - اچ پی یو ایکس و سولاریس همین حالا روی ماشینهای بیشتر از پنجاه پردازنده هم خوب کار میکنند . سان سرور بنام K Fire۱۵ داره - رک بخش سرورهای سایت سان - به قیمت " ده میلیون دلار" میفروشتش و ۱۰۵ تا پردازنده ۶۴ بیتی داره و همین نسخهء سولاریس معمولی روی اون هم کار میکنه و جواب میده و توانائیش ۶۵۰۰ میپسه ! یعنی ۲۰۰۰ میپس قوی تر از بزرگترین مین فریم IBM - تاریخ این امار متعلق به یکسال پیشه که من پروژه ای داشتم در (این مورد)

نتیجه : برای کاربردهای عمومی وب یعنی اونچیزی که در ۹۹ درصد اوقات مد نظره ASP.NET بهتر است مگر اینکه برنامهء خاصی برای انتقال وجود داشته باشه یا احتمال وجودش قابل تأمل باشه .

نتیجه کلی : تا اون حد که دات نت امکانات و توانائی داره ، قابلیت های مشابهش در بستر جاوا موجوده . در برخی موارد دات نت و در برخی دیگر جاوا برتر است الا اینکه اگر کاربرد خیلی بزرگ باشه یا برنامه خاصی برای انتقال بستر وجود داشته باشه یا احتمال وجودش قوی باشه ، در هر حال " تنها گزینهء موجود" جاوا ست ، در غیر این صورت باید بررسی کرد .

< تمام بحث مقایسه بسترهای دات نت و جاوا >

سوال : من متوجه شدم دات نت دقیقا" چیه و جاش کجاست و متوجه شدم يك Application Server چیه و به چه دردی میخوره و باز هم متوجه شدم فرق اینها در "مقیاس" پروژه است ، حالا میخوام کمی درمورد Application Server های دیگه بدونم .

جواب : اینترنت دریائی از اطلاعات است که میتونید ازش کمک بگیرید . تجربه شخصی من به استفاده از اوراکل و اورپون محدوده . در مورد JRUN هم مطالعه کردم . اورپون يك Application Server مبتنی بر J2EE است اما برای محیط لینوکس بهینه سازی شده است . در یکی از شرکت های نفتی ایرانی هم داره ازش استفاده میشه و فوق العاده جوابگوست . اما مثل اوراکل کامل نیست ، مثلاً" بانک اطلاعاتی نداره ، باید از چیزی مثل اوراکل یا مای اسکول استفاده کرد ، و نواقصی از این دست اما مجانی است و سورتس آزاد . www.orionserver.com .

JRUN محصول مکرومدیاست . این هم ناقصه و خیلی از قابلیت های اوراکل رو نداره (اوراکل خیلی خیلی گرونه) اما برخی مزایای خاصش باعث میشه آدم به انتخابش فکر کنه . مثلاً" قابلیت کلاسترینگ و لود بالانسینگ داره یا مثلاً" ColdFusion رو حمایت میکنه و ... قیمتش هم ارزونه . مثلاً" همین حالا سازمان ملی علوم و تحقیقات و تکنولوژی امریکا یعنی NIST داره از جی ران استفاده میکنه و زبان برنامه های وب اش ، خصوصا بخش امنیتی اش که زیاد کل کل میکنه هم کلد فیوژن است . اپلیکیشن سرورهای دیگه ای وجود دارن که چندان معروف نیستند . مثلاً" Borland Application Server که این مورد هم مبتنی بر J2EE است و بخشی از سایت خود بولند هم روی همین کار میکنه . اپلیکیشن سرور بولند از اورپون و جی ران کاملتره هر چند هنوز هم از اوراکل عقب تره . دپارتمان نرم افزار "ارتش امریکا" هم بطور کامل از محصولات بولند استفاده میکنه . برای توسعه نرم افزارهای ویندوزی از دلفی ، برای UP از توگدر و برای ارائه سرویس از اپلیکیشن سرور بولند . (وایت پیپر هاش رو میتونید تو سایت خود بولند پیدا کنید) و ... موارد متعدد دیگه .

جور اول : (تجربه شخصی)

کار با جاوا یا در مقیاس های بزرگ Application Server ها جاوا واقعا" سخته . (خصوصا" اگر آدم به محیط های قدرتمند و راحتی مثل دلفی عادت کرده باشه) در حالیکه کار با دات نت واقعا" راحت . کاربری جاوا هم مشکل تر از دات نته . در محیط دات نت اغلب تنظیمات یا وظایف کلیک اند ران هستند در حالیکه برای آماده سازی يك محیط مبتنی بر جاوا برای ارائه واقعی سرویس تخصص و تجربه لازمه و همیشه تجربه های اولیه با شکست همراه هستند . دات نت گرون نیست هر چند اگر واقع بین باشیم مجانی هم نیست . جاوا مجانی است و سورتس آزاد . اون چیزی که من بهش فکر میکنم اینه که برای کاربردهای کوچك ، معمولی ، متوسط دات نت مناسبه . برای کاربردهای واقعا" بزرگ دات نت اصولا" جوابگو نیست که بخاد مناسب باشه یا نباشه و جاوا تنها گزینه است حالا میخواد خوب باشه میخواد بد باشه . یعنی اگر قرار باشه سازمانی تیم نرم افزار تشکیل بده ، يك الگو و راه حل جامع (Total Solution) برای نرم افزار انتخاب کنه ، برای برنامه نویسه اش پول خرج کنه و پول بیشتری خرج کنه تا بمونن ، قرار نیست برنامه هاش خیلی خیلی بزرگ باشن ، دات نت گزینه خوبییه . اگر سازمانی قراره تیم نرم افزار داشته باشه و برنامه های فوق العاده بزرگ بنویسه که با توجه به نوع کاربرد احتمال تغییر پلت فرم یا خرید ماشین های بزرگتر و قوی تر و تغییر پردازنده و محتمل باشه ، اون محیط مال جاواست . امیدوارم در تمام متن مطلبم به عبارت " سازمان مقیاس" که برجسته تر بود دقت کرده باشید .

فصل دوم

آشنای با Java و J2EE و نحوه اجرای برنامه ها

نحوه اجرای برنامه های جاوا

برای اجرای اولین برنامه سه گام زیر را باید انجام دهید
 - در اولین قدم برای اجرای برنامه های خود نیاز به نصب Java™ 2 SDK (software Development Kit), Standard Edition دارید. اگر این برنامه موجود نیست آنرا از آدرس <http://java.sun.com/j2se/> دانلود کنید. اگر از IDE های موجود استفاده می کنید، آنها به طور خودکار عمل نصب و اجرا را انجام خواهند داد.
 - سپس برنامه نوشته شده را در فایل با پسوند java ذخیره کنید. بطور مثال HelloWorldApp.java

```
public class HelloWorldApp {
    public static void main(String[] args) {
        // Display "Hello World!"
        System.out.println("Hello World!");
    }
}
```

- برای کامپایل کردن پنجره Command Prompt را باز کرده و دستور cd c:\java را وارد کنید. اگر در يك درایو دیگر SDK را نصب کرده اید به جای درایو C از آن استفاده کنید.

```

MS-DOS Prompt
8 x 12
C:\WINDOWS>cd C:\java
C:\java>dir

Volume in drive C is DB02
Volume Serial Number is F3C4-E800
Directory of C:\java

.                <DIR>                07-22-99 11:23p .
..               <DIR>                07-22-99 11:23p ..
HELLOW~1 JAU      272    07-22-99 11:23p HelloWorldApp.java
1 file(s)        3,829 bytes
2 dir(s)         218,734,592 bytes free

C:\java>

```

به منظور کامپایل دستور زیر را وارد کند.

Javac HelloWorldApp.java

اگر پیغام خطا مبنی بر عدم پیدا کردن فایل javac دریافت کردید یک راه برای پیدا کردن javac این است که اگر Java 2 Software Development Kit در آدرس c:\jdk1.4 باشد دستور زیر را وارد کنید.

c:\jdk1.4\bin\javac HelloWorldApp.java

پس از اینکه کامپایل بدون خطا انجام شد یک فایل با پسوند class. به همان فولدر اضافه خواهد شد. این فایل همان فایل بایت کد است که در مفسر جاوا اجرا می شود.

```

MS-DOS Prompt
8 x 12
C:\java>dir

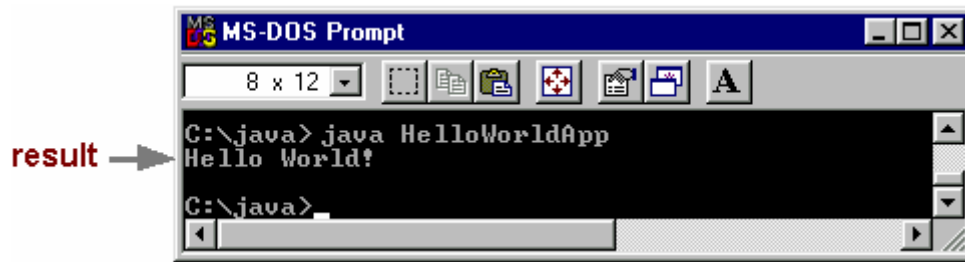
Volume in drive C is DB02
Volume Serial Number is F3C4-E800
Directory of C:\java

.                <DIR>                07-22-99 11:23p .
..               <DIR>                07-22-99 11:23p ..
HELLOW~1 JAU      272    07-23-99 12:39a HelloWorldApp.java
HELLOW~1 CLA      478    07-23-99 12:40a HelloWorldApp.class
2 file(s)        750 bytes
2 dir(s)         218,734,592 bytes free

C:\java>

```

- برای اجرای برنامه در همان فولدر دستور java HelloWorldApp را وارد کنید و خروجی را به شکل زیر خواهید دید.



اولین برنامه به زبان جاوا:

در زیر اولین برنامه به زبان جاوا را که پیغام HelloWorld را نشان میدهد مشاهده می کنید

```
public class HelloWorld {
// A program to display the message
// "Hello World!" on standard output
public static void main(String[] args) {
System.out.println("Hello World!");
}
} // end of class HelloWorld
```

تمامی برنامه های جاوا در داخل کلاس خود اجرا می شوند. اولین خط برنامه بالا نام کلاس را که HelloWorld میباشد بیان میکند. این نکته مهم را باید در خاطر بسپاریم که نام برنامه جاوا نام آن کلاس می باشد و در واقع نام فایل برنامه را که نیز برای کامپایل و اجرا استفاده می کنیم باید همان نام کلاس باشد.

در هنگام اجرای برنامه همیشه متد main() از داخل کلاس برنامه شبیه به شکل زیر اجرا خواهد شد (شبیه به برنامه های زبان C)

```
public static void main(String[] args) {
statements
}
```

زمان اجرا مفسر جاوا متد main() را اجرا می کند و اگر آن برنامه پارامتر ورودی داشته باشد، آنها را در داخل آرایه ورودی از نوع رشته ای به نام args قرار می دهد. کلمه public به این معنی است که متد main() می تواند از بیرون برنامه فراخوانی شود. در اینجا نوع خروجی که پوچ (null) با کلمه void مشخص شده. در قسمت مشخص شده با statement دستورات برنامه را که در مثال ما System.out.println("Hello World!") می باشد وارد می کنیم.

در زیر فرم کلی کلاس برنامه جاوا که علاوه بر متد main() سایر متدها و متغیرهای کلاس را به همراه دارد نشان داد شده.

```
public class program-name {
optional-variable-declarations-and-subroutines
public static void main(String[] args) {
statements
}
optional-variable-declarations-and-subroutines
```

}

نام برنامه با **program-name** که نام کلاس نیز می باشد مشخص شده و این کلاس می بایست در يك فایل به نام **program-name** و پسوند java ذخیره شود. در مثال ما HelloWorld.java می باشد و به این فایل سورس کد می گویند. بعد از کامپایل این فایل يك فایل دیگر به نام HelloWorld.class ساخته خواهد شد که شامل بایت کدهای (bytecode) جاوا برای اجرا در مفسر جاوا است. قسمت optional-variable-declarations-and-subroutines برای تعریف متغیرها و متدهای عضو کلاس است و از داخل متد main() قابل فراخوانی می باشند.

شی System.out يك کلاس از نوع PrintStream می باشد. کلاس PrintStream استاندارد جاوا برای چاپ اطلاعات بوده و هر شی از نوع این کلاس با متد print , println اطلاعات را به جریان خروجی ارسال میکند. جریان خروجی می تواند مقصدهای مختلف مثل يك فایل، شی دیگر، پورت در شبکه و سایر خروجی ها باشد.

تاریخچه پیدایش زبان جاوا

اولین نسخه زبان جاوا در سال ۱۹۹۱ توسط تیمی از برنامه نویسان شرکت سان میکرو سیستم به نام Oak ساخته شد و در سال ۱۹۹۲ با نام تجاری جاوا وارد بازار شد.

علت نام گذاری جاوا به این دلیل است هنگام انتخاب نام زبان افراد تصمیم گرفتند که ابتدای نامهای خود یعنی James Gosling, Arthur Van hoff و Andy bechtolsheim برای این زبان انتخاب کنند و همچنین چون مراسم نامگذاری کافی شاپ برگزار می شد یک فنجان قهوه داغ به عنوان سمبول زبان جاوا انتخاب گردید.

همانند سایر زبانهای برنامه سازی عناصر و اجزای اواج مجرد یا منفک از هم نیستند و در ارتباط تنگاتنگ با هم هستند. این پیوستگی اجزائی در عین حال توصیف یکی از وجوه خاص این زبان را مشکل میسازد. زبان جاوا یک زبان کاملاً نوع بندی شده می باشد.

در واقع بخشی از امنیت و قدرتمندی جاوا از همین موضوع است. اول اینکه هرمتغیری و عبارتی دارای نوع میباشد و دقیقاً معین می باشد. دوم این که کلیه انتسابها بطور مستقیم و صریح از نظر سازگاری کنترل میشوند. به این دلیل اجبار خودکار یا تلاقی در در هم پیچیدن پیش نخواهد آمد کامپایلر اواج کلیه عبارات و پارامترها و عبارتها را برای قابلیت سازگاری کنترل و بررسی میکند تا اطمینان حاصل شود و هرگونه ناسازگاری باید تصحیح شود. دقت داشته باشید همانند زبان ++C نمیتوانید دادهها هم شکل با طول کمتر را به طور مستقیم به نوع با طول بزرگتر تبدیل کنید مثلاً برای انتساب عدد صحیح به اعشاری باید قبل از عدد صحیح نام نوع آن عدد اعشاری را داخل پرانتز ذکر کنید.

استفاده از جاوا در اینترنت اولین بار توسط شرکت Netscape و با توافق با شرکت سان برای اینکه بتوان زبان استاندارد برای اجرای برنامه کوچک که در سمت سرویس گیرنده ها بود برای همین منظور چون اینترنت از به هم

پیوستن انواع مختلف شبکه ها و کامپیوتر ها بود . نیاز به طراحی زیر ساخت برای ماشینهای مختلف بود تا یک کد مشترک برای همه ماشینها را گرفته و بروی آن ماشین اجرا کند .
 به همین دلیل شرکت سان تصمیم به توسعه ماشین مجازی جاوا یا JVM (Java Virtual Machine) گرفت . حرکت اشیاء در شبکه و اینترنت باعث گسترش روزافزون جاوا شد و پویا شدن کد های فرستاده شده به سمت سرویس گیرنده به کمک Applet (mini application) حجم کا روی سرویس دهنده را کاهش می داد .

Applet

یک برنامه کاربردی جاوا که برای حرکت و انتقال در اینترنت و اجرا توسط یک مرورگر قابل انطباق با جاوا است و چون اکثر مرورگر ها از آن پشتیبانی می کنند و درست مثل یک تصویر با فایل صوتی با این تفاوت که Applet هوشمند است .

میتوان گفت نقش Applet همانند ActiveX در برنامه های Visual Basic می باشد .

امنیت

JAVA IDE

Borland JBuilder 2005 Enterprise Edition

JBuilder IDE نخستین IDE مائولار بود که جاوا را پشتیبانی کرد. معماری آن به خوبی مستند شده و راهنماهای موجود برای نوشتن plug-in ها قابل فهم هستند؛ بسیاری از فروشندگان واسط و همکاران کدباز (open source) اقدام به نوشتن ابزارهای plug-in برای اینپلاتفرم نموده اند.

JBuilder در سه نسخه عرضه می گردد: نگارش شخصی یا Personal Edition ، که به صورت مجانی قابل دانلود است، شامل IDE پایه و تعدادی ابزار اضافی از قبیل یک طراح GUI ، چاقوب یکپارچه JUnit ، و برخی آیتمهای دیگر است؛ ویرایش توسعه گر یا Developer Edition ، که گروهی از مشخصه های مختلف را گرد هم آورده است، به خصوص پشتیبانی XML و وب ، servlet ها، JSP ، و JSF (JavaServer Faces) ؛ و

ویرایش سازمانی یا Enterprise Edition، که سرویس‌های وب، J2EE، پشتیبانی CORBA، و رسم نمودار UML را به مجموعه می‌افزاید. از بین این سه ویرایش، JBuilder Enterprise Edition مورد بررسی قرار می‌دهیم. به دلیل تکامل آن، این IDE در میان چهار IDE که در اینجا مورد بررسی قرار گرفتند بهترین بود: عمل پیمایش در آن بصری و ساده است، و کلیک کردن در میان کارها هرگز به بن‌بست یا پنجره‌های غیر منتظره منتهی نمی‌گردد. در صورت مواجه شدن با مشکل، یک سیستم help در دسترس است – بهترین در میان چهار محصول دیگر. و برای توسعه‌گرانی که ترجیح می‌دهند کار را با آموختارها (tutorial) ها آغاز کنند، Borland گزینه‌هایی واضح با طراحی خوب فراهم می‌نماید.

پشتیبانی JBuilder برای مشخصه‌هایی که وجود آنها در یک IDE سازمانی high-end انتظار می‌رود ناب، هوشمندانه و کاربردی است. تنها استثنا تولید فایل برای Ant یک ابزار کمکی ساخت کدباز که معمولاً در جاوا مورد استفاده قرار می‌گیرد) است، که آزاردهنده می‌باشد. با چشم‌پوشی از این نقصان، محیط مزبور محیطی لذت‌بخش برای استفاده است.

فراتر از قابلیت IDE، JBuilder 2005 پشتیبانی برای برخی تکنولوژی‌های منحصربفرد، از قبیل یکپارچگی با CORBA، obfuscation، و تحلیل امنیت کد با استفاده از یک plug-in متعلق به Fortify را فراهم می‌آورد.

JBuilder همچنین به شکل قابل توجهی پشتیبانی برای XML و مشتقات بی‌شمار آن را تکمیل نموده است، و یک ویرایشگر HTML خوب به اضافه تعداد زیادی ابزار برای توسعه و تست سرویس‌های وب، و پشتیبانی برای J2ME و WAP را فراهم آورده است. فرقی نمی‌کند که کدام تکنولوژی با پروژه شما در هم آمیخته است، به احتمال قریب به یقین JBuilder آن را پشتیبانی می‌نماید.

JBuilder در هر صورت مدلسازی ضعیف است. بسته‌ی مزبور تنها دو نمودار UML را پشتیبانی می‌کند. این نقصان از آنجا ناشی می‌گردد که Borland اقدام به خریداری محصول Together نموده است. Together، یک بسته‌ی high-end مدلسازی است که در صورت نیاز شما به مدلسازی زیاد شما را به سوی آن هدایت می‌کند. از ابتدای ماه مارس، Borland یک مدل فروش را در پیش گرفته است که Together را با JBuilder و سایر ابزارهای Borland بر اساس نیازهای توسعه‌گر ترکیب می‌نماید. این مجموعه‌ی محصولات مبتنی بر وظیفه، که Core SDP نام گرفته است، اساس عرضه‌ی سازمانی Borland را از حالا به بعد شکل خواهد داد.

یک نکته جالب دیگر که در کنفرانس سالانه Eclipse، Borland اعلام نمود که مجموعه‌ای از Eclipse های plug-in را عرضه خواهد کرد که قابلیت‌های JBuilder 2005 را افزایش خواهد داد. این شرکت تا چه زمانی به پشتیبانی دو GUI برای یک محصول ادامه خواهد داد مشخص نیست، اما از این اعلام این گونه به نظر می‌رسد که احتمالاً JBuilder 2005 در نهایت به سوی Eclipse گام بر خواهد داشت. اگر این اتفاق رخ دهد، آن یک IDE عالی را رها خواهد ساخت، IDE ای که Borland قابلیت‌های بسیاری را به آن افزوده است.

IBM Rational Software Architect

خط جدید محصولات Rational Software شرکت IBM جایگزین خانواده WebSphere Studio می‌گردد (Rational Software Architect) RSA. نسخه ۶ بر خلاف شماره

نسخه‌اش نخستین نسل تحت نام جدید است. از میان مجموعه‌های متعددی که که مورد بررسی قرار ، RSA جامع‌ترین گزینه است.

IBM نیز مانند Borland از مدل مبتنی بر وظیفه استفاده می‌کند. RSA 6.0 به صورت چند لایه ساخته شده است. نخستین لایه Rational Web Developer است، که بخش مهمی از قابلیت مدلسازی را با خود ندارد؛ زیر آن Rational Application Developer قرار دارد، که فاقد قابلیت‌های طراحی وب‌نما است. در پایین‌ترین سطح نیز Eclipse 3.0 قرار دارد، IDE مجانی جاواگرا که در حال کسب محبوبیت بسیار زیادی است.

متأسفانه، تقریباً تمامی کاستی‌های RSA ها ناشی از این لایه پایینی است. اول از همه این که Eclipse یک رابط بصری نیست. تا وقتی که شما در طی یک مدت زمان طولانی با آن آشنا گردید، به احتمال زیاد به پنجره‌های غیر منتظره و بن‌بست بر خواهید خورد. IBM کار را با مجموعه‌ای از آموختارهای عالی ساده می‌نماید، اما همچنان پیمایش IDE در مقایسه با JBuilder 2005 یا Oracle JDeveloper دشوارتر است.

رابط RSA همچنین به میزان قابل ملاحظه‌ای کندتر از دو محصول مذکور است. درست است که هرگز به سطحی نمی‌رسد که کاربران را از خود ناامید سازد، اما کندی آن محسوس است و فاقد سرعت موجود در سایر محصولات است، به خصوص در هنگام سوییچ میان view های یک پروژه (همچون رفتن از طراحی به کدنویسی).

بخشی از این کندی ناشی از میزان نرم‌افزاری است که IBM به دور هسته Eclipse گرد آورده است. مجموعه قابل توجهی از نرم‌افزارهای سازمان‌گرا موجود بر روی ۱۴ عدد CD. این سرور کاربردی WebSphere شرکت IBM را به عنوان یک محیط تست، یک مجموعه نرم‌افزاری کامل برای توسعه پورتال‌های وب، و مدلسازی عالی و ابزارهای طراحی در بر می‌گیرد.

ابزارهای مدلسازی ۹ نمودار UML را پشتیبانی می‌نماید. بیشتر از محصولات Borland و Oracle. این نمودارها می‌توانند برای الگوهایی که در بر می‌گیرند یا ممکن است در بر گیرند تحلیل گردند، همچنین برای چیزی که IBM آن را ضد الگوها می‌نامد که در واقع اشکالات موجود در طراحی برنامه می‌باشند. برای مثال، کلاس‌هایی که ضعیف طراحی شده باشند با توصیف‌هایی از الگوهای طراحی مختل شده و نشانگرهایی که نشان می‌دهند چه چیز بایستی اصلاح گردد.

در ترکیب با تحلیل کد مبتنی بر وظیفه IBM ، این ابزارها به یک معمار کمک می‌کند که بیند پروژه‌های خوب چگونه پیاده‌سازی گردیده‌اند و چگونه آنها با راهنماهای طراحی و نیازمندی‌های سایت مطابقت دارند.

در حال حاضر تکنولوژی رابط وب برگزیده Rational JSF ، است. یک تکنولوژی که فرایند پیاده‌سازی را تسهیل می‌نماید. ابزارهای RSA شامل یک ویرایشگر WYSIWYG برای JSF است که توسط SDO ها (service data object) برای رابط‌های پایگاه داده پشتیبانی می‌گردد.

RSA دارای پشتیبانی جزئی برای C/C++ است، به علاوه پشتیبانی کامل برای جاوا. ابزارهای مدلسازی می‌توانند کار تبدیل به C++ را انجام دهند و ابزارهای متعدد سورس‌کد می‌توانند C++ را مورد تحلیل قرار دهند. به هر حال، IDE C++ فاقد یک کامپایلر و دیباگر است، یعنی باید جداگانه تهیه و نصب گردند. اگر شما یکی از این

ابزارها را در اختیار دارید می‌توانید آن را نصب نمایید، یا می‌توانید کامپایلرهای GNU C++ را برای این منظور دانلود کنید. این مشخصه‌ها همانند یک افزودنی عجیب و ناقص به نظر می‌رسند.

IBM RSA در این بررسی غنی‌ترین محصول از لحاظ دارا بودن مشخصه‌های متعدد است. برای معماران سازمانی که خواهان تسلط یافتن بر رابط و ابزارها هستند، آن یک محصول برگزیده توسعه جاوا محسوب می‌گردد.

Oracle JDeveloper

Oracle از به کار بردن مدل نسخه‌های مبتنی بر وظیفه‌ی IBM و Borland اجتناب ورزیده است و یک محصول قیمت پایین ارائه داده است. این شرکت از GUI خاص خود استفاده کرده که دارای یک طراحی بصری است.

JDeveloper تنها محصولی است که ظاهری زیبا داشته و محیطی تعاملی می‌باشد. زمانبندی تأخیرهای آغاز و سایر توابع بیانگر این مطلب است که کارایی آن در یک ارتباط اساسی با Borland Jbuilder است، اندکی جلوتر از محصول Sun، و به میزان قابل توجهی سریع‌تر از RSA محصول شرکت IBM است. در سطح تعامل شخصی با IDE، Oracle سریع‌تر به نظر می‌رسد.

دستیابی به بسیاری از مشخصه‌های JDeveloper در مقایسه با محصولات رقیب ساده‌تر است، که این امر انجام کارها را سرعت می‌بخشد.

Oracle همچنین یک سری موارد الحاقی جالب توجه را فراهم می‌آورد. نخستین مورد ابزاری است که یک تحلیل زمان اجرا از کد شما انجام می‌دهد. تحلیل مزبور بر اساس بررسی خود پیشنهادهایی را درباره کلاس‌هایی که می‌توانند نهایی گردند ارائه می‌دهد. این پیشنهادات فراتر از موارد معمولی هستند که تمامی IDE ها (شامل Oracle) همچنان که شما کد را تایپ می‌کنید فراهم می‌سازند، مواردی از قبیل نحوه بهینه‌سازی دستورات ورودی. در نسخه ویندوز، JDeveloper شامل JVM خاص خود است، که برای اشکال‌زدایی بهینه گردیده است. (در هر صورت، برای اهداف گسترش، Oracle به JVM سیستم رجوع می‌نماید).

برای کار با سرویس‌های وب، JDeveloper یک ناظر TCP را فراهم می‌نماید که توسعه‌گران را قادر به بررسی تک‌تک پکت‌ها می‌سازد مثل ثبت انتقال آنها و آشکار ساختن داده‌های آنها.

سایر ابزارها در مجموعه توسعه نرم‌افزار Oracle پشتیبانی سطح بالاتری را برای سرویس‌های وب (از قبیل orchestration و BPEL یا Business Process Execution Language) فراهم می‌آورند.

محدودیت عمده JDeveloper در مدلسازی UML است، جایی که محصول تنها چهار نمودار اصلی (usecase, activity, class, sequence) را پشتیبانی می‌نماید. JDeveloper نمودارهای متعدد غیر UML از قبیل طراحی نمودار EJB و Struts را پشتیبانی می‌کند.

در حالی که این محصول از لحاظ تعداد مشخصه‌های ارائه شده در معماری سازمانی به پای IBM و Borland نمی‌رسد، Oracle JDeveloper تمامی قابلیت‌هایی را که اغلب توسعه‌گران نیاز دارند فراهم می‌نماید. و با وجود مزیت قابل توجه قیمت آن در مقایسه با دو رقیب دیگر، آن به احتمال قریب به یقین می‌تواند محصول برگزیده برای بسیاری از سایت‌ها قلمداد گردد.

Sun Java Studio Enterprise

در دهه ۹۰، تعداد اندکی از فروشندگان اقدام به عرضه ابزارهای توسعه همپای Sun نمودند. Sun نخستین شرکتی بود که نوآوری‌های بسیاری را ایجاد نمود، از قبیل توانایی تغییر کد در دیباگر و ادامه اجرا.

هنگامی Sun سراغ ابزارهای توسعه جاوا رفت قافیه را باخت، و به سایر فروشندگان اجازه داد بازاری را که در واقع متعلق به خودش بود از آن خود نمایند. عرضه Java (JSE Studio Enterprise) 7 مخصوصاً با هدف حضور مجدد Sun در بازار ابزارهای جاوا صورت گرفت. این شرکت تلاش فراوانی را صرف مهیا ساختن این محصول نمود و، در عمل، آن مشخصه‌های متعدد منحصر بفردی را عرضه کرد. JSE بر پایه NetBeans بنا شده است، پلاتفرم کدبازی که با Eclipse رقابت می‌نماید. با وجود غلبه Eclipse، NetBeans یک پلاتفرم با ارزش است. که توانایی انجام تمام کارهایی که Eclipse می‌تواند انجام دهد را دارد. و همانند Eclipse، NetBeans از پشتیبانی توسعه‌گران plug-in برخوردار است، اگر چه Eclipse دارای تعداد بیشتری پروژه‌های فعال plug-in است.

از بسیاری جهات، JSE منحصر بفرد است: آن مشخصه‌های قابل توجهی را که سایر بسته‌ها فاقد آنها هستند فراهم می‌نماید، و آن فاقد برخی ابزارهایی است که سایر بسته‌ها ارائه می‌دهند. دو مشخصه بی‌نظیر و قابل ستایش JSE عبارتند از طرح‌ریزی اجرا و همکاری.

کارکرد "همکاری" تمامی توسعه‌گران را با استفاده از JSE در یک جلسه خاص مشابه IBM قرار می‌دهد و از این رو آنها می‌توانند پیغام‌ها و کد را با یکدیگر تبادل نمایند. پشتیبانی Whiteboarding نیز وجود دارد، و کانال‌های مجزایی برای مباحثات خصوصی و عمومی در آن موجود است.

هنگامی که تنظیمات انجام شد، مشخصه همکاری به صورت خودکار در هنگام اجرای JSE اجرا می‌گردد، از این رو با نشستن برای انجام کار همه اعضای یک گروه فوراً وارد تیم می‌شوند.

مشخصه جذاب دیگر JSE شبیه‌ساز لود آن است. تست نمودن برنامه‌های توزیع‌شده کار دشواری است، آنها اغلب نیازمند تنظیم‌های پیچیده به منظور بازتولید لودهایی هستند که توانایی بازنمایی فعالیت دنیای واقعی را دارند. قادر بودن به تست کارایی از درون JSE (با استفاده از تست لود داخلی آن) به معنی صرفه‌جویی زمانی قابل توجه برای توسعه‌گرانی است که بر روی برنامه‌های سازمانی کار می‌کنند.

پشتیبانی Sun از نمودارهای بیشمار UML اگر چه منحصر بفرد نیست اما جذاب است. از این لحاظ تنها RSA محصول IBM با آن برابری می‌نماید.

متأسفانه، از سایر لحاظ JSE دچار کاستی‌هایی است. آن Struts یا JSF را پشتیبانی نمی‌کند. در عوض، آن از WAF (Web Application Framework) خود SUN استفاده می‌نماید. آن یک جایگزین است که به سختی جذاب واقع خواهد شد، چرا که در حال حاضر تعداد بسیار زیادی تکنولوژی جاوا وجود دارد که در حوزه Web UI برنامه‌های سازمانی رقابت می‌کنند.

در سطح کدنویسی، JSE پیشنهادهایی را برای بهبود کد ارائه می‌دهد. نهایتاً، رابط دارای برخی جنبه‌های آشفته است، که این فکر را به ذهن می‌آورد که نسخه ویندوز JSE یک تبدیل مستقیم از نسخه Solaris است.

برای مثال، یک سری پنجره‌ها برخی مواقع از شما در مورد مکان سیستم‌های فایل mount شده یا نقاط اتصال سوال می‌کنند. این اصطلاحات در دنیای ویندوز وجود ندارند، اما در یونیکس معمول هستند. این مساله از آنجا ناشی می‌شود که سیستم help هیچ کاری برای یاری رساندن به توسعه‌گر ویندوز برای درک چیزی که JSE در این پنجره‌ها خواهان آن است انجام نمی‌دهد.

JSE 7 شرکت Sun را برای توسعه جاوای سازمانی در جایگاه مناسبی قرار می‌دهد. اما اگر شما نیازی به نسخه‌های بی‌نظیر آن ندارید که واقعا هم جذاب هستند، سایر محصولات که در اینجا مورد بررسی قرار گرفتند گزینه‌های بهتری خواهند بود.

قابلیت‌ها

در حال حاضر هیچ زبان برنامه‌نویسی دیگری دارای چنین محیط‌های قدرتمندی (که در اینجا بررسی شد) نیست. حتی Visual Studio .Net 2003 در قیاس با آنها کم فروغ جلوه می‌نماید، اگر چه انتظار می‌رود Visual Studio .Net 2005 این فاصله را کاهش دهد.

اما با وجود کیفیت و قابلیت‌های گسترده این چهار IDE، آنها فاقد عناصری هستند مدت‌ها پیش می‌بایست در آنها ایجاد می‌گردید.

جای برخی کارکردهای کدنویسی ساده خالی است. برای مثال، چرا نباید قادر باشیم لیترال‌ها یا بسته‌های منبع را از لحاظ املائی بررسی نماییم. همچنین، مشخصه‌های پیشرفته‌تر تنها حداقل پیاده‌سازی‌ها را دارند. به عنوان مثال، سازندگان GUI برای Oracle، Borland، و Sun تنها کد Swing را تولید می‌نمایند، تقریباً چنان که گویی Standard Widget Toolkit وجود ندارد.

به استثنای Borland، ویرایشگرهای XML هنگامی که می‌بایست قدرتمند باشند ضعیف ظاهر گردیدند، و همچنین ویرایشگرهای HTML هیچ محصولی توانایی تولید تست‌های مهم واحد به روشی که ابزارهای واسط انجام می‌دهند را ندارد. در عوض، این IDE ها بنیان‌ها (stub ها) را فقط برای JUnit تولید می‌کنند، حتی هنگامی که تست‌ها برای یک کلاس خاص واضح هستند.

ارزیابی گزینه‌های IDE

تنها راه برای خریداری تکنولوژی چه سخت‌افزار باشد و چه نرم‌افزار شناخت صحیح نیازهای خود است. شما در مورد IDE های جاوا ناگزیر به پیمودن این گام مقدماتی خواهید بود، چرا که این چهار محصول همگی به خوبی طراحی و پیاده‌سازی شده‌اند.

یک ارتباط طبیعی میان محصولات Borland و IBM وجود دارد، زیرا آنها بسته‌هایی هستند که پروژه‌های بزرگ سازمانی را با پشتیبانی خاص برای معماران نرم‌افزار مد

نظر قرار داده‌اند. در مورد هر دو، محصولات اضافی که توسط این فروشندگان به فروش می‌رسد می‌تواند قابلیت‌های بیشتری را فراهم آورند.

از بین این دو، IBM دارای مشخصه‌های کامل‌تری می‌باشد، اما آن قدری کندتر اجرا می‌گردد و رابط آن کمتر از Borland بصری است.

اگر نیازهای شما تماماً در سطح معماری نیستند، در این صورت JDeveloper شرکت Oracle انتخاب خوبی به شمار می‌آید، چون به خوبی با ویندوز کارکرده و اگر چه Borland نیز در این حوزه رقیب شایسته‌ای می‌باشد. اگر قیمت را در نظر بگیریم، در این صورت Oracle برنده‌ی رقابت است. فقط Sun JSE 7 می‌تواند برای آن دسته از توسعه‌گرانی که نیازمند مشخصه‌های بی‌نظیر آن یعنی تست بارگذاری، همکاری گروهی، و طرح‌ریزی پروژه‌های بزرگ هستند توصیه گردد.

هر چهار فروشنده نسخه‌های مخصوص ارزیابی را برای دانلود مجانی در اختیار عموم قرار داده‌اند تا در صورت تمایل بتوانید اجرای آزمایشی آنها را تجربه نمایید. به هر حال، نصب، تنظیم، و تست این محصولات کار ساده‌ای نیست. پیشنهاد این است که شما با محصول Oracle کار خود را آغاز نمایید، که نصب آن ساده‌تر از سایرین است، و در اغلب موارد بیشتر چیزهایی که نیاز دارید را فراهم می‌نماید. اگر محدودیت‌های مدلسازی JDeveloper شرکت Oracle شما را محدود می‌سازد، توصیه می‌شود JBuilder شرکت Borland (Enterprise Edition) یا Rational RSA شرکت IBM را دانلود کنید منبع:

آشنای مقدماتی با جاوا و قواعد نحوی آن

اعلان متغیر :

بصورت `type identifier [=value]` در آن **Identifier** نام متغیر و **type** نوع داده آن در جاوا می‌باشد. می‌توانید با گذاشتن `=` مقدار داخواه آن را مقدار دهی اولیه نماید. با استفاده از کاما می‌توانید بیش از یک متغیر بایک نوع تعریف کنید

توضیحات :

برای خوانا تر شدن برنامه خود می‌توانید از توضیحات در برنامه خود استفاده کنید همچنین وقتی برنامه شما بزرگ بوده فهم آن را ساده تر میکند. برای این کار می‌توانید از `//` یا از `/* */` برای توضیحات چند خطی استفاده کنید.

```
int d = 3, e, f = 5; // declares three more ints/ initializing // d and f.
byte z = 22; // initializes z.
double pi = 3.14159; // declares an approximation of pi.
char x = 'x'; // the variable x has the value 'x'.
```

قلمرو و زمان حیاط متغیر ها :

جاوا به متغیر ها اجازه می دهد تا در درون یک بلوک تعریف شود (داخل { }) پس هر بار که یک بلوک جدید را شروع میکنید یک قلمرو جدید ایجاد میشود و در داخل آنها اشیاء قابل رویت و زمان حیاط آنها مشخص میشود . تعریف اشیاء میتواند بصورت سراسری یا محلی باشد.

اگر یک متد دارای پارامتر های ورودی باشد همانند یک متغیر از آن استفاده میشود . اگر برنامه شامل بلوکهای مختلف تودر تو باشد اشیاءی اعلان شده در بلوک بیرونی برای کدهای بلوک درونی قابل رویت میباشد اما معکوس این قضیه درست نمی باشد .

همانطور که گفته شد زمان حیاط یک متغیر محدود به به آن بلوک میباشد و اگر اعلان یک متغیر شامل مقدار دهی اولیه باشد انگاه هر زمان که به بلوک وارد میشویم آن متغیر مقدار دهی اولیه میشود .

تبدیل خودکار :

برای اینکار میباید نوع مقصد از نوع منبع باشد و همچنین آن دو با هم سازگار باشند مثلاً نوع `int` با `char` یا `Boolean` سازگار نیستند بطور مثال اگر `i` از نوع `int` و `L` از نوع `long` باشد دستور `L=i` ; درست می باشد.

تبدیل غیر خودکار انواع ناسازگار:

استفاده آن در جای که بطور مثال می خواهید مقدار `int a` را به یک `byte b` نسبت دهید برای اینکار از تبدیل `target =(target type)value;` که به فرم `b=(byte)a;` میشود.

boolean	True , False	void	بدون نوع یا null
byte	۱ بایت	char	۲ بایتی کارکتر
Short	۲ بایت صحیح	float	۴ بایت اعشاری
long	۸ بایت صحیح	double	۸ بایت اعشاری
int	۴ بایت صحیح	string	رشته طول متغیر

عملگر های منطقی و بولی:

این عملگر ها بر روی عملوندهای بولی عمل می کنند و دو مقدار `Boolean` را ترکیب می کنند تا یک مقدار `Boolean` ایجاد شود.

AND با &

OR با |

XOR با ^

NOT با !

آنها ی که در بالا گفته شد برای عملگر های بیتی بود و برای `Boolean` :

AND با &&

OR با ||

مساوی با ==

نامساوی با !=

خروجی برنامه زیر مثالها را بهتر شرح میدهد.

```
class BoolLogic {
    public static void main(String args[] ){
```

```

boolean a = true;
boolean b = false;
boolean c = a | b;
boolean d = a & b;
boolean e = a ^ b;
boolean f = ( !a & b ) | ( a & !b );
boolean g = !a;
System.out.println(" a = " + a);
System.out.println(" b = " + b);
System.out.println(" a | b = " + c);
System.out.println(" a&b = " + d);
System.out.println(" a^b = " + e);
System.out.println("!a&b | a&!b = " + f);
System.out.println(" !a = " + g);
}
}

```

خروجی:

```

a = true
b = false
a | b = true
a&b = false

a^b = true
!a&b | a&!b = true
!a = false

```

اعلان اشیاء :

برای اینکار باید از نوع آن شیء یک متغیر تعریف کرده سپس باید یک نمونه فیزیکی از شیء بدست آورده و با کمک متغیر با آن کار کنید. برای گرفتن شیء فیزیکی در حافظه Ram باید از عملگر new استفاده کنید. این عملگر حافظه به شیء تخصیص میدهد و و یک ارجاع (آدرس حافظه آن شیء) را به متغیر تعریف شده از آن نوع را برمیگرداند. در زبان جاوا کلیه اشیاء کلاس می باید بصورت پویا تخصیص یابند.

```
Box mybox; // declare reference to object
```

```
mybox = new Box(); // allocate a Box object
```

ابتدا متغیر mybox به عنوان اشاره گر به شیء Box تعریف میکنیم که مقدار اولیه آن (null) یا تهی خواهد بود یعنی هنوز به شیء واقعی اشاره نمی کند. در خط بعدی به mybox را به یک شیء واقعی انتساب می دهد.

شکل عمومی عملگر new :

شکل عمومی به صورت

```
class-var = new classname();
```

در اینجا class-var متغیر از نوع کلاس مورد نظر و classname نام کلاسی که میخواهیم از آن شیء معرفی کنیم classname() به عنوان سازنده کلاس به کار می رود. سازنده

(constructor) در کلاسها به عنوان تعریف کننده نحوه ایجاد شی می باشد و باید صریح در داخل کلاس خود معرفی شده باشد. اگر در داخل کلاس خود سازنده صریح موجود نباشد جاوا یک سازنده پیش فرض برای آن ارائه میکند .
مزیت استفاده از new در این است که برنامه در زمان اجرا هر وقت که نیاز به شی آنرا ایجاد می کند، توجه کنید که اگر به علت کمبود حافظه new نتواند حافظه تخصیص یک استثنا در زمان اجرا رخ می دهد.

عملگرها

عملگر های جاوا را میتوان در چهارگروه دسته بندی کرد : عملگرهای حسابی ، بیتی ، رابطه ای و منطقی . اگر با زبان C / C++ آشنا باشید کاربرد عملگرها دقیقاً همانند کاربرد آنها در جاوا است .

عملگرهای حسابی (Arithmetic)

این عملگر ها در عبارات ریاضی روی اعداد استفاده می شوند
+ جمع

- منها

* ضرب

/ تقسیم

% باقیمانده

++ افزایش یک واحد

-- کاهش یک واحد

+= انتساب به همراه اضافه کردن

-= انتساب به همراه تفریق کردن

*= انتساب به همراه ضرب کردن

/= انتساب به همراه تقسیم کردن

%= انتساب به همراه تعیین باقیمانده کردن

برخی از این عملگرها روی نوع char قابل استفاده اند. اما نمی توان روی boolean استفاده کرد. در عبارات ریاضی عملگر ها با اولویت یکسان بسته به ترتیب آنها از سمت چپ پردازش میشوند. اگر بخواهیم عملگر با اولویت کمتر زود تر از عملگر با اولویت بالاتر پردازش شود باید آن را در داخل دو پرانتز () گذاشت. مثالهای زیر نحوه استفاده از عملگر ها را نشان میدهد .

```
class BasicMath {
public static void main(String args[] ){
// arithmetic using integers
System.out.println("Integer Arithmetic");
```

```

int a = 1 + 1;
int a = a * 3;
int a = b / 4;
int a = c - a;
int a = - d;
System.out.println("a = " + a);
System.out.println("a = " + b);
System.out.println("a = " + c);
System.out.println("a = " + d);
System.out.println("a = " + e);
// arithmetic using doubles
System.out.println("\nFloating Point Arithmetic");
double da = 1 + 1;
double db = da * 3;
double dc = db / 4;
double dd = dc - a;
double de = - dd;
System.out.println("da = " + da);
System.out.println("db = " + db);
System.out.println("dc = " + dc);
System.out.println("dd = " + dd);
System.out.println("de = " + de);
} }

```

خرو جي برنامه به شکل زیر است

```

integer Arithmetic
a=2
b=6
c=1
d=-1
e=1
floating point arithmetic
da=2
db=6
dc=1.5
dd=-0.5
de=0.5

```

عملگرها بيتي (bitwise)

در جاوا چندین عملگر رفتار بیتي تعريف شده میباشد که روی انواع عددی و char اعمال می شوند
~ مکمل بیتي

& and بیتي

| or بیتی

^ xor بیتی

>> شیفت به راست بیتی و با صفر پر شدن

<< شیفت به چپ بیتی و با صفر پر شدن

&= انتساب و and بیتی

|= انتساب و or بیتی

^= انتساب و xor بیتی

=>> انتساب و شیفت به راست بیتی و با صفر پر شدن

=<< انتساب و شیفت به چپ بیتی و با صفر پر شدن

به کمک مثال طرز کار این عملگر ها را میتوان فهمید .

```
class BitLogic {
public static void main(String args[] ){
String binary[] = {
"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
"1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
};
int a = 3; // 0 + 2 + 1 or 0011 in binary
int b = 6; // 4 + 2 + 0 or 0110 in binary
int c = a | b;
int d = a & b;
int e = a ^ b;
int f =( ~a & b )|( a & ~b);
int g = ~a & 0x0f;
System.out.println(" a = " + binary[a]);
System.out.println(" b = " + binary[b]);
System.out.println(" a|b = " + binary[c]);
System.out.println(" a&b = " + binary[d]);
System.out.println(" a^b = " + binary[e]);
System.out.println("~a&b|a&~b = " + binary[f]);
System.out.println(" ~a = " + binary[g]);
} }
```

خروجی به شکل زیر است .

```
a=1011
b=0110
a^Eb=0111
a&b=0010
a&b=0101
```

```
~a&b^Ea&~b=0101
```

```
~a=1100
```

عملگر انتساب (Assignment)

علامت = و شکل کلی `var=expression;` در اینجا نوع متغیر `var` باید با خروجی عبارت `expression` سازگار باشد. در زبانهای `C/C++` و جاوا این امکان وجود دارد که تا زنجیره ای از انتسابها رابه وجود آورید .

```
int x, y, z;
```

```
x = y = z = 100; // set x, y, and z to 100
```

قوانین ارتقاء انواع

علاو بر ارتقاء `short, byte` به `int` جاوا چندین قانون ارتقا انواع را تعریف کرده که در عبارات قابل استفاده اند. قانون اول اینکه همان طور که گفته شد کلیه مقادیر `short, byte` به `int` ارتقاء می یابند. انگاه اگر یک عملوند از نوع `long` باشد کل عبارت به `long` ارتقاء می یابد. اگر یک عملوند از نوع `float` باشد کل عبارت به `float` ارتقاء می یابد. این قانون در مورد `double` نیز صادق می باشد. برنامه زیر نشان می دهد که چگونه هر یک از مقادیر در عبارت ارتقاء می یابد تا با آرگمان دوم به هر یک از عملگرها دودویی، مطابقت یابد.

```
class Promote {
    public static void main(String args[] ){
        byte b = 42;
        char c = 'a';
        short s = 1024;
        int i = 50000;

        float f = 5.67f;
        double d = . 1234;
        double result =( f * b )+( i / c ( - )d * s);
        System.out.println((f * b )+ " * " +( i / c )+ " - " +( d * s));
        System.out.println("result = " + result);
    }
}
```

در عبارت `double result =( f * b )+( i / c ( - )d * s);` حاصل `f*b` ، `b` به `float` ارتقاء یافته و جواب عبارت نیز از نوع `float` خواهد بود. در عبارت بعدی یعنی `i/c` ، `c` به نوع `int` ارتقاء یافته و جواب از نوع `int` خواهد بود. نتیجه عبارت `d*s` ، نیز از نوع `double` خواهد بود. در نهایت چون نوع `double` از سایر انواع بزرگتر می باشد جواب نهایی از نوع `double` خواهد بود.

عملگر ؟

همانند زبان `C++` در جاوا میتوانیم از علامت ؟ برای عملگر سه تایی ویژه برای جایگزینی انواع مشخصی از دستورات `if-then-else` استفاده کنیم. شکل کلی این عملگر به فرم زیر است.

```
experssion1 ? experssion2 : experssion3
```

در اینجا `experssion1` هر عبارت بولي ميتواند باشد. اگر `experssion1` صحيح باشد عبارت `experssion2` اجرا خواهد شد در غير اينصورت `experssion3` اجرا ميشود. در زیر مثال استفاده از `?` بيان شده.

```
ratio = denom == 0 ? 0 : num / denom;
```

زمانی که جاوا این عبارت انتساب را ارزیابی می کند ابتدا عبارت سمت چپ علامت سوال بررسی می شود اگر `denom` برابر با صفر باشد آنگاه عبارت بین علامت سوال و علامت `(colon)` ارزیابی می شود و به عنوان مقدار کل عبارت `?` استفاده می شود. حال اگر `denom` مساوی با صفر نباشد آنگاه عبارت بعد از `(colon)` ارزیابی شده و برای مقدار کل عبارت `?` استفاده می شود. نتیجه تولید شده توسط عملگر `?` سپس به `ratio` نسبت داده می شود. در برنامه زیر نحوه استفاده از `?` را برای قدر مطلق مشاهده می کنید.

```
class Ternary {
    public static void main(String args[]){
        int i/ k;

        i = 10;
        k = i < 0 ? -i : i; // get absolute value of i
        System.out.print("Absolute value of ");
        System.out.println(i + " is " + k);
    }
}
```

خروجی برنامه به شکل زیر می باشد.

Absolute value of -10 is 10

کلاسها در جاوا

در زمانی که یک کلاس را تعریف می کنید در حقیقت شکل و طبیعت آن شی را به کمک متدها و صفات آن را اعلان می کنید برای اعلان یک کلاس از واژه کلیدی `class` استفاده می کنیم و کلاسهای که تا به کنون از آنها استفاده کرده اید کلاسهای محدودی از شکل کامل کلاسها بوده اند.

```
Class class_name {

    Member_Variables

    Member_Method
}
```

داده ها یا متغیرهای تعریف شده در داخل یک کلاس را متغیرهای نمونه (`instance variables`) می نامند. این نام گذاری به این دلیل است که هر نمونه از کلاس شامل یک کپی از این متغیرهای خاص خودش است. بنا براین داده های مربوط به یک شی ، جدا و منحصر به فرد از داده های اشیاء دیگر است.

کدهای کلاس که روی متغیرهای نمونه اعمال را انجام میدهند را در داخل متدهای آن کلاس قرار می دهند. این اعمال می تواند اعمال محاسباتی ، اعمال نمایش خروجی ، اعمال تبدیل و غیره باشند. متد main() را زمانی در کلاسها استفاده می کنند که نیاز به اجرای آن کلاس توسط کامپایلر جاوا دارید و یا آن کلاس نقطه شروع در برنامه شما باشد.

در این جا يك مثال ساده از کلاس نشان داده شده. نام این کلاس Box می باشد و سه متغیر نمونه در داخل خود دارد و متدی در داخل آن پیاده سازی نشده.

```
class Box {
double width;
double height;
double depth;
}
```

هر کلاس زمانی که تعریف می شود يك نوع جدیدی از داده ها را ایجاد می کند و در مثال ما این نوع جدید Box است. توجه داشته باشید که اعلان يك کلاس يك شی واقعی را به وجود نمی آورد و فقط الگو را ایجاد می کند. برای ایجاد يك شی واقعی از نوع Box باید از دستور زیر استفاده کنید.

```
Box mybox = new Box(); // create a Box object called mybox
```

پس از اجرا mybox نمونه ای از Box خواهد بود. به منظور دسترسی به متغیرهای نمونه شی از عملگر نقطه استفاده کنید. این عملگر نام يك شی را اب يك متغیر نمونه پیوند می دهد. به عنوان مثال برای انتساب مقدار 100 به متغیر width از دستور زیر باید استفاده کرد.

```
mybox.width = 100;
```

بطور کلی عملگر نقطه برای دسترسی به متغیر نمونه و هم برای متدهای موجود در شی استفاده می شود. در اینجا يك برنامه کامل که از کلاس Box استفاده می کند را مشاهده می کنید.

```
/* A program that uses the Box class.
Call this file BoxDemo.java
*/
class Box {
double width;
double height;
double depth;
}
// This class declares an object of type Box.
class BoxDemo {
public static void main(String args[] ){
Box mybox = new Box();
double vol;
// assign values to mybox's instance variables
mybox.width = 10;
mybox.height = 20;
mybox.depth = 15;
// compute volume of box
vol = mybox.width * mybox.height * mybox.depth;
System.out.println("Volume is " + vol);
```

```
}
}
```

حال برنامه نوشته شده را در فایلی به نام BoxDemo.java قرار بدهید. دلیل این کار هم مشخص است و این اینکه متد main() در داخل این کلاس قرار دارد و کامپایلر جاوا به طور خود کار آنرا اجرا خواهد کرد.

بعد کامپایل کردن دو فایل با پسوند class، ایجاد خواهند شد که یکی از آن دو برای کلاس Box و دیگری برای BoxDemo. ضرورتی برای این وجود ندارد که هر دو کلاس BoxDemo, Box در داخل يك فایل منبع قرار بگیرند. این امکان وجود دارد که هر کلاس را در فایل جدا مخصوص خود گذاشته و آنها را به ترتیب Box.java و BoxDemo.java نامگذاری کنید. برای اجرای این برنامه BoxDemo.class را اجرا کنید. پس از اجرا خروجی زیر حاصل خواهد شد.

Volume is 3000

همان طور که قبلاً اشاره کردیم هر شی دارای کپی نمونه های خاص خود است و اگر دو نمونه از شی Box داشته باشیم هر کدام متغیر های خود را خواهند داشت. تغییر روی هر کدام از آنها تاثیری بر روی دیگری نداشته. این نکته را مثال زیر نشان می دهد.

```
// This program declares two Box objects.
class Box {
double width;
double height;
double depth;
}
class BoxDemo2 {
public static void main(String args[] ){
Box mybox1 = new Box();
Box mybox2 = new Box();
double vol;
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's
instance variables */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
// compute volume of first box
vol = mybox1.width * mybox1.height * mybox1.depth;
System.out.println("Volume is " + vol);
// compute volume of second box
vol = mybox2.width * mybox2.height * mybox2.depth;
System.out.println("Volume is " + vol);
}
}
```

خروجی تولید شده به فرم زیر خواهد بود.

Volume is 3000

Volume is 162

تعریف کنترل دسترسی

همان طور که در شی گرای گفته شده قابلیت کپسوله سازی داده را با کدی که با آن داده سرو کار دارد پیوند میدهد. خطالت مهم دیگر کپسوله سازی کنترل دستیابی (access control) است. به کمک کپسوله سازی می توانید کنترل کنید که چه بخشهایی از یک برنامه می توانند به اعضای یک کلاس دسترسی داشته باشند و از سوء استفاده جلوگیری کنید. اجازه دادن دسترسی به دادهها فقط از طریق یک مجموعه خوش تعریف از روشها مانع سوء استفاده از داده های آنها می شود. به کار ساخت جعبه سیاه (black box) می گویند که از آن کلاس استفاده می کنیم ولی از نحوه کار در داخل آن خبر نداریم.

چگونه گی دسترسی به یک متغیر یا متد عضو کلاس توسط توصیف گر دسترسی تعریف می شود. جاوا مجموعه غنی از توصیف گر دسترسی را در اختیار دارد. برخی جوانب کنترل دسترسی به شدت با وراثت و بسته ها (packages) مرتبط هستند. بسته ها یک نوع گروچ بندی برای کلاسها است. اگر یک بار پایه کنترل دسترسی را فرا بگیرید آنگاه بقیه مطالب را به راحتی خواهید آموخت.

توصیف گر های دسترسی در جاوا با کلمات کلیدی public, private, protected مشخص می شوند. همچنین جاوا به طور پیش فرز از protected برای سطح دسترسی در وراثت استفاده می کند.

اگر یک عضو کلاس را با public توصیف کنیم، آن عضو توسط هر کد دیگری در برنامه قابل دسترس خواهد بود. اگر یک عضو کلاس را با private توصیف کنیم، آن عضو توسط پس از آن فقط توسط متدهای عضو آن کلاس قابل دسترس خواهد بود. اکنون متوجه شده اید که چرا در قبل از متد main() توصیف گر public قرار می گیرد. این متد توسط اجرا کننده برنامه های جاوا یعنی فایل اجرای java.exe فراخوانی خواهد شد. به طور پیش فرز اگر از توصیف گر در تعریف اعضاء استفاده نکنیم، آنها در کلاسهای مربوط به بسته ای (package) که آن کلاس داخل آن قرار دارد قابل دسترس خواهند بود اما در بیرون از بسته قابل دسترسی نمی باشند.

در کلاسهای که قبلاً استفاده کردیم به طور پیش فرز از public استفاده کرده اند. شما معمولاً مایل هستید که دسترسی به اعضای داده های یک کلاس را محدود کنید فقط برخی متدها قابل دسترس باشند. همچنین شرایطی وجود دارند که مایلید که متدهای که مخصوص یک کلاس اختصاص می باشند را توسعه دهید. یک توصیف گر دسترسی قبل از سایر مشخصات نوع عضو یک کلاس قرار می گیرند و این به این معنی است که توصیفگر باید شروع کننده دستور اعلان یک عضو باشد. در زیر یک مثال را مشاهده می کنید.

```
public int i;
private bouble j;
private int myMethod(int a/ char b ){ //...
```

برای یدرک تاثیرات دسترسی عمومی (public) و اختصاصی (private) برنامه زیر را در نظر بگیرید.

```
/* This program demonstrates the difference between
public and private.
*/
class Test {
int a; // default access
public int b; // public access
private int c; // private access
```

```
// methods to access c
void setc(int i ){ // set c's value
c = i;
}
int getc (){ // get c's value
return c;
} }
class AccessTest {
public static void main(String args[] ){
Test ob = new Test();
// These are OK/ a and b may be accessed directly
ob.a = 10;
ob.b = 20;
// This is not OK and will cause an error
// ob.c = 100; // Error!
// You must access c through its methods
ob.setc(100); // OK
System.out.println("a/ b/ and c :" + ob.a + " " +
ob.b + " " + ob.getc());
}
}
```

همان طور که ملاحظه می کنید داخل کلاس Test، متغیر a از دسترسی پیش فرز استفاده می کند که در این مثل مطابق مشخص نمودن با public است. متغیر b به طور صریح با public مشخص شده . متغیر c دارای دسترسی اختصاصی است. این معنی است که توسط کدهای خارج از کلاس قابل دسترسی نخواهد بود و در داخل کلاس Access Test نمی توان c را به طور مستقیم مورد استفاده قرار داد. این عضو توسط متدهای عمومی آن یعنی getc() و setc() باید .ورد دسترسی قرار گیرد.

اگر دستور ob.c=100 را در داخل برنامه استفاده کنید کامپایلر از برنامه خطا خواهد گرفت و برنامه قابل اجرا نخواهد بود. برای اینکه با جنبه های عملی کنترل دسترسی آشنا شوید کلاس زیر برای ساخت يك stack ایجاد شده.

```
// This class defines an integer stack that can hold 10 values.
class Stack {
/* Now/ both stck and tos are private .This means
that they cannot be accidentally or maliciously
altered in a way that would be harmful to the stack.
*/
private int stck[] = new int[10];
private int tos;
// Initialize top-of-stack
Stack (){
tos =- 1;
}
// Push an item onto the stack
void push(int item ){
if(tos==9)
```

```

System.out.println("Stack is full.");
else
    stck[++tos] = item;
}
// Pop an item from the stack
int pop () {
    if(tos < 0) {
        System.out.println("Stack underflow.");
        return 0;
    }
    else
        return stck[tos--];
}
}

```

همان طور که ملاحظه می کنید هم آرایه stck (نگه دارنده پشته) و هم متغیر tos (که نمایان گر بالای پشته) به عنوان private معرفی شده اند. این به این معنی است که آنها قابل دسترسی و جایگزین نیستند. به عنوان مثال اختصاصی تعریف نمودن tos آن را در مقابل قرار دادن غیر عمدی مقداری که فراتر از انتهای آرایه stck است محافظت می کند. برنامه بعدی نشان دهنده استفاده از کلاس توسعه یافته stack است .

```

class TestStack {
    public static void main(String args[] ) {
        Stack mystack1 = new Stack();
        Stack mystack2 = new Stack();
        // push some numbers onto the stack
        for(int i=0; i<10; i++) mystack1.push(i);
        for(int i=10; i<20; i++) mystack2.push(i);
        // pop those numbers off the stack
        System.out.println("Stack in mystack1:");
        for(int i=0; i<10; i++)
            System.out.println(mystack1.pop());
        System.out.println("Stack in mystack2:");
        for(int i=0; i<10; i++)
            System.out.println(mystack2.pop());
        // these statements are not legal
        // mystack1.tos = - 2;
        // mystack2.stck[3] = 100;
    }
}

```

اگر چه متدها معمولاً دسترسی به داده های تعریف شده یک کلاس را کنترل می کنند، اما همیشه هم این طور نیست. کاملاً به جاست که هرگاه دلیل خوبی برای این کار داشته باشیم، اجازه دهیم تا یک متغیر داخل کلاس از دسترسی public استفاده کند. به عنوان مثال اکثر کلاسهای ساده با کمترین توجه نسبت به کنترل دسترسی به متغیرهای نمونه ایجاد شده اند و این بی توجهی فقط برای ساده بودن مثالها بوده.

محافظت دسترسی (Access Protection)

همان طور که می دانید دسترسی به یک عضو private در یک کلاس فقط به سایر اعضای آن کلاس محدود شده. بسته ها (package) قابلیت این را دارند که کنترل دیگری به برنامه اضافه کنند. همان طوری که خواهید دید جاوا سطوح مختلفی برای کنترل طبقه بندی شده و روییت پذیری متغیرها و متدها در داخل کلاسها و زیر کلاسها یی داخل بسته فراهم میکند.

بسته ابزاری برای کپسوله سازی بالاتر از سطح کپسوله سازی در کلاسها است و در بر گیرنده فضای نام و قلمرو متغیرها و متدها است. زمانی که تعداد کلاسها و اشیاء در توسعه برنامه های کاربردی افزایش پیدا می کند که امروزه در عمل بسیار افزایش یافته برای طبقه بندی آنها در سطح بالاتر از کلاس از بسته های جاوا استفاده میکنیم.

بسته ها به عنوان ظروفی برای برای کلاسها و سایر بسته های وابسته هستند. کلاسها به عنوان ظروفی برای داده ها و کدها می باشند و کوچکترین واحد مجرد سازی در جاوا هستند. به لحاظ نقش متقابل بین کلاسها و بسته ها ، جاوا چهار طبقه بندی برای روییت پذیری اعضای کلاس مشخص کرده:

- ۱- زیر کلاسها در همان بسته
- ۲- غیر زیر کلاسها در همان بسته
- ۳- زیر کلاسها در بسته های مختلف
- ۴- کلاسهای که نه در همان بسته و نه در زیر کلاسها هستند.

همچنین سه توصیفگر گفته شده یعنی public , private , protected را هم فراهم کرده. با این روشها می توانیم سطوح مختلفی از دسترسی ایجاد کنیم.

اگر چه مکانیزم کنترل دسترسی در جاوا ممکن است در ابتدا پیچیده باشد، اما می توان به شکل آنها توضیح داد. هر چیزی که در تعریف آن از public استفاده شود از هر جا قابل دسترسی می باشد. هر چیزی که در تعریف آن از private استفاده شود خارج از کلاس خودش قابل روییت نمی باشد. زمانی که یک عضو فاقد مشخصات دسترسی صریح باشد، آن عضو برای کلاسها و زیر کلاسهای موجود در بسته قابل روییت می باشد و به صورت دسترسی پیش فرض است. اگر شما خواستار آن هستید که یک عضو خواه مدت یا متغیر خارج از بسته جاری و فقط به کلاسهای مستقیماً از کلاس شما به صورت زیر کلاس در آمده اند قابل روییت باشد، باید آنرا با توصیفگر protected تعریف نماید.

یک کلاس فقط دو سطح دسترسی ممکن دارد: پیش فرض و عمومی . عمومی زمانی که یک کلاس به عنوان public اعلام می شود، توسط هر کد دیگری قابل دسترسی است . اگر کلاس دسترسی پیش فرض را داشته باشد، فقط توسط سایر کدهای داخل همان بسته قابل دسترسی خواهد بود.

یک مثال از دسترسی

این مثال کلیه ترکیبات مربوط به اصلاح گرهای کنترل دسترسی را نشان می دهد و دارای دو بسته و پنج کلاس است. به یاد داشته باشید که کلاسهای مربوط به دو بسته متفاوت، لازم می باشد که در دایرکتوریهای به نام p1 و p2 که نام بسته ها نیز می باشند ذخیره خواهند شد. در بسته اول سه کلاس Protection و Derived و samepackage محود است. کلاس اول یعنی Protection دارای چهار متغیر int که هر یک در حالتی مختلف مجاز تعریف شده اند.

متغیر n با حفاظت پیش فرض اعلان شده. m-pri به عنوان private و n-pro با protected همچنین n-pub به عنوان public تعریف شده.

کلاسهای بعدی در این بسته سعی می کنند که به متغیرهای نمونه از یک کلاس دسترسی پیدا کنند. خطهای که به لحاظ محدودیتهای دسترسی کامپایل نمی شوند با استفاده از توضیح خطی یعنی با // مشخص شده اند. قبل از هر یک از این خطوط توضیحی قرار دارد که مکانهای را که از آنجا این سطح از حفاظت اجازه دسترسی می یابد را فهرست می نماید.

دومین کلاس Derived یک زیر کلاس از Protection در همان بسته p1 است. این مثال دسترسی Drived را به متغیرهای Protection برقرار می کند به جز n-pri که یک متغیر از نوع دسترسی private است.

سومین کلاس Samepackage یک زیر کلاس از Protection نبوده اما در همان بسته p1 قرار دارد. بنابراین به کلیه متغیرها به جز n-pri دسترسی خواهد داشت.

```
package p1;
public class Protection {
    int n = 1;
    private int n_pri = 2;
    protected int n_pro = 3;
    public int n_pub = 4;
    public Protection () {
        System.out.println("base constructor");
        System.out.println("n = " + n);
        System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
    class Derived extends Protection {
        Derived () {
            System.out.println("derived constructor");
            System.out.println("n = " + n);
            // class only
            // System.out.println("n_pri = " + n_pri);
            System.out.println("n_pro = " + n_pro);
            System.out.println("n_pub = " + n_pub);
        }
        class SamePackage {
            SamePackage () {
                Protection p = new Protection();
                System.out.println("same package constructor");
                System.out.println("n = " + p.n);
                // class only
                // System.out.println("n_pri = " + p.n_pri);
                System.out.println("n_pro = " + p.n_pro);
                System.out.println("n_pub = " + p.n_pub);
            }
        }
    }
}
```

اکنون کد منبع یک بسته دیگر به نام p2 را ملاحظه می کنید. دو کلاس تعریف شده در p2 دو شرایطی را که توسط کنترل دسترسی تحت تاثیر قرار گرفته اند را پوشش داده است.

اولین کلاس یعنی Protection2 یک زیر کلاس از p1.Protection است. این کلاس دسترسی به کلیه متغیرهای مربوط به p1.Protection را به غیر از n-pri که از نوع private است بدست می آورد. به یاد داشته باشید که به طور پیش فرض فقط اجازه دسترسی از داخل کلاس یا بسته را می دهد نه از زیر کلاسهای بسته های اضافی. در نهایت کلاس OtherPackage فقط به یک متغیر n-pub که به عنوان public اعلان شده دسترسی دارد.

```
package p2;
class Protection2 extends p1.Protection {
    Protection2 () {
        System.out.println("derived other package constructor");
        // class or package only
        System.out.println("n = " + n);
        // class only
        // System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
    class OtherPackage {
        OtherPackage () {
            p1.Protection p = new p1.protection();
            System.out.println("other package contryctor");
            // class or package only
            System.out.println("n = " + p.n);
            // class only
            // System.out.println("n_pri = " + p.n_pri);
            // class/ subclass or package only
            // System.out.println("n_pro = " + p.n_pro);
            System.out.println("n_pub = " + p.n_pub);
        }
    }
}
```

اگر مایل به اجرای این دو بسته هستید دو فایل آزمایشی وجود دارد که می توانید از آنها استفاده نمایید. یکی از این فایلها را برای بسته p1 در اینجا نشان داده ایم.

```
// Demo package p1.
package p1;
// Instantiate the various classes in p1.
public class Demo {
    public static void main(String args[] ){
        Protection ob1 = new Protection();
        Derived ob2 = new Drived();
        SamePackage ob3 = new SamePackage();
    }
}
```

فایل آزمایشی برای p2 به فرم زیر می باشد.

```
// Demo package p1.
```

```

package p1;
// Instantiate the various classes in p1.
public class Demo {
public static void main(String args[] ){
Protection2 ob1 = new Protection2();
OtherPackage ob2 = new OtherPackage();
}
}

```

سازنده گان (Constructors)

برای مقدار دهی متغیرهای یک کلاس می توانیم آنها را تک تک مقدار دهی اولیه کنیم. اما این روش مشکل می باشد حتی اگر برای مقداری دهی متغیرهای یک کلاس تابعی را ایجاد کنیم، در زمان اجرا باید خود کد مربوط به فراخوانی آن را بنویسید. جاوا به اشیاء این امکان را می دهد تا در زمان ایجاد شدن خودشان را مقدار دهی اولیه کنند.

این مقدار دهی اولیه خودکار با استفاده از سازنده انجام میگیرد. سازنده به محض ایجاد یک شی آن را مقدار دهی می کند و نام آن معادل با نام همان کلاس می باشد. هنگامی که یک کلاس سازنده دارد، سازنده در زمان تعریف استفاده از کلاس و قبل از تکمیل new فراخوانی می شود. دقت داشته باشید که سازنده هیچ نوع بازگشتی حتی void ندارد و این به این دلیل است که مقدار بازگشتی سازنده یک کلاس از نوع خود کلاسی که در آن کامل شده است. در زیر مثال مربوط به کلاس Box را به همراه سازنده آن تکمیل تر کرده ایم.

```

/* Here/ Box uses a constructor to initialize the
dimensions of a box.
*/
class Box {
double width;
double height;
double depth;
// This is the constructor for Box.
Box () {
System.out.println("Constructing Box");
width = 10;
height = 10;
depth = 10;
}
// compute and return volume
double volume () {
return width * height * depth;
} }
class BoxDemo {
public static void main(String args[] ){
// declare/ allocate/ and initialize Box objects
Box mybox1 = new Box();
Box mybox2 = new Box();
}
}

```

```
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("volume is " + vol);
}
}
```

پس از اجراي اين برنامه خروجي به شکل زیر خواهد بود.

```
Constructing Box
Constructing Box
Volume is 1000
Volume is 1000
```

همان طوري که مشاهده مي کنيد mybox2 و mybox1 هنگام ايجاد شدن به وسيله Box() مقدار دهی اولیه مي شوند و چون سازنده به همه اشیاء ايجاد شده از نوع Box مقدار ابعاد اولیه 10*10*10 را مي دهد هر دو شیء تعريف شده ابعاد یکسني خواهند داشت.

متد println() در داخل Box فقط براي به منظور توصیف بهتر قرار گرفته. اکثر متدهای سازنده چیزی را نشان نمی دهند و فقط وظیفه دادن مقادیر اولیه را دارند. همان طوري که مي دانيد استفاده از عملگر new در تخصیص يك شیء ايجاد شده به فرم زیر مي باشد.

```
class-var = new classname();
```

اکنون مي فهميد که چرا پرانتزهای بعد از نام کلاس مورد نیاز مي باشد و چیزی که واقعاً اتفاق مي افتد به فرم mybox1=new Box() است. عملگر new سازنده را فراخواني میکند و اگر سازنده اي براي کلاس تعريف نشده باشد جاوا از سازنده پیش فرز براي این کار استفاده مي کند. سازنده پیش فرز به طور خود کار کلیه متغیرهای نمونه ار با عدد صفر مقدار دهی اولیه مي کند. این سازنده براي کلاسهای ساده کفایت مي کند و برای کلاسهای پیچیده باید سازنده تعريف کنید. این قابلیت وجود دارد که به سازنده پارامتر اضافه کنیم و در زمان ايجاد به طور پویا مقدار دهی اولیه کنیم. این عمل مفید بودن سازنده را افزایش مي دهد. در زیر مثال Box را با سازنده پارامتردار مشاهده مي کنید.

```
/* Here/ Box uses a parameterized constructor to
initialize the dimensions of a box.
*/
class Box {
double width;
double height;
double depth;
// This is the constructor for Box.
Box(double w/ double h/ double d ){
width = w;
height = h;
depth = d;
}
```

```
// compute and return volume
double volume (){
return width * height * depth;
}
}
class BoxDemo {
public static void main(String args[] ){
// declare/ allocate/ and initialize Box objects
Box mybox1 = new Box(10/ 20/ 15);
Box mybox2 = new Box(3/ 6/ 9);
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("volume is " + vol);
}
}
```

خروجي برنامه به شكل زير خواهد بود.

```
Volume is 3000
Volume is 162
```

وراثت در جاوا

به کمک وراثت مي توان يك کلاس عمومي ساخته و ویژگی هاي که در يك مجموعه از اگلام مشترك مي باشند را در داخل آن قرار داده و سپس از اين کلاس پايه براي ايجاد ساير کلاسها که تفاوت جزئي با ساير کلاسها دارند استفاده کنید. در وراثت کلاس به ارث برنده چیزهاي را که منحصر به فرد خودش باشد را به آن اضافه مي کند. در جاوا کلاسي که به ارث برده مي شود را ابر کلاس (superclass) مي نامند. کلاسي راکه عمل ارث بري را انجام داده را زير کلاس (subclass) مي نامند.

بنابراين زير کلاس روايت تخصصي تر و مشخص تر از کلاس بالاتر يعني ابر کلاس است. زير کلاس کلیه متغير ها و متد هاي ابر کلاس را به متد ها و متغيرهاي منحصر به فرد خود اضافه مي کند.

براي ارث بردن از يك کلاس، خيلي ساده کافي است که يك کلاس را با استفاده از واژه کليدي extend در کلاس ديگر قرار دهيد. براي فهم کامل اين مطلب يك مثال ساده را نشان مي دهيم. در اینجا يك ابر کلاس به نام A ويك زير کلاس به نام B ايجاد مي کنيم. دراستفاده از واژه extend دقت فرمايد.

```
// A simple example of inheritance.
// Create a superclass.
class A {
int i, j;
void showij (){
System.out.println("i and j : " + i + " " + j);
} }
// Create a subclass by extending class A.
class B extends A {
```

```

int k;
void showk () {
    System.out.println("k : " + k);
}
void sum () {
    System.out.println("j+j+k : " + (i+j+k));
} }
class SimpleInheritance {
    public static void main(String args[] ) {
        A superOb = new A();
        B subOb = new B();
        // The superclass may be used by itself.
        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb :");
        superOb.showij();
        System.out.println();
        /* The subclass has access to all public members of
        its superclass .*/
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        System.out.println("Contents of subOb :");
        subOb.showj();
        subOb.showk();
        System.out.println();
        System.out.println("Sum of i/ j and k in subOb:");
        subOb.sum();
    }
}

```

خروجي اين برنامه به شكل زیر می باشد.

```

Contents of superOb:
i and j :10 20
Contents of subOb:
i and j :7 8
k :9
Sum of i/ j and k in subOb:
i+j+k :24

```

همان طور که مشاهده می کنید زیر کلاس B در بر گیرنده کلیه اعضای کلاس A است. به همین دلیل است که subclass می تواند به i و j دسترسی داشته باشد و showij() را فرا خوانی نماید. همچنین در داخل متد sum() می توان به طور مستقیم به i و j راکه بخشی از B بودند ارجاع نمود. ابر کلاس بودن به معنی این نیست که نمی توان خود آن ابر کلاس را به تنهایی استفاده کرد و همانند یک کلاس عادی با آن بر خورد می شود. نکته دیگری که وجود دارد این که یک زیر کلاس می تواند ابر کلاس یک زیر کلاس باشد. شکل عمومی اعلان یک کلاس که از یک ابر کلاس ارث می برد به فرم زیر می باشد.

```
class subclass-name extends superclass-name {
// body of class
}
```

اگر چه يك زیر کلاس در برگیرنده کلیه اعضاء ابر کلاس خود مي باشد اما نمي تواند به اعضائي از کلاس بالا که به عنوان private اعلان شده اند، دسترسی داشته باشد. به عنوان مثال سلسله مراتب ساده کلاس زیر را در نظر بگیرید.

```
/* In a class hierarchy/ private members remain
private to their class.
```

```
This program contains an error and will not
compile.
```

```
*/
```

```
// Create a superclass.
```

```
class A {
```

```
int i; // public by default
```

```
private int j; // private to A
```

```
void setij(int x/ int y ){
```

```
    i = x;
```

```
    j = y;
```

```
}}
```

```
// A's j is not accessible here.
```

```
class B extends A {
```

```
int total;
```

```
void sum (){
```

```
    total = i + j; // ERROR/ j is not accessible here
```

```
}}
```

```
class Access {
```

```
public static void main(String args[] ){
```

```
    B subOb = new B();
```

```
    subOb.setij(10, 12);
```

```
    subOb.sum();
```

```
    System.out.println("Total is " + subOb.total);
```

```
}
```

```
}
```

این برنامه کاملتر نخواهد شد زیرا ارجاع به j در داخل متد sum() در B سبب خطا در دسترسی خواهد شد. از آنجای که j به عنوان private اعلان شده، فقط توسط سایر اعضای کلاس خود قابل دسترسی است و زیر کلاسهای آن هیچ گونه دسترسی به آن ندارند. همان طور که قبلاً اعلان کردیم عضو از کلاس که توسط private اعلان شده برای کلاس خود اختصاصی شده و برای کدهای خارج از خود قابل دسترسی نیست.

برای نشان دادن قدرت واقعی وراثت يك مثال را بررسی خواهیم کرد. در این جا کلاس Box به نحوی گسترش یافته تا يك عنصر چهارم تحت نام weight را در بر گیرد و کلاس جدید شامل width ,height ,depth ,weight و يك box خواهد بود.

```
// This program uses inheritance to extend Box.
```

```
class Box {
```

```
double width;
```

```

double height;
double depth;
// construct clone of an object
Box(Box ob ){ // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
Box(double w/ double h/ double d ){
width = w;
height = h;
depth = d;
}
// constructor used when all dimensions specified
Box (){
width =- 1; // use- 1 to indicate
height =- 1; // an uninitialized
depth =- 1; // box
}
// compute and return volume
double volume (){
return width * height * depth;
}
}
// Here/ Box is extended to include weight.
class BoxWeight extends Box {
double weight; // weight of box
// constructor for BoxWeight
BoxWeight(double w/ double h/ double d/ double m ){
width = w;
height = h;
depth = d;
weight = m;
} }
class DemoBoxWeight {
public static void main(String args[] ){
Boxweight mybox1 = new BoxWeight(10/ 20/ 15/ 34.3);
Boxweight mybox2 = new BoxWeight(2/ 3/ 4/ 0.076);
double vol;
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight);
System.out.println();
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);
System.out.println("Weight of mybox2 is " + mybox2.weight);
}
}

```

```
}
```

نتیجه خروجی به فرم زیر خواهد بود.

```
Volume of mybox1 is 3000
Weight of mybox1 is 34.3
Volume of mybox2 is 24
Weight of mybox2 is 0.076
Boxweight
```

کلاس Boxweight کلیه متدها و متغیرهای کلاس Box را به ارث برده و متغیر weight را به آنها اضافه می کند. برای زیر کلاس ضرورتی ندارد که اعضاء موجود در کلاس بالاتر یعنی ابر کلاس را مجدداً پیاده سازی کند. از مزیت‌های اصلی وراثت این است که کافی است فقط یک بار یک ابر کلاس را ایجاد کرده و در داخل آن خصلت‌های مشترک را قرار داده آنگاه از آن برای ایجاد هر تعداد زیر کلاس استفاده کرد. بطور مثال در زیر یک کلاس از کلاس Box به ارث برده شده و خصلت رنگ را به آن اضافه کرده.

```
// Here/ Box is extended to include color.
class ColorBox extends Box {
int color; // color of box
ColorBox(double w/ double h/ double d/ double c ){
width = w;
height = h;
depth = d;
color = c;
}
}
```

یک متغیر ابر کلاس می تواند به یک شی زیر کلاس ارجاع کند. یک متغیر ارجاع مربوط به یک کلاس بالا را می توان به ارجاعی، به هر یک از زیر کلاس‌های مشتق شده از آن کلاس بالا انتساب نمود. در بسیاری از شرایط این جنبه از وراثت کاملاً سودمند است و به عنوان نمونه مثال زیر را در نظر بگیرید.

```
class RefDemo {
public static void main(String args[] ){
Boxweight weightbox = new BoxWeight(3/ 5/ 7/ 8.37);
Box plainbox = new Box();
double vol;
vol = weightbox.volume();
System.out.println("Volume of weightbox is " + vol);
System.out.println("Weight of weightbox is " +
weightbox.weight);
System.out.println();
// assign BoxWeight reference to Box reference
plainbox = weightbox;
vol = plainbox.volume(); // OK/ volume ()defined in Box
System.out.println("Volume of plainbox is " + vol);
/* The following statement is invalid because plainbox
dose not define a weight member .*/
```

```
// System.out.println("Weight of plainbox is " + plainbox.weight
}
}
```

در اینجا weightbox يك ارجاع به شي Boxweight است و plainbox يك ارجاع به اشياء Box است و مي توان plainbox را به عنوان يك ارجاع به شي weightbox انتساب نمود. نکته مهم كه در اين جا قابل بيان است اين كه نوع متغير ارجاع و نه نوع شيئي كه به آن اشاره شده كه تعيين مي كند کدام اعضا قابل دسترس هستند. اين به اين معني است كه هنگامی كه يك ارجاع مربوط به يك شي زیر کلاس، به يك متغير ارجاع ابر کلاس انتساب مي شود، شما فقط به بخشهاي از شي دسترسي داريد كه توسط ابر کلاس تعريف شده باشد. براي همين دليل مي باشد كه plainbox نمي تواند به weight دسترسي داشته باشد چون ابر کلاس نمي تواند آگاهي به موارد اضافه شده به زیر کلاس داشته باشد. براي همين دليل آخرين خط كد در توضيح درج شده.

ايجاد سلسه مراتب چند سطحي (Multilevel)

مي توان سلسه مراتبي ساخت كه كه شامل چندين لايه وراثت باشد. كاملاً قابل توجه است كه كه از يك زیر کلاس هم مي توان زیر کلاسي ديگري را نيز بدست آورد. به عنوان مثال اگر سه کلاس A,B,C داشته باشيم نگاه C مي تواند يك زیر کلاس از B و خود B يك زیر کلاس از A باشد و C كليخصلتهاي موجود در A,B را در خود دارد.

در مثال بعدي زیر کلاس boxweight به عنوان يك ابر کلاس استفاده شده تا زیر کلاسيهاي تحت عنوان shipment را ايجاد كند. کلاس shipment كليخصلتهاي Boxweight و Box را به ارث برده و يك فيلد به نام cost به آن افه شده كه هزينه كشتيراني يك محموله را نگهداري ميكند.

```
// Extend BoxWeight to include shipping costs.
// Start with Box.
class Box {
private double width;
private double height;
private double depth;
// construct clone of an object
Box(Box ob ){ // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
Box(double w, double h, double d ){
width = w;
height = h;
depth = d;
}

// constructor used when no dimensions specified
Box () {
```

```

width =- 1; // use- 1 to indicate
height =- 1; // an uninitialized
depth =- 1; // box
}
// constructor used when cube is created
Box(double len ){
width = height = depth = len;
}
// compute and return volume
double volume (){
return width * height * depth;
}
}
// Add weight.
class BoxWeight extends Box {
double weight; // weight of box
// construct clone of an object
BoxWeight(BoxWeight ob ){ // pass object to constructor
super(ob);
weight = ob.weight;
}
// constructor used when all parameters are specified
BoxWeight(double w/ double h/ double d/ double m ){
super(w, h, d); // call superclass constructor
weight = m;
}
// default constructor
BoxWeight (){
super();
weight =- 1;
}
// constructor used when cube is created
BoxWeight(double len/ double m ){
super(len);
weight = m;
}
}
// Add shipping costs
class Shipment extends BoxWeight {
double cost;
// construct clone of an object
Shipment(Shipment ob ){ // pass object to constructor
super(ob);
cost = ob.cost;
}
// constructor used when all parameters are specified
BoxWeight(double w, double h, double d,
double m, double c ){

```

```

super(w, h, d); // call superclass constructor
cost = c;
}
// default constructor
Shipment (){
super();
cost = - 1;
}
// constructor used when cube is created
BoxWeight(double len, double m, double c ){
super(len, m);
cost = c;
}
}
class DemoShipment {
public static void main(String args[] ){
Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);
Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.28);
double vol;
vol = shipment1.volume();
System.out.println("Volume of shipment1 is " + vol);
System.out.println("Weight of shipment1 is " + shipment1.weight);
System.out.println("Shipping cost :$" + shipment1.cost);
System.out.println();
vol = shipment2.volume();
System.out.println("Volume of shipment2 is " + vol);
System.out.println("Weight of shipment2 is " + shipment2.weight);
System.out.println("Shipping cost :$" + shipment2.cost);
}
}

```

خروجي اين برنامه بصورت زیر مي باشد:

```

Volume of shipment1 is 3000
Weight of shipment1 is 10
Shipping cost :$3.41
Volume of shipment2 is 24
Weight of shipment2 is 0.76
Shipping cost :$1.28

```

وراثت امکان استفاده مجدد از کدهای قبلی را به خوبی به وجود می آورد یعنی دیگر نیاز به تکرار کد متدها در داخل زیر کلاسها نداریم. نکته مهم دیگر این که متد `super()` همواره به سازنده موجود در نزدیکترین ابر کلاس ارجاع می کند. متد `super()` در `shipment` ، سازنده `Boxweight` را فراخوانی می کند. `Super()` در `Boxweight` سازنده موجود در `Box` را فراخوانی می کند.

در یک سلسه مراتب کلاس، اگر یک سازنده کلاس بالا نیازمند پارامترها باشد، آگاه کلیه زیر کلاسها باید آن پارامترها را بالای خط (up the line) بگذارند. این امر چه یک زیر کلاس پارامترهای خودش را نیاز داشته باشد و چه نیاز نداشته باشد، صحت خواهد داشت.

توجه داشته باشید که در مثال گفته شده فقط برای راحتی شما هر سه کلاس در داخل يك فایل قرار گرفته و در جاوا هر کدام از کلاسها باید در فایلهاي خاص خودشان قرار گرفته و جداگانه کامپایل شوند.

وقتي که سلسه مراتب کلاس ایجاد مي شود سازنده گان کلاسها سلسله مراتب را تشکیل مي دهند. سازنده گان به ترتیب مشتق شدنشان از ابر کلاس به زیر کلاس فراخواني مي شوند. چون متد `super()` باید اولین دستوري باشد که در يك سازنده زیر کلاس اجرا مي شود، این ترتیب هم‌امطور حفظ مي شود، خواه متد `super()` استفاده شود یا نه. اگر `super()` استفاده نشود انگاه سازنده پیش فرض یا سازنده بدون پارامتر هر يك از زیر کلاسها اجرا خواهند شد. برنامه زیر نشان مي دهد که چه زمانی سازنده ها اجرا مي شود.

```
// Demonstrate when constructors are called.
// Create a super class.
class A {
A () {
System.out.println("Inside A's constructor.")
} }
// Create a subclass by extending class A.
class B extends A {
B () {
System.out.println("Inside B's constructor.")
} }
// Create another subclass by extending B.
class C extends B {
C () {
System.out.println("Inside C's constructor.")
} }
class CallingCons {
public static void main(String args[] ) {
C c = new C();
}
}
```

خروجي به فرم زیر مي باشد:

```
Inside A's constructor
Inside B's constructor
Inside C's constructor
```

همان طوري که ملاحظه کردید سازنده گان کلاسها به ترتیب مشتق شدنشان فراخواني مي شوند زیرا يك ابر کلاس نسبت به زیر کلاسهاي خود آگاهي ندارد، هرگونه مقدار دهی اولیه که برای اجرا شدن نیاز داشته باشد، جدا از هر گونه مقدار دهی اولیه انجام شده توسط زیر کلاس است.

متد `super()`

در جاوا امکان این وجود دارد يك ابر کلاس را ایجاد کرده و در آن جزئیات پیاده سازي خود را در خودش نگهداري کند. در این شرایط راهي براي زیر کلاس وجود ندارد تا مستقیماً به این متغیر ها مربوط به خودش دسترسی داشته باشد و یا آنها را مقدار دهی اولیه کند. از انجاي که کپسوله سازي يك خصلت اولیه در شي گراي است، این باعث تعجب نخواهد بود که جاوا راه حل را با کمک متد `super()` حل کرده. هر زمان که لازم باشد تا يك زیر کلاس به ابر کلاس قبلي خود ارجاع کند با استفاده از `super()` این کار را انجام می دهد.

متد `super()` دو شکل عمومي دارد، اولین شکل سازنده ابر کلاس را فراخواني میکند. دومین شکل به منظور دسترسی به يك عضو ابر کلاس که توسط يك عضو زیر کلاس مخفی مانده استفاده می شود. يك زیر کلاس می تواند متد سازنده تعریف شده توسط ابر کلاس مربوطه را با استفاده از فرم `super` فراخواني می کند:

```
super( parameter-list);
```

در این متد `parameter-list` مشخص کننده هر پارامتری است که توسط سازنده در ابر کلاس مورد نیاز می باشد. این متد باید همواره اولین دستور اجرا شده داخل يك سازنده زیر کلاس باشد. به نحوه استفاده از این متد در مثال زیر دقت کنید.

```
// BoxWeight now uses super to initialize its Box attributes.
```

```
class BoxWeight extends Box {
double weight; // weight of box
// initialize width/ height/ and depth using super()
BoxWeight(double w/ double h/ double d/ double m ){
super(w, h, d); // call superclass constructor
weight = m;
}
}
```

متد `Boxweight()` فراخواني `super()` را با پارامتر هاي `w` و `d,h` انجام میدهد. این کار سبب میشود تا سازنده `Box()` فراخواني شده و با استفاده از مقادیر `height`, `depth` و `width` مقداری دهی اولیه کند و دیگر `Boxweight` خود این مقدار دهی اولیه را انجام نمی دهد. فقط لازم است تا مقدار دهی منحصر به فرد خود `weight` را مقدار دهی دهی اولیه نماید. انجام این عمل `Box` را آزاد می گذارد تا در صورت تمایل این مقادیر را `private` بسازد.

در مثال قبلي متد `super()` با سه پارامتر فراخواني شد. اما چون سازندگان ممکن است انباشته شوند، می توان متد `super()` را با استفاده از هر شکل تعریف شده توسط ابر کلاس فراخواني کرد. سازنده ای که اجرا می شود همان سازنده ای است که با پارامترها مطابقت دارد.

به طور مثال در این جا يك پیاده سازي کامل از کلاس `Boxweight` موجود است که سازنده گان را برای روش هاي گوناگون ممکن ساخته شدن يك `box` فراهم میکند. در هر حالت متد `super()` با استفاده از پارامترهاي تقزیبي فراخواني می شود. دقت کنید که `height`, `depth` و `width` در داخل `Box` به صورت اختصاصي در آمده اند.

```
// A complete implementation of BoxWeight.
```

```
class Box {
private double width;
private double heght;
private double deoth;
// construct clone of an object
Box(Box ob ){ // pass object to constructor
```

```

width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
Box(double w/ double h/ double d ){
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
Box (){
width =- 1; // use- 1 to indicate
height =- 1; // an uninitialized
depth =- 1; // box
}
// constructor used when cube is created
Box(double len ){
width = height = depth = len;
}
// compute and return volume
double volume (){
return width * height * depth;
}
}
// BoxWeight now fully implements all constructors.
class BoxWeight extends Box {
double weight; // weight of box
// construct clone of an object
BoxWeight(BoxWeight ob ){ // pass object to constructor
super(ob);
weight = ob.weight;
}
// constructor used when all parameters are specified
Box(double w, double h, double d, double m ){
super(w, h, d); // call superclass constructor
weight = m;
}
// default constructor
BoxWeight (){
super();
weight =- 1;
}
// constructor used when cube is created
BoxWeight(double len, double m ){
super(len);
weight = m;
}
}

```

```

}
class DemoSuper {
public static void main(String args[] ){
BoxWeight mybox1 = new BoxWeight(10 ,20 ,15 ,34.3);
BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
BoxWeight mybox3 = new BoxWeight(); // default
BoxWeight mycube = new BoxWeight(3, 2);
BoxWeight myclone = new BoxWeight(mybox1);
double vol;
vol = mybox1.vilume();
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight);
System.out.println();
vol = mybox2.vilume();
System.out.println("Volume of mybox2 is " + vol);
System.out.println("Weight of mybox2 is " + mybox2.weight);
System.out.println();
vol = mybox3.vilume();
System.out.println("Volume of mybox3 is " + vol);
System.out.println("Weight of mybox3 is " + mybox3.weight);
System.out.println();
vol = myclone.vilume();
System.out.println("Volume of myclone is " + vol);
System.out.println("Weight of myclone is " + myclone.weight);
System.out.println();
vol = mycube.vilume();
System.out.println("Volume of mycube is " + vol);
System.out.println("Weight of mycube is " + mycube.weight);
System.out.println();
}
}

```

خروجي این برنامه به شکل زیر خواهد بود:

```

Volume of mybox1 is 3000
Weight of mybox1 is 34.3
Volume of mybox2 is 24
Weight of mybox2 is 0.076
Volume of mybox3 is- 1
Weight of mybox3 is- 1
Volume of myclone is 3000
Weight of myclone is 34.3
Volume of mycube is 27
Weight of mycube is 2

```

سازنده در کلاس Boxweight که به شکل زیر است:

```

// construct clone of an object
BoxWeight(BoxWeight ob ){ // pass object to constructor

```

```

super(ob);
weight = ob.weight;
}

```

به نکته توجه کنید که متد `super()` با یک شی از نوع `Boxweight` و نه از نوع `Box` فراخوانی شده است سازنده کلاس `Box` یعنی `Box(Box ob)` را فراخوانی می کند. همان طوری که می دانید یک متغیر ابر کلاس را می توان برای ارجاع به هر شی مشتق شده از آن کلاس مورد استفاده قرار داد. برای همین دلیل می باشد که ما قادر هستیم یک شی `Boxweight` را به سازنده `Box` پاس کنیم و البته `Box` نسبت به اعضای خود آگاهی دارد. زمانی که یک زیر کلاس متد `super()` را فراخوانی می کند، در اصل این متد به ابر کلاس کلاسی که از آن فراخوانی شده ارجاع می کند. این امر حتی در یک سلسله مراتب چند سطحی هم نیز صادق است و دقت داشته باشید که متد `super()` باید اولین دستوری باشد که در داخل یک سازنده زیر کلاس اجرا می شود.

دومین کاربرد `super` همانند `this` عمل می کند با این تفاوت که `super` به ابر کلاس زیر کلاسی که در آن استفاده می شود اشاره دارد و فرم عمومی استفاده از به صورت زیر می باشد.

`Super.member`

در اینجا `member` می تواند یک متد یا یک متد در ابر کلاس باشد. این روش زمانی کاربرد پیدا می کند که در آن اسامی اعضای یک زیر کلاس، اعضای با هما اسمی را در ابر کلاس مخفی می سازد. این سلسله مراتب ساده کلاس را در نظر بگیرید:

```

// Using super to overcome name hiding.
class A {
int i;
}
// Create a subclass by extending class A.
class B extends A {
int i; // this i hides the in A
B(int a,int b ){
super.i = a; // i in A
i = b; // i in B
}
void show (){
System.out.println("i in superclass : " + super.i);
System.out.println("i in subclass : " + i);
} }
class UseSuper {
public static void main(String args[] ){
B subOb = new B(1,2);
subOb.show();
}
}

```

بعد از اجرای این برنامه خروجی زیر نمایش داده خواهد شد.

```

i in superclass :1
i in subclass :2

```

اگر چه متغیر نمونه i در B متغیر نمونه I در A را پنهان می کند اما super امکان دسترسی به I تعریف شده در کلاس بالا به وجود می آورد. همچنین میتوان از super می توان برای فراخوانی متدهای که توسط زیر کلاس مخفی شده اند استفاده کرد.

متد finalize()

در بعضی از مواقع لازم است که تا یک شیء هنگامی که در حال خراب شد است یک عمل خاص را انجام دهد. خراب شدن شیء زمانی است که در اتمام استفاده از شیء قصد برداشتن آن از حافظه و حذف اشاره گر به آن را داریم. به عنوان مثال ممکن است که یک شیء در برگزیده منابع غیر جاوا نظر یک دستگیر فایل (file handle) یا فونت کاراکتر ویندوز باشد، و میخواهید اطمینان پیدا کنید که قبل از اینکه آن شیء خراب شود منابع فوق رها شوند.

برای اداره چنین شرایطی جاوا مکانیزمی تحت نام تمام کننده (finalization) را فراهم آورده. به کمک این مکانیزم می توان عملیات مشخصی تعریف نماید که در زمانی که اشغال جمع کن (garbage collector) جاوا قصد گرفتن حافظه آن شیء و افزودن آن به به فضای آزاد سیستم را دارد آن عملیات اتفاق بیافتد. برای افزودن یک تمام کننده به کلاس کافی است که متد finalize() را به کلاس خود اضافه کنید. شکل کلی استفاده از متد finalize() به فرم زیر است.

```
protected void finalize()
{
// finalization code here
}
```

توصیفگر protected برای این می باشد که از دسترسی به متد finalize() توسط کدهای تعریف شده خارج از کلاس جلوگیری کند. توجه کنید که متد finalize() قبل از شروع کار اشغال جمع کن جاوا بر روی آن شیء اجرا می شود. اگر یک شیء اشاره گر به آن گم شود، بر اثر اشتباه در برنامه شما و آن شیء از قلمرو خارج شود نمی توان تشخیص داد که متد finalize() چه زمانی اجرا خواهد شد. بنابراین برنامه شما باید سایر وسایل برای آزاد کردن منابع سیستم مورد استفاده شیء را تدارک ببیند. برنامه شما نباید فقط متکی به finalize() باشد. البته جاوا این پیش بینی را کرده و یک یک سیستم جمع کردن این اشیاء در خود ایجاد کرده.

استفاده از this

در زمانهای لازم می باشد که یک روش به شیئی که آنرا فراخوانی می کند ارجاع کند. برای این کار جاوا واژه کلیدی this را تعریف کرده است. به کمک this می توان در داخل هر متد به یک ارجاع به شیء جاری داشت. این به این معنی می باشد که this همواره ارجاعی به شیئی است که متد روی آن فراخوانی شده است. میتوان از this هر جایی که ارجاع به یک شیئی از نوع کلاس جاری مجاز می باشد استفاده کرد. برای فهم بهتر در مورد کار this روایت بعدی کلاس Box را در نظر بگیرید.

```
// A redundant use of this.
Box(double w/ double h/ double d ){
this.width = w;
this.height = h;
this.depth = d;
}
```

این روایت از سازنده دقیقاً مثل روایت اولیه کار می کند. استفاده از this به ندرت انجام می گیرد. در داخل Box() همواره this به شئی فراخوانده شده ارجاع می کند.

مخفی نمودن متغیر نمونه

حتماً می دانید که اعلان دو متغیر محلی با يك نام داخل يك قلمرو یا قلمرو های بسته شده، غیر مجاز است. به طور جالبی می توان متغیرهای محلی شامل پارامترهای رسمی برای متدها، داشته باشید که با اسامی متغیرها نمونه کلاسها مشترک باشند. اما هنگامی که يك متغیر محلی همان اسم متغیر نمونه را داشته باشد، متغیر محلی متغیر نمونه را مخفی می سازد. برای همین دلیل بود که از height, depth و weight به عنوان اسامی پارامترهای سازنده Box() داخل کلاس Box استفاده نکردیم. اگر از آنها به همین روش استفاده می کردیم، آنگاه width به پارامترهای رسمی ارجاع می کرد و متغیر نمونه width را مخفی نگه می داشت. اگر چه آسان تر می باشد که از اسامی متفاوت استفاده کنیم، اما راه حل دیگری برای چنین شرایطی وجود دارد. چون this امکان ارجاع مستقیم به شیئی را به شما می دهد، می توانید با استفاده از آن هر نوع تصادف و یکی شدن اسامی محلی را رفع کنید. به عنوان مثال، در این جا روایت دیگری از Box() وجود دارد که از height, weight به عنوان اسامی پارامترها استفاده نموده و آنگاه از this برای دستیابی به متغیرهای نمونه با همین اسامی استفاده کرده است:

```
// Use this to resolve name-space collisions.
Box(double width/ double height/ double depth ){
this.width = width;
this.height = height;
this.depth = depth;
}
```

دقت داشته باشید که استفاده از this در چنین متنی ممکن است گاهی گیج کننده شود و برخی برنامه نویسان مراقب هستند تا از اسامی متغیرهای محلی و پارامترهای رسمی که متغیرهای نمونه را مخفی می سازند استفاده نکنند. البته سایر برنامه نویسان معتقدند استفاده اسامی مشترک برای وضوح مفید بوده و از this برای غلبه بر مخفی سازی متغیرهای نمونه کمک می گیرند. انتخاب یکی از دو روش با سلیقه شما ارتباط دارد.

استفاده از کلاسهای انتزاعی (abstract)

در جاوا این امکان وجود دارد که يك ابر کلاس تعریف نموده که ساختار يك انتزاع معین را بدون يك پیاده سازی کامل از هر روشی اعلان نماید. این به این معنی است که می خواهید يك ابر کلاس ایجاد کنید که فقط يك شکل عمومی شده تعریف کند که توسط کلیه زیر کلاسها به اشتراک گذاشته خواهد شد و پر کردن جزئیات این شکل عمومی

به عهده هر يك از زیر کلاسها واگذار مي شود. و شيوه براي وقوع اين شرايط زماني است كه يك ابر کالس تواناي ايجاد يك پياده سازي با معني براي يك متد را نداشته باشد.

تعريف متد area() خيلي ساده يك نگهدارنده مكان است. اين متد روش مساحت انواع شي را محاسبه نكرده و نمايش نمي دهد. هنگام ايجاد كتابخانه هاي خاص کلاس خود اين غير معمول نيست كه اگر يك متد هيچ تعريف با معني در متن ابر کلاس خود نداشته باشد. جاوا براي حل از متد انتزاعي (abstract method) استفاده مي كند. مي توانيد توسط زیر کلاسها و با مشخص كردن اصلاح كننده نوع abstract متدهاي معني را لغو كنيد. گاهي به اين متدها جوابگويي زیر کلاس (subclass responsibility) مي گويند زيرا آنها هيچ پياده سازي مشخص شده اي در ابر کلاس ندارند. بنا بر اين يك زیر کلاس بايد آنها را لغو نمايد چون نمي تواند به سادگي روايت تعريف شده در ابر کلاس را استفاده نمايد.

براي اعلام يك متد انتزاعي از شكل عمومي زیر استفاده مي كنيم.

abstract type name(parameter-list);

همان طور كه ملاحظه مي كنيد در اين جا بدنه متد اعلان نشده . هر کلاسي كه يك يا چند متد انتزاعي دارد بايد به عنوان انتزاعي اعلام گردد. براي اعلان يك کلاس به عنوان انتزاعي از واژه كليدي abstract در جلوي واژه كليدي class در ابتداي تعريف کلاس مي كنيم. براي يك کلاس انتزاعي هيچ شئي نم توتن ايجاد نمود. يعني يك کلاس انتزاعي نبايد بطور مستقيم با با عملگر new نمونه سازي شود. اين به خاطر اين مي باشد كه يك کلاس انتزاعي بطور كامل تعريف نشده است. همچنين نمي توانيد سازنده گان انتزاعي يا متدهاي ايستاي انتزاعي اعلان كنيد.

هر زیر کلاس از يك کلاس انتزاعي بايد يا كلييه متدهاي انتزاعي موجود در ابر کلاس را پياده سازي كند و يا خودش به عنوان يك abstract اعلان شود. در اينجا مثال ساده اي از يك کلاس با يك روش انتزاعي مشاهده مي كنيد كه بعد از آن يك زیر کلاس قرار گرفته كه آن متد را پياده سازي مي كند.

```
// A Simple demonstration of abstract.
abstract class A {
    abstract void callme();
    // concrete methods are still allowed in abstract classes
    void callmetoo () {
        System.out.println("This is a concrete method.");
    }
}
class B extends A {
    void callme () {
        System.out.println("B's implementetion of callme.");
    }
}
class AbstractDemo {
    public static void main(String args[] ) {
        B b = new B();
        b.callme();
        b.callmetoo();
    }
}
```

توجه داشته باشيد كه هيچ شئي از A در برنامه اعلان نشده است. همان طوري كه قبلاً گفته شد امكان نمونه سازي يك کلاس انتزاعي وجود ندارد. نکته اي كه بايد توجه داشته باشيد اين كه کلاس A يك متد واقعي با نام callmtoo() راپياده سازي مي كند.

این امر کاملاً قابل قبول است. کلاسهای انتزاعی می توانند تا زمانی که تناسب را حفظ نمایند، در بر گیرنده پیاده سازی ها باشند. اگر چه نمی توان از کلاسهای انتزاعی برای نمونه سازی اشیاء استفاده کرد اما می توان از آنها برای ایجاد ارجاعات برای اشیاء استفاده کرد زیرا روش جاوا از طریق استفاده از ارجاعات ابر کلاس پیاده سازی خواهد شد. بنابراین باید امکان ایجاد یک ارجاع به یک کلاس انتزاعی وجود داشته باشد بطوری که با استفاده از آن ارجاع به یک شیئی زیر کلاس اشاره نمود. شما در کلاس بعدی این موضوع را مشاهده خواهید کرد. با استفاده از از یک کلاس انتزاعی میتوانید کلاس Figure() را توسعه دهید. چون مفهوم با معنای برای مساحت یک شکل دو بعدی تعریف نشده وجود ندارد، روایت بعدی این برنامه area() را به عنوان یک انتزاع داخل کلاس Figure اعلان می کند. البته این بدان معنی است که کلیه کلاسهای مشتق شده از Figure باید متد area() را لغو کنند.

```
// Using abstract methods and classes.
abstract class Figure {
    double dim1;
    double dim2;
    Figure(double a/ double b ){
        dim1 = a;
        dim2 = b;
    }
    // area is now an abstract method
    abstract double area();
}
class Rectangle extends Figure {
    Rectangle(double a,double b ){
        super(a,b);
    }
    // override area for rectangle
    double area (){
        System.out.println("Inside Area for Rectangle.");
        return dim1 * dim2;
    }
}
class Triangle extends Figure {
    Triangle(double a,double b ){
        super(a,B);
    }
    // override area for right triangle
    double area (){
        System.out.println("Inside Area for Teriangle.");
        return dim1 * dim2 / 2;
    }
}
class AbstractAreas {
    public static void main(String args[] ){
        // Figure f = new Figure(10/ 10); // illegal now
```

```

Rectangle r = new Rectangle(9/ 5);
Triangle t = new Triangle(10/ 8);
Figure figref; // this is OK/ no object is created
figref = r;
System.out.println("Area is " + figref.area());
figref = t;
System.out.println("Area is " + figref.area());
}
}

```

همان طوري كه توضيح درون متد main() نشان مي دهد، ديگر امكان اعلان اشياء از نوع Figure وجود ندارد چون اكنون بصورت انتزاعي است كه كليۀ زير كلاسهاي Figure بايد متد area() را لغو كنند. براي اثبات اين امر سعي كنيد كه يك زير كلاس ايجاد نمايد كه متد area() را لغو نمي كند. حتماً يك خطاي زمان كامپايل دريافت خواهيد كرد . اگر چه امكان ايجاد يك شي از نوع Figure وجود ندارد اما مي توانيد يم كتغير ارجاع از نوع Figure ايجاد نمايد. متغير figref به عنوان ارجاعي به Figure اعلان شده و بدان معني مي باشد كه با استفاده از آن مي توان به يك شي از هر كلاس مشتق شده از Figure ارجاع كرد.

مفهوم static

شرائطي وجود دارد كه مايليد يك عضو كلاس را طوري تعريف كنيد كه مسبقاً از هر شيئي آن كلاس مورد استفاده قرار گيرد. بطور معمول يك عضو كلاس بايد فقط همراه يك شيئي از همان كلاس مورد دسترسي قرار گيرد. اما اين امكان وجود دارد كه عضوي را ايجاد كنيم كه توسط خودش استفاده شود، بدون اينكه به يك نمونه مشخص ارجاع كنيم. براي ايجاد چنين عضوي ، قبل از آن واژه كليدي static را قرار دهيد. زماني يك عضو به عنوان static اعلام شود مي توان قبل از ايجاد هر شيئي از آن كلاس ، و بدون ارجاعي به هيچ يك از اشياء آن را مورد استفاده قرار دهيد. مي توان متدها را هم به عنوان static اعلان نمايد.

رايچ ترين مثال براي يك عضو static همان متد main() است. به اين دليل اين كار انجام مي شود كه چون بايد قبل از وجود هر نوع شيئي فراخواني شود . متغيرهاي نمونه اعلان شده به عنوان static ضرورتاً متغيرهاي سراسري هستند . هنگامي كه اشياء آن كلاس اعلان مي شوند هيچ كپي از متغير static ساخته نمي شود. به جاي آن كليۀ نمونه هاي آن كلاس همان متغير هاي static را به اشتراك مي گذارند. متدهاي اعلان شده به عنوان static داراي چندين محدوديت هستند. آنها فقط ساير متدهاي static را فراخواني مي كند و فقط به دادهاي static دسترسي دارند. آنها به هيچ وجه امكان ارجاع به super, this را ندارند. اگر لازم است محاسباتي انجام دهيد تا متغيرهاي static را مقدار دهني اوليه نمايد، مي توانيد يك بلوك static اعلان نمايد كه فقط يك بار آن هم زماني كه كلاس براي اولين بار بارگذاري مي شود، اجرا گردد. مثال بعدي كلاسي را نشان مي دهد كه يك متد static برخي متدهاي static و يك بلوك مقداري دهني اوليه static دارد :

```

// Demonstrate static variables/ methods/ and blocks.
class UseStatic {
static int a = 3;
static int b;
static void meth(int x ){
System.out.println("x = " + x);
}
}

```

```

System.out.println("a = " + a);
System.out.println("b = " + b);
}
static {
System.out.println("Static block initialized.");
b = a * 4;
}
public static void main(String args[] ){
meth(42);
}
}

```

به محض این که کلاس usestatic بارگذاری می شود ، کلیه دستورات static اجرا می شوند. ابتدا مقدار a برابر با 3 می شود، سپس بلوک static اجرا شده (یک پیام چاپ خواهد شد) و در نهایت مقدار $a*4$ یا عدد 12 در b به عنوان مقدار اولیه نهاده می شود. سپس متد main() فراخوانی می شود که متد math() را فراخوانی خواهد کرد و مقدار 42 را به x خواهد گذاشت. سه دستور println() به دو متغیر static یعنی a, b و همچنین به متغیر محلی x ارجاع می کنند. همان طور که قبلاً گفتیم ارجاع به هر یک از متغیر های نمونه داخل یک متد static غیر مجاز می باشد. خروجی برنامه به شکل زیر خواهد بود:

Static block initialized.

x = 42

a = 3

b = 12

متدها و متغیر های اعلان شده با static را می توان در خارج از کلاسی که تعریف شده اند مستقل از هر نوع شیئی مورد استفاده قرار داد. برای انجام این کار کافی است که نام کلاس را با یک عملگر نقطه بعد از مشخص نمایید. بطور مثال می توانید اگر بخواهید یک متد static را از خارج از کلاس مربوطه فراخوانی کنید، با استفاده از شکل عمومی زیر این کار را انجام دهید:

classname.method()

در اینجا classname نام کلاسی است که متد static در آن اعلان شده است. همان طوری که می توانید ببینید، این شکل بندی مشابه همان است که برای فراخوانی متدهای غیر static از طریق متغیرهای ارجاع شیئی انجام می گرفت. یک متغیر static را نیز می توان با همان روش با استفاده از عملگر نقطه ای روی نام کلاس مورد دسترسی قرار داد. این روشی است که که جاوا به وسیله آن یک روایت کنترل شده از توابع سراسری و متغیرهای سراسری را پیاده سازی می کند. در این جا مثالی وجود دارد، در داخل متد main() متد classname() و متغیر b که static هستند در خارج از کلاسهای خود مورد دسترسی قرار می گیرند.

```

class StaticDemo {
static int a = 42;
static int b = 99;
static void callme () {
System.out.println("a = " + a);
}
}

```

```

class StaticByName {
public static void main(String args[] ){
StaticDemo.callme();
System.out.println("b + " + StaticDemo.b);
}
}

```

استفاده از اشیاء به عنوان پارامتر

تا کنون فقط از انواع ساده به عنوان پارامترهای متدها استفاده کردیم. اما هم صحیح و هم معمول است که اشیاء را نیز به متدها پاس کنیم. مثال زیر را در نظر بگیرید:

```

// Objects may be passed to methods.
class Test {
int a/ b;
Test(int , int j ){
a= i;
b = j;
}
// return true if o is equal to the invoking object
boolean equals(Test o ){
if(o.a == a && o.b == b )return true;
else return false;
}
}
class PassOb {
public static void main(String args[] ){
Test ob1 = new Test(100, 22);
Test ob2 = new Test(100, 22);
Test ob3 = new Test(-1,- 1);
System.out.println("ob1 == ob2 :" + ob1.equals(ob2));
System.out.println("ob1 == ob3 :" + ob1.equals(ob3));
}
}

```

برنامه این خروجی را تولید خواهد کرد.

```

ob1 == ob2 :true
ob1 == ob3 :false

```

در مثال متد equals() در داخل Test دو شیء را از نظر کیفیت مقایسه نموده و نتیجه را برمی گرداند. یعنی این متد شیء فراخواننده را با شیء که پاس شده مقایسه کرده و اگر محتوای آن دو یک باشد آنگاه true را بر می گرداند. در غیر این صورت false را برمی گرداند. پارامتر o در متد equals() مشخص کننده Test به عنوان نوع آن می باشد. اگرچه Test یک نوع کلاس ایجادشده توسط برنامه است، اما به عنوان انواع توکار جاوا و به همان روش مورد استفاده قرار گرفته است. یک از رایجترین کاربردهای پارامترهای شیئی مربوط به سازنده گان است. غالباً ممکن است که بخواهید یک شیء جدید را بسازید طوری که این شیء در ابتدا نظیر یک شیء موجود باشد. برای انجام این کار باید یک تابع سازنده تعریف نمایید که شیء از کلاس خود را به عنوان یک پارامتر انتخاب می کند.

بطور مثال روایت بعدی از Box() به يك شیئی امکان داده تا آغاز گر دیگری باشد.

```
// Here/ Box allows one object to initialize another.
class Box {
double width;
double height;
double depth;
// construct clone of an object
Box(Box ob ){ // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
Box(double w, double h, double d ){
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
Box (){
width =- 1; // use- 1 to indicate
height =- 1; // an uninitialized
depth =- 1; // box
}
// constructor used when cube is created
Box(double len ){
width = height = depth
}
// compute and return volume
double volume (){
return width * height * depth;
} }
class OverloadCons2 {
public static void main(String args[] ){
// create boxes using the various constructors
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box();
Box mycube = new Box(7);
Box myclone = new Box(mybox1);
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);
// get volume of cube
```

```

vol = mycube.volume();
System.out.println("Volume of cube is " + vol);
// get volume of clone
vol = myclone.volume();
System.out.println("Volume of clone is " + vol);
}
}

```

بعدها خواهید دید که وقتی شروع به ایجاد کلاسهای خود می‌نمایید، معمولاً برای اینکه اجازه دهیم تا اشیاء بصورتی موثر و آسان ساخته شوند، لازم است اشکال‌های سازنده را فراهم آوریم.

استفاده از instanceof

در زبانهای برنامه نویسی امروزی این امکان وجود دارد که بتوان بعد کامپایل و در زمان اجرا نوبی شیء را تشخیص داد. به طور مثال یک نخ (thread) که داخل آن اشیاء مختلف تولید کرده و یک نخ که اینها را پردازش می‌کند، در شرایطی برای پردازنده نخ مفید می‌باشد که نوع هر یک از اشیائی را که قصد پردازش دارد بداند. دلیل دیگری که دانستن نوع شیء را در زمان اجرا مهم می‌کند تبدیل نوع یا casting می‌باشد. در جاوا یک تبدیل غیرمجاز سبب بروز خطای زمان اجرا می‌شود و اکثر تبدیلات غیر مجاز در زمان کامپایل قابل رفع هستند. اما casting شامل سلسله مراتب کلاس می‌باشد که می‌تواند تبدیلات غیر مجاز تولید کند و فقط در زمان اجرا قابل کشف می‌باشند. برای مثال یک ابر کلاس A می‌تواند به زیر کلاس B تبدیل کنیم و یک شیء از نوع C را به نوع A تبدیل کنیم. اما مجاز نیستیم یک شیء B را به نوع C یا بالعکس تبدیل کنیم. از آنجایی که یک شیء از نوع A می‌تواند به اشیائی از نوع B یا C ارجاع کند در زمان اجرا برای فهمیدن شیء ارجاع شده می‌توان قبل از تبدیل تستی را انجام داد. به کمک عملگر زمان اجرای جاوا instanceof می‌توان این کنترل را انجام داد. فرم عمومی instanceof به شکل زیر می‌باشد.

object instanceof type

یک نمونه از کلاس به کمک object و نوع کلاس با type مشخص می‌شود. اگر object از نوع مشخص شده باشد آنگاه عملگر instanceof مقدار true بر می‌گرداند. در غیر این صورت false بر می‌گرداند. به این ترتیب instanceof کمکی است برای اینکه برنامه شما اطلاعات نوع در باره یک شیء را در زمان اجرا بدست آورد. برنامه زیر نشان دهنده استفاده از instanceof است.

// Demonstrate instanceof operator.

```

class A {
int i/ j;
}
class B {
int i/ j;
}
class C extends A {
int k;
}
class D extends A {
int k;
}

```

```

class InstanceOf {
public static void main(String args[] ){
A a = new A();
B b = new B();
C c = new C();
D d = new D();
if(a instanceof A)
System.out.println("a is instance of A");
if(b instanceof B)
System.out.println("b is instance of B");
if(c instanceof C)
System.out.println("c is instance of C");
if(c instanceof A)
System.out.println("c is instance of A");
if(a instanceof C)
System.out.println("a is instance of C");
System.out.println();
// compare types of derived types
A ob;
ob = d; // A reference to d
System.out.println("ob new refers to d");
if(ob instanceof D)
System.out.println("ob is instance of D");
System.out.println();
ob = c; // A reference to c
System.out.println("ob new refers to c");
if(ob instanceof D)
System.out.println("ob is instance of D");
else
System.out.println("ob is instance of D");
if(ob instanceof A)
System.out.println("ob is instance of A");
System.out.println();
// all object can be cast to Object
if(a instanceof object)
System.out.println("a may be cast to Object");
if(b instanceof object)
System.out.println("b may be cast to Object");
if(c instanceof object)
System.out.println("c may be cast to Object");
if(d instanceof object)
System.out.println("d may be cast to Object");
}
}

```

خروجي حاصل از اجراي اين برنامه به شكل زير مي باشد:

a is instance of A
b is instance of B

c is instance of C
c can be cast to A

ob now refers to d
ob is instance of D

ob now refers to c
ob cannot be cast to D
ob can be cast to A

a may be cast to Object
b may be cast to Object
c may be cast to Object
d may be cast to Object

کلاس Singleton

در برخی مواقع با کلاسهای برخورد می کنیم که لزوماً باید یک و فقط یک متغیر از آنها تعریف شود. بطور مثال یک عامل یا شی که به یک منبع قابل اشتراک دسترسی دارد اما هیچ چیزی نمی تواند شی را از تعریف متغیر دیگری از آن باز دارد پس چه کاری می توان انجام داد. الگویی تک برگ برای انجام این کار است . الگویی تک برگ با گرفتن وظیفه ایجاد و قطع دسترسی به متغیر در خود شی طرح را محدود می کند. چنین کاری تضمین می کند که تنها یک متغیر ایجاد شود و دسترسی به آن منفرد باشد. در زیر پیاده سازی از الگویی تک برگ را مشاهده می کنید.

```
/*
a class refrence to the singleton instance
*/
public class Singleton {
private static Singleton instance;
protected Singleton(){}
public static Singleton getInstance(){
if (instance== null) {
instance= new Singleton();
}
return instance;
}
}
```

کلاس Singleton یک متغیر static از نوع Singleton دارد که دسترسی به آن را فقط به روال getInstance() محدود کرده.

متدها در جاوا

زیر برنامه ها در جاوا

زیر برنامه عضو يك کلاس را با کلمه متد در جاوا شناساي مي کنند و فرم کلي تعريف يك متد به شکل زیر مي باشد.

```
modifiers return-type subroutine-name ( parameter-list ) {
statements
}
```

در قسمت statement دستوراتي را که در داخل متد مورد نظر شما انجام مي شود مي توانيد با توجه به پارامتر هاي ورودی دستورات را وارد کنید.

در قسمت modifiers که در شروع تعريف يك متد مي نويسيم مشخصات حقيقي يك متد را تعيين ميکند. همان طور که گفته شد کلمه public به اين معني است که اين متد خارج از کلاس توسط شي تعريف شده از نوع کلاس مي تواند مورد دسترسي قرار بگيرد.

متد يك تابع مي باشد که کار خاصي را انجام مي دهد مثل کارهاي محاسباتي پس بايد يك خروجي داشته باشد. نوع خروجي تابع را با return-type که يکي از انواع داده هاي جاوا مي باشد مشخص مي کنيم. اگر متد شما خروجي نداشته باشد بايد مقدار void را در انجا بنويسيد.

در قسمت parameter-list از متد پارامتر هاي ورودی به متد مورد نظر را مشخص مي کنيم. در نوشتن پارامتر ورودی ابتدا نوع پارامتر ورودی را به فرم type parameter-type تعيين کرده و اگر چند پارامتر به عنوان ورودی داشته باشيم آنها را با علامت کاما از هم جدا مي کنيم.

بطور مثال "double x,double y" و دقت کنید که نوشتن به فرم "double x,y" اشتباه مي باشد.

در زیر چند مثال از متدها را مشاهده مي کنید.

```
public static void playGame() {
// "public" and "static" are modifiers; "void" is the
// return-type; "playGame" is the subroutine-name;
// the parameter-list is empty
... // statements that define what playGame does go here
}
```

```
int getNextN(int N) {
// there are no modifiers; "int" in the return-type
// "getNextN" is the subroutine-name; the parameter-list
// includes one parameter whose name is "N" and whose
// type is "int"
... // statements that define what getNextN does go here
}
```

```
static boolean lessThan(double x, double y) {
// "static" is a modifier; "boolean" is the
// return-type; "lessThan" is the subroutine-name; the
// parameter-list includes two parameters whose names are
// "x" and "y", and the type of each of these parameters
// is "double"
... // statements that define what lessThan does go here
}
```

افزودن يك متد به كلاس Box

در اغلب اوقات از متدها براي دسترسي به مقدار متغيرهاي نمونه استفاده مي كنيم. همچنين مي توانيد متدهاي تعريف كنيد كه به صورت داخلي و توسط خود كلاس مورد استفاده قرار گيرند. در مثال Box محاسبه فضاي اشغالي (volume) چيزي است كه توسط كلاس Box در مقايسه با كلاس BoxDemo راحت تر مديريت مي شود. همچنين چون فضاي اشغالي يك box به اندازه آن دارد به نظر مي رسد كه بهتر است كه كلاس Box اين محاسبه را انجام دهد . در مثال زير يك متد به Box اضافه نموده ايم.

```
// This program includes a method inside the box class.
class Box {
double width;
double height;
double depth;
// display volume of a box
void volume () {
System.out.print("Volume is ");
System.out.println(width * height * depth);
} }
class BoxDemo3 {
public static void main(String args[] ){
Box mybox1 = new Box ();
Box mybox2 = new Box ();
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's
instance variables */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
// display volume of first box
mybox1.volume ();
// display volume of second box
mybox2.volume ();
}
}
```

اين برنامه خروجي زير را توليد مي كند:

```
Volume is 3000
Volume is 162
```

از يك نقطه نظر عمومي يك متد شيوه جاوا براي براي پياده سازي زير روالها (subroutines) است. زماني كه يك متد از متغير نمونه اي كه توسط كلاس خود تعريف شده استفاده مي كند، اين كار را بطور مستقيم و بدون ارجاع صريح به يك شي و بدون عملگر نقطه اي انجام مي دهد. يك متد همواره نسبت به شيئي از كلاس خود

قرار داده می شود. زمانی که این تعبیه اتفاق می افتد این شی شناخته شده است. این به آن معنی است که متغیرهای height, depth و width در داخل متد volume() به طور صریح به کپی های در آن شی که این متد را فعال می کند ارجاع می کنند.

فرض کنید که بخش دیگری از برنامه شما قصد فهمیدن فضای اشغالی يك box را دارد و راه بهتر پیاده سازی volume() این است که آن را وادار کنیم که فضای اشغالی box را محاسبه کرده و نتیجه را به فراخواننده برگرداند. مثال بعدی روایت پیشرفته تر برنامه قبلی است که این همین کار را انجام می دهد.

```
// Now/ volume ()returns the volume of a box.
```

```
class Box {
double width;
double height;
double depth;
// compute and return volume
double volume () {
return width * height * depth;
} }
class BoxDemo4 {
public static void main(String args[] ){
Box mybox1 = new Box ();
Box mybox2 = new Box ();
double vol;
// assign values to mybox1's instance variables
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's
instance variables */
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
// get volume of first box
vol = mybox1.volume ();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume ();
System.out.println("Volume is " + vol);
}
}
```

ارسال پارامتر در مثال قبلی ابعاد هر box باید به طور جداگانه با استفاده از يك سلسله دستورات نظیر زیر تعیین گردد :

```
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
```

این کد اگرچه کار می کند اما دو اشکال دارد. اول این که آن بد ترکیب و مستعد خطا می باشد و دوم این که در طراحی خوب برنامه های جاوا باید متغیرهای نمونه فقط از

طریق متدهای تعریف شده توسط کلاس خودشان قابل دسترس باشند. در آینده قادر خواهید بود تا رفتار يك متد را تغییر دهید اما نمی توانید رفتار يك متغیر نمونه بي حفاظ (افشا شده) را تغییر دهید. پس راه بهتر این است که ابعاد Box را به يك متد که وظیفه مقدار دهی را دارد ارسال کنیم. در برنامه زیر این کار نشان داده شده:

```
// This program uses a parameterized method.
class Box {
double width;
double height;
double depth;
// compute and return volume
double volume () {
return width * height * depth;
}
// sets dimensions of box
void setDim(double w, double h, double d) {
width = w;
height = h;
depth = d;
} }
class BoxDemo5 {
public static void main(String args[]) {
Box mybox1 = new Box ();
Box mybox2 = new Box ();
double vol;
// initialize each box
mybox.setDim(10, 20, 15);
mybox.setDim(3, 6, 9);
// get volume of first box
vol = mybox1.volume ();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume ();
System.out.println("Volume is " + vol);
}
}
```

ارسال ارگومان با مقدار و مرجع

در فراخوانی با مقدار (call by value) مقدار يك آرگومان را در پارامتر رسمی متد کپی می کند. بنابراین تغییرات روی پارامتر در متد اعمال می شود تاثیری در روی آرگومانی که روی آن فراخوانی شده ندارد. در روش فراخوانی با مرجع (call by reference) به يك آرگومان (نه مقدار آن آرگومان) به پارامتر پاس (گذر) داده می شود. داخل متد از این ارجاع برای دسترسی به آرگومان واقعی مشخص شده در فراخوانی استفاده می شود و هر گونه تغییر در روی پارامتر روی ارگومان تاثیر خواهد داشت. در جاوا وقتی که يك نوع ساده را به يك متد پاس می کنیم این نوع به وسیله مقدارش پاس می شود. به طور مثال برنامه زیر را در نظر بگیرید:

```
// Simple types are passed by value.
class Test {
void meth(int i, int j ){
i *= 2;
j /= 2;
} }
class CallByValue {
public static void main(String args[] ){
Test ob = new Test ();
int a = 15, b = 20;
System.out.println("a and b before call : " + a + " " + b);
ob.meth(a, b);
System.out.println("a and b after call : " ++ a + " " + b);
}
}
```

خروجي به فرم زیر خواهد بود:

```
a and b before call :15 20
a and b after call :15 20
```

به خوبی مشاهده می کنید که عملیات اتفاق افتاده داخل meth() هیچ تاثیری در روی مقادیر a و b که در فراخوانی استفاده شده است، نخواهد داشت. در این جا مقادیر آنها به 10 و 30 تغییر نمی یابد. بیاد بیاورید که وقتی یک متغیر از یک نوع کلاس ایجاد می کنید، شما فقط یک ارجاع به شی خلق می کنید. بدین ترتیب، وقتی این ارجاع را به یک متد ارسال می کنید پارامتری که آن را دریافت می کند به همان شی ارجاع می کند که توسط آرگومان به آن ارجاع داده شده بود. این به این معنی است که اشیاء با استفاده از روش فراخوانی مرجع تغییرات روی اشیاء در داخل متد سبب تغییر روی شیئی است که به عنوان آرگومان استفاده شده است. مثال زیر را در نظر بگیرید:

```
// Objects are passed by reference.
class Test {
int a, b;
Test(int i, int j ){
a = i;
b = j;
}
// pass an object
void meth(Test o ){
o.a *= 2;
o.b /= 2;
} }
class CallByRef {
public static void main(String args[] ){
Test ob = new Test(15, 20);
System.out.println("ob.a and ob.b before call : " + ob.a + " " + ob.b);
ob.meth(ob);
```

```
System.out.println("ob.a and ob.b after call : " ++ ob.a + " " + ob.b);
}
}
```

برنامه فوق خروجی زیر را تولید می کند:

```
ob.a and ob.b before call :15 20
ob.a and ob.b after call :30 10
```

همان طور که می بینید اعمال داخل متد (meth) شیئی را که به عنوان یک آرگومان استفاده شده تحت تاثیر قرار داده است.

برگرداندن اشیاء

یک متد قادر به این است هر نوع داده شامل انواع کلاسی که ایجاد می کند را برگرداند. به طور مثال در برنامه بعدی متد (incrByTen) یک شیئی را بر می گرداند که در آن مقدار a ده واحد بزرگتر از مقدار آن در شیئی که آن را فراخوانده است.

```
// Returning an object.
class Test {
int a;
Test(int i ){
a = i;
}
Test incrByTen (){
Test temp = new Test(a 10);
return temp;
} }
class RetOb {
public static void main(String args[] ){
Test ob1 = new Test(2);
Test ob2;
ob2 = ob1.incrByTen ();
System.out.println("ob1.a : " + ob1.a);
System.out.println("ob2.a : " + ob2.a);
ob2 = ob2.incrByTen ();
System.out.println("ob2.a after second increase : " + ob2.a);
}
}
```

خروجی به شکل زیر خواهد بود:

```
ob1.a :2
ob2.a :12
ob2.a after second increase :22
```

هر بار که متد (incrByTen) فراخوانده می شود یک شیئی جدید تولید شده و یک ارجاع به آن شیئی جدید به روال فراخواننده برگردان می شود. از آنجایی که کلیه اشیاء به صورت پویا و با استفاده از new تخصیص می یابند، نگرانی راجع شیئی که خارج از قلمرو برورد نخواهید داشت، زیرا در این متدی که شیئی در آن ایجاد شده پایان خواهد

گرفت. يك شي مادامي كه از جايي در برنامه شما ارجاعي به آن جود داشته باشد، زنده خواهد ماند.

انباشت متدها

در شي گراي اين امكان وجود دارد كه دو يا چند متد را در داخل يك كلاس را كه همنام مي باشند تعريف كرد. در جاوا به اين امكان انباشتن متدها مي گويند. انباشتن روشها يكي از جنبه هاي هيچان انگيز و سودمند جاوا است و چند شكلي را پياده مي كند. زماني كه يك متد انباشته شده فراخواني گردد، جاوا از نوع يا شماره آرگومانها بعنوان راهنماي تعيين روايت (version) متد انباشته شده اي كه واقعاً فراخواني مي شود، استفاده مي كند.

به اين ترتيب متدهاي انباشته شده بايد در نوع يا شماره پارامترهايشان متفاوت باشند. زماني كه جاوا با يك فراخواني به يك متد انباشته شده مواجه مي شود، خيلي ساده روايتي از روش را اجرا مي كند كه پارامترهاي آن با آرگومانهاي استفاده شده در فراخواني مطابقت داشته باشند. مثال زير انباشتن متدها را نشان مي دهد.

```
// Demonstrate method overloading.
class OverloadDemo {
void test () {
System.out.println("No parameters");
}
// Overload test for one integer parameter.
void test(int a) {
System.out.println("a : " + a);
}
// Overload test for two integer parameters.
void test(int a, int b) {
System.out.println("a and b : " + a + " " + b);
}
// Overload test for a double parameter.
double test(double a) {
System.out.println("double a : " + a);
return a*a;
} }
class Overload {
public static void main(String args[] ) {
OverloadDemo ob = new OverloadDemo ();
double result;
// call all versions of test ()
ob.test ();
ob.test(10);
ob.test(10, 20);
result = ob.test(123.2);
System.out.println("Result of ob.test(123.2 :) " + result);
}
}
```

خروجي برنامه يه شكل زير خواهد شد:

No parameters

a :10

a and b :10 20

double a :123.2

Result of ob.test(123.2 :)15178.2

همان طور که مشاهده می کنید متد Test() به چهار شکل مختلف نوشته شده است. اولین روایت هیچ پارامتری را نمی گیرد. در دومین روایت یک پارامتر ورودی با مقدار صحیح و در سومین دو پارامتر با نوع صحیح داریم. در آخرین روایت یک پارامتر از نوع double می گیرد. برگرداندن مقداری در آخرین روایت هرگز نتیجه حاصل از عمل انباشت متدها نیست چون انواع برگشتی نقشی در تجزیه و تحلیل انباشت ندارد. زمانی که یک متد انباشته شده فراخوانی می شود، جاوا به دنبال تطبیقی بین آرگمانهای استفاده شده برای فراخوانی متد و پارامترها آن متد می گردد. اما، این تطابق نباید لزوماً همیشه صحیح باشد. در برخی شرایط تبدیل انواع خودکار جاوا می تواند نقشی در تجزیه و تحلیل انباشت داشته باشد. برای مثال برنامه بعدی را در نظر بگیرید.

// Automatic type conversions apply to overloading.

```
class OverloadDemo {
```

```
void test (){
```

```
System.out.println("No parameters");
```

```
}
```

// Overload test for two integer parameters.

```
void test(int a, int b ){
```

```
System.out.println("a and b : " + a + " " + b);
```

```
}
```

// Overload test for a double parameter.

```
double test(double a ){
```

```
System.out.println("Inside test(double )a : " + a);
```

```
} }
```

```
class Overload {
```

```
public static void main(String args[] ){
```

```
OverloadDemo ob = new OverloadDemo ();
```

```
int i = 88;
```

```
ob.test ();
```

```
ob.test(10, 20);
```

```
ob.test(i); // this will invoke test(double)
```

```
ob.test(123.2); // this will invoke test(double)
```

```
}
```

```
}
```

این برنامه خروجی زیر را ایجاد می کند.

a and b :10 20

Inside test(double)a :88

Inside test(double)a :123.2

در کلاس overloadDemo داخل آن متد test(int) موجود نمی باشد. بنابراین هنگامی که متد test() همراه با یک عدد صحیح داخل overload فراخوانی می شود، هیچ

تطبيق دهنده اي پيدا نخواهد شد. اما جاوا مي تواند به طور خودكار يك عدد صحيح را به يك double تبديل كند و اين تبديل براي رفع فراخواني مورد استفاده قرار مي گيرد. بنا بر اين بعد از آنكه test(int) پيدا نشد جاوا i را ارتقاء داده و آنگاه test(double) فراخواني مي شود. جاوا فقط در صورتي كه هيچ تطبيق دقيقي پيدا نكند از تبديل خودكار استفاده مي كند.

لغو(جلوگيري) متد

در يك سلسله مراتب از كلاس و زماني كه يك متد در يك زيركلاس همان نام و نوع يك متد در ابر كلاس خود داشته باشد، آنگاه بيان مي شود كه ان متد در زير كلاس ، متد موجود در ابر كلاس را لغو نموده(يا از پيشروي آمتد جلو گيري مي نمايد). وقتي كه يك متد لغو شده از داخل يك زير كلاس فراخواني مي شود، همواره به روايتي از آن متد كه توسط زير كلاس تعريف شده، ارجاع خواهد نمود و روايتي كه ابر كلاس از همان متد تعريف نموده، پنهان خواهد شد. مورد زير را در نظر بگيريد:

```
// Method overriding.
class A {
    int i, j;
    A(int a, int b ){
        i = a;
        j = b;
    }
    // display i and j
    void show () {
        System.out.println("i and j : " + i + " " + j);
    }
}
class B extends A {
    int k;
    B(int a, int b, int c ){
        super(a, b);
        k = c;
    }
    // display k -- this overrides show () in A
    void show () {
        System.out.println("k : " + k);
    }
}
class Override {
    public static void main(String args[] ){
        B subOb = new B(1, 2, 3);
        subOb.show (); // this calls show () in B
    }
}
```

حاصل ايجاد شده توسط برنامه به شكل زير مي باشد:

k:3

وقتي show() روي يك شي از نوع B فراخواني مي شود، روايتي از show كه داخل B تعريف شده مورد استفاده قرار مي گيرد، يعني روايت اعلان شده در A را لغو مي كند. اگر مي خواهيد به روايت ابر كلاس يك متد لغو شده دسترسي داشته باشيد، اين كار را با استفاده از super انجام دهيد. براي مثال در اين روايت از B روايت ابر كلاس

show() داخل روایت مربوط به زیر کلاس فراخوانی خواهد شد. این امر به کلیه متغیر های نمونه اجازه می دهد تا به نمایش درآیند.

```
class B extends A {
int k;
B(int a, int b, int c ){
super(a, b);
k = c;
}
void show (){
super.show (); // this calls A's show ()
System.out.println("k : " + k);
}
}
```

اگر این روایت از A را در برنامه قبلی جایگزین نمایید، خروجی زیر را مشاهده می کنید :

```
i and j :1 2
k:3
```

در اینجا super.show() روایت کلاس بالایی show() را فراخوانی می کند. لغو متد زمانی اتفاق می افتد که اسامی و نوع دو متد یکسان باشند. اگر چنین نباشد، آنگاه دو متد خیلی ساده انباشته خواهند شد. به عنوان مثال ، این روایت اصلاح شده مثال قبلی را در نظر بگیرید :

```
// Methods with differing type signatures are overloaded -- not
// overridden.
class A {
int i, j;
A(int a, int b ){
i = a;
j = b;
}
// display i and j
void show (){
System.out.println("i and j : " + i + " " + j);
} }
// Create a subclass by extending class A.
class B extends A {
int k;
B(int a, int b, int c ){
super(a, b);
k = c;
}
// overload show ()
void show(String msg ){
System.out.println(msg + k);
} }
class Override {
```

```
public static void main(String args[] ){
B subOb = new B(1, 2, 3);
subOb.show("This is k :"); // this calls show ()in B
subOb.show (); // this calls show ()in A
}
}
```

حاصل ایجاد شده توسط برنامه به شکل زیر می باشد:

```
This is k:3
i and j :1 2
```

روایت show() در B یک پارامتر رشته ای می گیرد. این عمل سبب متفاوت شدن تایید به نوع آن از نوع موجود در A شده، که هیچ پارامتری را نمی گیرد. بنابراین انباشت شدن (یا مخفی شدن اسم) اتفاق نمی افتد.

متدهای Native

در برخی از مواقع گاهی اتفاق می افتد که می خواهید از یک زیر روال (subroutine) نوشته شده توسط سایر زبانهای غیر از جاوا استفاده کنید. معمولاً به دلایل در دسترس بودن روالهای که لازم داریم به زبانهای دیگر یا کار با مورد خاص و سیستمی بودن قسمتی از برنامه از زیر روالهای نوشته شده با سایر زبانها استفاده می کنیم. چنین زیر روالهای به صورت کدهای قابل اجرا برای CPU و محیط خاصی که در آن کار می کنید عرضه می شوند یعنی به صورت کدهای بومی (Native). به عنوان مثال ممکن است که بخواهید یک روال کد بومی را فراخوانی کنید تا به زمان اجرای سریعتر برسید. و با بخواهی از یک کتابخانه تخصصی شده مثل یک بسته آماري استفاده کنید.

اما از آنجایی که برنامه های جاوا به کد بایتی کامپایل می شوند و سپس توسط سیستم در حین اجرای جاوا تفسیر خواهند شد، بنا براین به نظر می رسد فراخوانی یک زیر روال کد بومی از داخل برنامه جاوا غیر ممکن باشد. خوشبختانه، جاوا واژه کلیدی native را فراهم دیده تا از آن برای اعلان متدهای کدهای بومی استفاده شود. هر بار که این متدها اعلان می شوند می توانید آنها را از درون برنامه خود فراخوانی کنید، درست همانند فراخوانی هر متد دیگری در جاوا. برای اعلان یک متد بومی اصلاًحگر native را قبل از متد قرار دهید. اما برای آن بدنه ای تعریف نکنید. بطور مثال:

```
public native int meth ();
```

هر بار که یک متد بومی را اعلان می نماید، مجبورید متد بومی نوشته و یکسری مراحل پیچیده تر را تعقیب کنید تا آن را به کدهای جاوای خود پیوند دهید. به این نکته توجه داشته باشید که مراحل دقیقی که لازم است طی نماید ممکن است برای محیطها و روایتهای گوناگون جاوا متفاوت باشند. همچنین زبانی که مورد استفاده برای پیاده سازی روال بومی موثر می باشد. این بحث از JDK (version 1.02) و ابزار های آن استفاده می کند و محیط windows NT/95 را فرض می کند. زبانی هم که برای پیاده سازی روال بومی استفاده شده زبان C می باشد. آسان ترین روش برای درک این عمل انجام یک کار عملی است. برای شروع برنامه کوتاه زیر را که از یک متد native به نام test() استفاده می کند وارد نماید :

```
// A simple example that uses a native method.
```

```

public class NativeDemo {
int i;
int j;
public static void main(String args[] ){
NativeDemo ob = new NativeDemo ();
ob.i = 10;
ob.j = ob.test (); // call a native method
System.out.println("This is ob.j : " + ob.j);
}
// declare native method
public native int test ();
// load DLL that contains static method
static {
System.loadLibrary("NativeDemo");
}
}

```

اگر به بلوک static دقت کنید، همانطوری که قبلاً گفتیم یک بلوک static فقط یک بار اجرا می شود و آن هم زمانی است که برنامه شما شروع به اجرا می نماید (دقیقتر زمانی که آن کلاس برای اولین بار بار گذاری می شود). در این حالت ، از بلوک static استفاده شده تا کتابخانه پیوند پویا در زمان اجرا (dynamic link library) را که در برگیرنده پیاده سازی روال بومی test() است بار گذاری نماید. کتابخانه فوق توسط متد LoadLibrary() بارگذاری می شود که بخشی از کلاس System است و شکل عمومی آن به فرم زیر است :

Static void LoadLibrary(string filename)

پارامتر filename رشته ای است که نام فایل کتابخانه ای را نگهداری می کند. برای محیط ویندوز فرض شده که پسوند dll را داشته باشد. بعد از وارد کردن برنامه آنرا کامپایل کنید تا NativeDemo.class تولید شود. سپس به کمک java.exe برای ایجاد دو فایل کمک بگیرید. این دو فایل NativeDemo.1 و NativeDemo.C می باشند. شما NativeDemo.h را در پیاده سازی روال test() می گنجانید. فایل NativeDemo.C یک stub (رابط بین برنامه جاوا و روال بومی) را درخود دارد که کدهای ارتباطی در داخل آن موجود می باشند. برای تولید NativeDemo.h از دستور بعدی javah NativeDemo استفاده کنید که این دستور یک فایل سرآیند به نام NativeDemo.h را ایجاد می کند. این فایل همان فایلی است که باید در فایل برنامه به زبان C برای روال test() گنجانده شود.

حاصل تولید شده این فرمان به شکل زیر است :

```

/* DO NOT EDIT THIS FILE - it is machine generated */
#include
/* Header for class NativeDemo */
#ifndef _Included_NativeDemo
#define _Included_NativeDemo
typedef struct ClassNativeDemo {
long j;
long j;
} ClassNativeDemo;
HandleTo(NativeDemo);
#ifdef __cplusplus

```

```
extern "C" {
#ifdef
extern long NativeDemo_test(struct HNativeDemo *);
#endif
}
#endif
#endif
```

چند چیز در این فایل است که باید به آنها دقت نماید. اول اینکه ساختار ClassNativeDemo دو عضو را در بر می گیرد: i و j. این متغیرهای نمونه تعریف شده توسط NativeDemo در فایل جاوا را معین می کند. دوم این که ماکرو HandleTo() نوع ساختار HNativeDemo را ایجاد می کند، که برای ارجاع به شیئی که test() را فراخوانی می کند مورد استفاده قرار می گیرد. به خط بعدی دقت کنید که الگوی برای متد test() ایجاد کرده اید تعریف می کند:

```
extern long NativeDemo_test(struct HNativeDemo *);
```

دقت کنید که نام تابع NativeDemo-test است. باید این را به عنوان نام متد بومی که پیاده سازی می کنید بکار ببرید. یعنی به جای ایجاد یک روال C موسوم به test() یک تابع موسوم به NativeDemo-test() را ایجاد می کنید. پیشوند NativeDemo برای این اضافه شده تا مشخص کننده متد test() که بخشی از کلاس NativeDemo می باشد، باشد. پیشوند گذاری نام متد بومی با نام کلاس فراهم کننده شیوه ای است برای تمایز بین روایتهای مختلف. به عنوان یک قانون عمومی، توابع بومی، یک نام کلاس که در آن اعلان شده اند را به عنوان یک پیشوند قبول می کنند. مقدار برگشتی تابع NativeDemo-test() از نوع long است، اگر چه داخل برنامه جاوا که آن را فراخوانی می کند به عنوان یک int توصیف شده است. دلیل آن این است که در جاوا اعداد صحیح مقادیر 32 بیتی هستند و در زبان C اندازه مقدار long همان 32 بیت می باشد. الگوی NativeDemo-test() یک پارامتر از نوع HNativeDemo* را تعریف می کند. کلیه متدهای بومی حداقل یک پارامتر دارند که اشاره گری است به شیئی که متد بومی را فراخوانی نموده است. این پارامتر ضرورتاً شبیه به this است. می توانید از این پارامتر استفاده کرده تا دسترسی به متغیرهای نمونه شیئی که متد را فراخوانی می کند، داشته باشد. بمنظور تولید فایل stub برای NativeDemo.C از این دستور استفاده کنید:

java-stubs NativeDemo

فایل تولید شده به شکل زیر است:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include
/* Stubs for class NativeDemo */
/* SYMBOL : "NativeDemo/test ()I" / Java_Native_test_stub */
__declspec(dllexport) stack_item
*Java_NativeDemo_test_stub(stack_item *_P / struct execenv *_EE )_{
extern long NativeDemo_test(vide *);
_P[0].i = NativeDemo_test(_P[0].p);
return _P _+ 1;
}
```

شما باید این فایل را کامپایل نموده و با فایل پیاده سازی خود پیوند دهید. پس از تولید فایل‌های سرآیند و stub ضروری، می‌توانید پیاده سازی خود را از test() بنویسید. از روایتی که در زیر نشان داده ایم، استفاده کنید :

```
/* This file contains the C version of the
test ()method.
*/
#include "stubpreamble.h"
#include "NativeDemo.h"
long NativeDemo_test(struct HNativeDemo *this)
{
printf("This is inside the native method.\n");
printf("this->i :%ld\n"/ unhand(this->i);
return( unhand(this->i);
}
```

دقت کنید که این فایل در بر گیرنده stubpreamble.h است که شامل اطلاعات رابط سازی (interfacing) است. این فایل توسط کامپایلر جاوا برای شما فراهم می‌شود. فایل سرآیند NativeDemo قبلاً توسط java ایجاد شده بود. پس از ایجاد test.c باید آن را و NativeDemo.C را کامپایل کنید. بعد این دو فایل را به عنوان یک dll (کتابخانه پویا) با یکدیگر پیوند دهید. به منظور انجام اینکار با کامپایلر میکروسافت C++/C، از خط فرمان دستور CL/LD NativeDemo.ctest.c استفاده کنید. این فرمان یک فایل تحت عنوان NativeDemo.dll تولید می‌کند. هر بار که این کار انجام شود، میتوانیید برنامه جاوا را اجرا نماید. انجام این کار خروجی زیر را تولید می‌کند:

```
This is inside the native method.
this->i :10
This is obj.j :20
```

در داخل test.c به استفاده از متد unhand() دقت نماید. این ماکرو توسط جاوا تعریف شده و یک اشاره گر را به نمونه ساختار ClassNativeDemo که در NativeDemo.h تعریف شده و همراهی شیئی است که test() را فراخوانی می‌کند، بر می‌گرداند. در کل هر گاه نیاز به دسترسی به اعضاء یک کلاس جاوا داشته باید از unhand() استفاده می‌کنید. یادآوری می‌کنیم که استفاه از native بستگی به پیاده سازی و محیط دارند و باید مستندات موجود در سیستم توسعه جاوا خود را مطالعه نمایید تا جزئیات مربوط به روشهای بومی را درک کنید. اگر چه استفاده از متدهای بومی مزیت‌های دسترسی به روالهای کتابخانه ای و اجرای سریعتر بعضی از کارها را فراهم می‌کند اما دو مشکل اساسی را در خود نشان می‌دهد، کاهش امنیت و کاهش قابلیت حمل. یک متد بومی ریسک امنیتی را مطرح می‌کند. از آنجایی که یک متد بومی کد واقعی ماشین را اجرا می‌کند، می‌تواند به هر بخش از رایانه میزبان دسترسی داشته باشد. یعنی که کد بومی منحصر به محیط اجرای جاوا نیست. این کد امکان حمله ویروسی را هم می‌دهد. به همین دلیل ریز برنامه ها نمی‌توانند از روشهای بومی استفاده کنند. همچنین بارگذاری dll ها می‌تواند محدود شود و بار گذاری آنها منوط به تصدیق مدیر امنیتی باشد.

دومین مشکل قابلیت حمل است چون کد بومی داخل يك dll گنجانده شده، باید روی ماشین‌هایی که در حال اجرای برنامه جاوا است حاضر باشد. به علاوه چون هر متد بومی بستگی به CPU و سیستم عامل دارد، هر dll بطور وراثتی غیر قابل حمل می‌شود. به این ترتیب، يك برنامه جاوا که از متدهای بومی استفاده می‌کند، فقط قادر است که روی يك ماشین که برای آن يك dll سازگار نصب شده باشد، اجرا شود.

فصل سوم

فصل سوم

JFC and Swing

در برنامه‌های جاوا به منظور ایجاد رابط گرافیکی کاربر (GUI) و اضافه کردن قابلیت‌های گرافیکی بالا از Java Foundation Class (JFC) استفاده می‌کنیم. کلاس‌های پایه جاوا (JFC) قابلیت‌های مختلف برای ایجاد کردن برنامه‌ها به صورت ویژوال و امکان کار کردن با موس فراهم می‌کنند. JFC شامل کامپوننت‌های Swing، توابع کار با اشیاء دوبعدی (Java 2D API)، پشتیبانی از کشیدن و انداختن (Drag & Drop) و سایر امکانات دیگر می‌باشد.

اصلی‌ترین قابلیت JFC کامپوننت‌های Swing است و از این کامپوننت‌ها برای ایجاد و به کارگیری عناصر ویژوال مثل دکمه‌ها، جعبه‌متن‌ها، قاب‌ها، منوها و سایر گزینه استفاده می‌شود. بطور کلی Swing دارای هفده بسته کلاس‌های قابل استفاده است که آنها را در جدول زیر نشان داده ایم و اکثر برنامه‌های ساده از دو بسته javax.swing و javax.swing.event در کد خود استفاده می‌کنند.

javax.accessibility	javax.swing.plaf	javax.swing.text.html
javax.swing	javax.swing.plaf.basic	javax.swing.text.parser
javax.swing.border	javax.swing.plaf.metal	javax.swing.text.rtf
javax.swing.colorchooser	javax.swing.plaf.multi	javax.swing.tree

javax.swing.event javax.swing.table javax.swing.undo
 javax.swing.filechooser javax.swing.text

اولین برنامه Swing

در این برنامه يك فریم ساده برای نشان دادن يك پیام تولید خواهیم کرد.

```
import javax.swing.*;

public class HelloWorldSwing {
    private static void createAndShowGUI() {
        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);

        //Create and set up the window.
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Add the ubiquitous "Hello World" label.
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

پس از اجرای این برنامه قاب زیر را مشاهده خواهید کرد که با کمک پرچسب يك پیغام Hello World را نشان خواهد داد.



در کدهای زیر نحوه استفاده از کلاس فریم جاوا یا JFrame نوشته نشان داده شده: با کمک اعلان `new JFrame("HelloWorldSwing");` يك فریم با عنوانی که با پارامتر ورودی دریافت می کند ایجاد کرده و به شی ایجاد شده از آن نوع انتساب می دهد

```
JFrame.setDefaultLookAndFeelDecorated(true);
JFrame frame = new JFrame("HelloWorldSwing");
...
frame.pack();
frame.setVisible(true);
```

دستور اول نوشته شده برای مشخص کردن شکل و فرم پنجره ها به همراه لبه های آنها است. دو دستور آخر فریم را آماده و نشان خواهند داد. برای اضافه کردن برچسب از کلاس برچسب جاوا یعنی JLabel که عنوان برچسب را به هنگام ساخت از ورودی می گیرد کمک می گیریم. به منظور اضافه کردن برچسب به فریم ساخته شده از متد add() از getContentPane که در آن نام عنصر را به داخل پاس می کنیم. توجه داشته باشید که به طور مستقیم نمی توان به فریم عنصر اضافه کرد و از content pane برای اضافه کردن استفاده می شود. دلیل این امر این است که کلاس فریم یک ظرف (با ظرفهای EJB اشتباه گرفته نشود) شامل اجزای گرافیکی به جز منو و تعریف پنجره ها است. دستور زیر زمانی که دکمه بستن سمت راست در بالا کلیک می شود برنامه ما را از سیستم خارج می کند.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

به جای دستور بالا در ورژنهای قبلی از دستور زیر استفاده می شده:

```
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```

استفاده از کد زیر به جای فراخوانی مستقیم برای این می باشد که اگر در نخ برنامه اشکالی به وجود بیاید به طور خودکار از برنامه خارج خواهد شد.

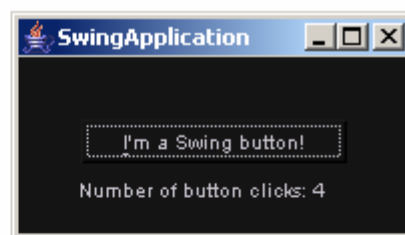
```
javax.swing.SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        /* create and show the GUI */
    }
});
```

Look and Feel

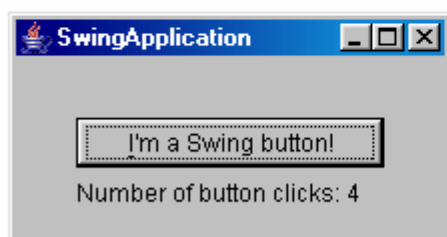
برنامه های ویژوال جاوا می توانند در سیستمهای مختلف با توجه به فرم اشکال در آن سیستم به همان روش نشان داده شوند. همچنین می توانند با یکی از look&feel موجود داخلی جاوا مثل look&feel مخصوص کامپیوترهای مکینتاش کار کنند. چند عدد از این look&feel ها در زیر نشان داده شده. میتوان در جاوا look&feel خاص خود را ایجاد کند مثل look&feel مخصوص محصولات Oracle که اگر محصولات آن را نصب کرده باشید آنرا خواهید دید.



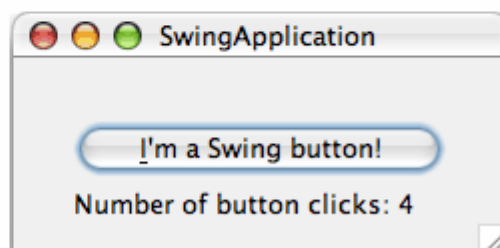
Java look and feel



GTK+ look and feel



Windows look and feel



Mac OS look and feel

کدهای زیر برای مشخص کردن Look&Feel در برنامه به کار می رود و در آن متغیر رشته ای lookAndFeel برای مشخص کردن Look&Feel که به صورت پیش فرض در سیستم می باشد است.

```
String lookAndFeel = null;
...
lookAndFeel =
UIManager.getCrossPlatformLookAndFeelClassName();
...
try {
    UIManager.setLookAndFeel(lookAndFeel);
} catch (Exception e) { }
...// Create and show the GUI...
```

شما می توانید مقدار دلخواه را در داخل این متغیر قرار دهید مثل رشته زیر که Look&Feel مخصوص ویندوز را نشان می دهد.

```
"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
```

در زمان اجرا به کمک کدهای زیر می توانید Look&Feel را تغییر دهید:

```
UIManager.setLookAndFeel(InfName);
SwingUtilities.updateComponentTreeUI(frame);
frame.pack();
```

اضافه کردن دکمه
از کلاس JButton برای اضافه کردن دکمه ها و کنترل اعمال آنها استفاده می کنیم.
نحوه تعریف دکمه به شکل زیر است:

```

JButton button = new JButton("I'm a Swing button!");
button.setMnemonic('i');
button.addActionListener(/*...create an action listener...*/);

```

خط اول يك دکمه با متن که روي آن نوشته مي شود ايجاد مي کند. در خط دوم يك میانبر براي آن کلید به کمک متد `setMnemonic()` ايجاد کرده و با استفاده از `Alt-i` فشرده شدن دکمه را شبیه سازي مي کنیم. کد هاي بعدي نشان دهنده اضافه کردن اجزاء گرافيکي به ظرف مخصوص پانل پيش از اضافه کردن آن به فریم است.

```

JPanel panel = new JPanel(new GridLayout(0,1));
panel.add(button);
panel.add(label);
panel.setBorder(BorderFactory.createEmptyBorder(...));

```

خط اول يك پانل با کلاس `JPanel` تعريف مي کند و اين شي يك مدير طرح بندي (layout manager) است که قابليت تعيين اندازه و وضعيت مکاني هر کدام از اجزا را دارد. کد `new GridLayout(0,1)` ظرف را وادار مي کند که محتوا هر کدام در يك خط نشان داده شده و در همان اندازه تعيين شده باشند. خط آخر براي تعيين شکل لبه پنجره است. در مثال تعداد کلیک کردن دکمه متد `setBorder()` فاصله `Panel` را از چپ، بالا، راست را به تعداد 30 پیکسل و فاصله از پائين را به تعداد 10 پیکسل تنظيم خواهد کرد. در شکل زیر فاصله از چپ و راست نشان داده شده.



دستگیر کردن رخدادها

هر زماني که کاربر دکمه موس را فشار مي دهد يا کارکتری تایپ مي کند يك رخداد اتفاق مي افتد و هر شيء مي تواند از اين رخدادها باخبر شود. براي اين کار اين شي بايد يك `event listener` را تکميل کند. در زي مثال مربوط به کلیک کردن دکمه نشان داده شده.

```

public class SwingApplication implements ActionListener {
    ...
    JButton button = new JButton("I'm a Swing button!");
    button.addActionListener(this);
    ....

    public void actionPerformed(ActionEvent e) {
        numClicks++;
        label.setText(labelPrefix + numClicks);
    }
}

```

هر دستگیر رخداد سه قسمت کد نیاز دارد. قسمت اول دستور کامل کردن رابط شنود کننده (listener)، در مثال بالا در خط اول نشان داده شده. قسمت دوم کد براي ثبت شنود کننده براي آن کامپوننت، که در اين مثال کلاس `JButton` است. در قسمت آخر کد براي اجرا در زماني که رخداد مورد نظر اتفاق مي افتد نوشته مي شود. در اين

مثال زمانی که دکمه کلیک می شود یکی به شمارنده مورد نظر می افزاید و آنرا در برجسب متن نشان می دهد.
 کامپوننتهای Swing انواع رخدادهای را پشتیبانی می کنند که در جدول زیر لیست آنها را مشاهده می کنید.

Act that Results in the Event	Listener Type
زمانی که کاربر کلیدی را فشار اده یا منوی را انتخاب میکند	ActionListener
کاربر یک فریم را می بندد	WindowListener
اگر کاربر دکمه موس را روی یکی از کامپوننتهای گرافیکی فشار دهد	MouseListener
اگر کاربر موس را روی کامپوننتها انتقال دهد	MouseMotionListener
زمانی که کامپوننت قابل دیدن شود	ComponentListener
زمانی که کامپوننت محل تمرکز (focus) صفحه کلید را بدست آورد	FocusListener
زمانی که جدول یا لیست انتخابی تغییر کند	ListSelectionListener
زمانی که هر کدام از خصیصه های (property) کامپوننت گرافیکی تغییر کند.	PropertyChangeListener

توجه داشته باشید که کدهای دستگیر کردن رخدادهای همگی در داخل یک نخ تنها اجرا می شوند و نام آن event-dispatching thread می باشد.

مثال تعداد کلیک کردن دکمه:

```
//SwingApplication.java is a 1.4 example that requires no other files.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class SwingApplication implements ActionListener {
    private static String labelPrefix = "Number of button clicks: ";
    private int numClicks = 0;
    final JLabel label = new JLabel(labelPrefix + "0");

    //Specify the look and feel to use. Valid values:
    //null (use the default), "Metal", "System", "Motif", "GTK+"
    final static String LOOKANDFEEL = null;

    public Component createComponents() {
        JButton button = new JButton("I'm a Swing button!");
        button.setMnemonic(KeyEvent.VK_I);
        button.addActionListener(this);
        label.setLabelFor(button);
    }
}
```

```

/*
 * An easy way to put space between a top-level container
 * and its contents is to put the contents in a JPanel
 * that has an "empty" border.
 */
JPanel pane = new JPanel(new GridLayout(0, 1));
pane.add(button);
pane.add(label);
pane.setBorder(BorderFactory.createEmptyBorder(
    30, //top
    30, //left
    10, //bottom
    30) //right
);

return pane;
}
public void actionPerformed(ActionEvent e) {
    numClicks++;
    label.setText(labelPrefix + numClicks);
}
private static void initLookAndFeel() {
    String lookAndFeel = null;

    if (LOOKANDFEEL != null) {
        if (LOOKANDFEEL.equals("Metal")) {
            lookAndFeel =
UIManager.getCrossPlatformLookAndFeelClassName();
        } else if (LOOKANDFEEL.equals("System")) {
            lookAndFeel = UIManager.getSystemLookAndFeelClassName();
        } else if (LOOKANDFEEL.equals("Motif")) {
            lookAndFeel = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
        } else if (LOOKANDFEEL.equals("GTK+")) { //new in 1.4.2
            lookAndFeel = "com.sun.java.swing.plaf.gtk.GTKLookAndFeel";
        } else {
            System.err.println("Unexpected value of LOOKANDFEEL specified:
"
                                + LOOKANDFEEL);
            lookAndFeel =
UIManager.getCrossPlatformLookAndFeelClassName();
        }

        try {
            UIManager.setLookAndFeel(lookAndFeel);
        } catch (ClassNotFoundException e) {
            System.err.println("Couldn't find class for specified look and feel:"
                                + lookAndFeel);
            System.err.println("Did you include the L&F library in the class
path?");

```

```

        System.err.println("Using the default look and feel.");
    } catch (UnsupportedLookAndFeelException e) {
        System.err.println("Can't use the specified look and feel ("
            + lookAndFeel
            + ") on this platform.");
        System.err.println("Using the default look and feel.");
    } catch (Exception e) {
        System.err.println("Couldn't get specified look and feel ("
            + lookAndFeel
            + "), for some reason.");
        System.err.println("Using the default look and feel.");
        e.printStackTrace();
    }
}

private static void createAndShowGUI() {
    //Set the look and feel.
    initLookAndFeel();

    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    JFrame frame = new JFrame("SwingApplication");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    SwingApplication app = new SwingApplication();
    Component contents = app.createComponents();
    frame.getContentPane().add(contents, BorderLayout.CENTER);
    frame.pack();
    frame.setVisible(true);
}

public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}
}

```

استفاده از برجسبها

به کمک کلاس JLabel شما می توانید متنها و تصاویری که قابل انتخاب نیستند را نمایش داده. اگر می خواهید متنها و تصاویر بصورت محاوره ای بوده و دارای حالت باشند از دکمه ها استفاده کنید. به کمک کدهای html میتوانید برجسبها با رنگها،

قلمها و خطهای مختلف ایجاد کنید. اگر متن شما یک رنگ و قلم دارد شما می توانید از سربار پردازش html اجتناب کرده و از متدهای `setFont()` یا `setForeground()` استفاده کنید. در زیر کدهای مربوط به یک مثال را مشاهده می کنید که سه برجسب را نمایش می دهد و پنجره به سه سطر با ارتفاع یکسان تقسیم شده.

```
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.ImageIcon;
public class LabelDemo extends JPanel {
    public LabelDemo() {
        super(new GridLayout(3,1)); //3 rows, 1 column
        JLabel label1, label2, label3;

        ImageIcon icon = createImageIcon("images/middle.gif",
                                           "a pretty but meaningless splat");
        //Create the first label.
        label1 = new JLabel("Image and Text",
                           icon,
                           JLabel.CENTER);
        //Set the position of its text, relative to its icon:
        label1.setVerticalTextPosition(JLabel.BOTTOM);
        label1.setHorizontalTextPosition(JLabel.CENTER);
        //Create the other labels.
        label2 = new JLabel("Text-Only Label");
        label3 = new JLabel(icon);
        label1.setToolTipText("A label containing both image and text");
        label2.setToolTipText("A label containing only text");
        label3.setToolTipText("A label containing only an image");
        add(label1);
        add(label2);
        add(label3);
    }
    /** Returns an ImageIcon, or null if the path was invalid. */
    protected static ImageIcon createImageIcon(String path,
                                                String description) {
        java.net.URL imgURL = LabelDemo.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL, description);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }
    private static void createAndShowGUI() {
```

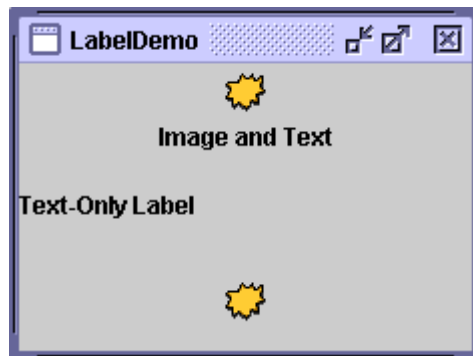
```
//Make sure we have nice window decorations.
JFrame.setDefaultLookAndFeelDecorated(true);
JFrame frame = new JFrame("LabelDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
LabelDemo newContentPane = new LabelDemo();
newContentPane.setOpaque(true); //content panes must be opaque
frame.setContentPane(newContentPane);
//Display the window.
frame.pack();
frame.setVisible(true);
}

public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}
}
```

در اغلب اوقات يك برچسب ساير کامپوننتهاي گرافيکي مثل جعبه متن را شرح مي دهد و شما براي بهبود قدرت دستيابي برنامه خود مي توانيد از متد `setLabelFor()` براي شناساي برچسب آن کامپوننت استفاده کنيد. يك مثال در زير نوشته شده.

```
amountLabel.setLabelFor(amountField);
```

پس از اجراي برنامه خروجي به شکل زير خواهد بود.



مثال تبديل درجه سلسيوس به فارنهايت:

در اين مثال براي تبديل و نشان دادن درجه حرارات سلسيوس به فارنهايت از يك دکمه و جعبه متن به همراه دو برچسب استفاده کرده ايم. کد زير يك جعبه متن در جاوا با عدد 5 براي سازنده ايجاد مي کند. اين عدد تعداد ستونها را در جعبه براي نمايش مناسب آن فونت تنظيم مي کند و تعداد کاراکترها را محدود نمي کند.

```
JTextField tempCelsius = null;
...
tempCelsius = new JTextField(5);
```

زمانی که روی دکمه کلیک می شود یا کلید Enter بر روی صفحه کید فشرده می شود عمل تبدیل انجام شده پس یک شنود کننده اعمال روی دکمه تبدیل و جعبه متن تعریف می کنیم. همانند زیر:

```
convertTemp.addActionListener(this);
tempCelsius.addActionListener(this);
...
public void actionPerformed(ActionEvent event) {
    //Parse degrees Celsius as a double and convert to Fahrenheit.
    int tempFahr = (int)((Double.parseDouble(tempCelsius.getText()))
        * 1.8 + 32);
    fahrenheitLabel.setText(tempFahr + " Fahrenheit");
}
```

پس از فشار دکمه یا زدن کلید Enter متد `getText()` از کلاس جعبه متن عدد وارد شده از طرف کاربر را می دهد. سپس با متد `parseDouble()` عدد متنی را به عدد اعشاری تبدیل می کند. در آخر به کمک متد `setText()` عدد محاسبه شده را به کاربر نشان می دهد. در زیر کد کامل این مثال نشان داده شده.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CelsiusConverter implements ActionListener {
    JFrame converterFrame;
    JPanel converterPanel;
    JTextField tempCelsius;
    JLabel celsiusLabel, fahrenheitLabel;
    JButton convertTemp;

    public CelsiusConverter() {
        //Create and set up the window.
        converterFrame = new JFrame("Convert Celsius to Fahrenheit");
        converterFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        converterFrame.setSize(new Dimension(120, 40));

        converterPanel = new JPanel(new GridLayout(2, 2));

        //Add the widgets.
        addWidgets();

        //Set the default button.
        converterFrame.getRootPane().setDefaultButton(convertTemp);

        //Add the panel to the window.
        converterFrame.getContentPane().add(converterPanel,
            BorderLayout.CENTER);

        //Display the window.
        converterFrame.pack();
    }
}
```

```

        converterFrame.setVisible(true);
    }

    //Create and add the widgets.

    private void addWidgets() {
        //Create widgets.
        tempCelsius = new JTextField(2);
        celsiusLabel = new JLabel("Celsius", SwingConstants.LEFT);
        convertTemp = new JButton("Convert");
        fahrenheitLabel = new JLabel("Fahrenheit", SwingConstants.LEFT);

        //Listen to events from the Convert button.
        convertTemp.addActionListener(this);

        //Add the widgets to the container.
        converterPanel.add(tempCelsius);
        converterPanel.add(celsiusLabel);
        converterPanel.add(convertTemp);
        converterPanel.add(fahrenheitLabel);

        celsiusLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
        fahrenheitLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    }

    public void actionPerformed(ActionEvent event) {
        //Parse degrees Celsius as a double and convert to Fahrenheit.
        int tempFahr = (int)((Double.parseDouble(tempCelsius.getText()))
            * 1.8 + 32);
        fahrenheitLabel.setText(tempFahr + " Fahrenheit");
    }

    private static void createAndShowGUI() {
        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);

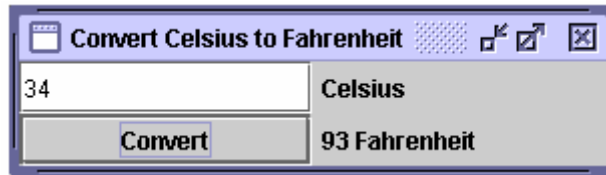
        CelsiusConverter converter = new CelsiusConverter();
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}

```

}

پس از اجرا فرم به شکل زیر خواهد بود.



مثال بهبود یافته مبدل:

در این مثال از متن رنگی و دکمه پیش فرز به همراه icon استفاده کرده ایم. اضافه کردن HTML :

چندین روش برای فرمت دهی اجزای گرافیکی Swing وجود دارد و یکی از این روشها استفاده از تگهای html است. دکمه ها، برچسبها، متنها و سایر کامپوننتها می توانند به کمک کدهای html برای نمایش فرمت دهی شوند. در مثال مبدل پیشرفته تر به کمک تگها می توانیم رنگ و اندازه متن خروجی را تعیین کنیم. همان طور که در کد زیر مشاهده می کنید رنگ متن آبی شده و با 176& نماد درجه از نشان می دهیم.

```
fahrenheitLabel.setText("<html><font Color=blue>" +  
tempFahr + "&#176 </font> Fahrenheit</html>");
```

اضافه کردن يك icon : بعضی از کامپوننتهای Swing این قابلیت را دارند که به همراه يك icon نمایش داده شوند. شی icon رابط برای چسباندن icon به کامپوننتها و با نام ImageIcon است. فرمت تصویرهای icon باید از انواع GIF, JPEG, PNG باشد. کد زیر برای گذاشتن icon در داخل دکمه convertTemp است.

```
ImageIcon convertIcon = createImageIcon("images/convert.gif",  
"Convert temperature");
```

```
...
```

```
convertTemp = new JButton(icon);
```

در متد createImageIcon() اولین آرگومان محل و نام فایل icon برای بار گذاری را مشخص کرده و آرگومان دوم شرح برای استفاده از icon است.

تنظیم دکمه پیش فرز: يك دکمه در ظرف سطح بالا می تواند به عنوان دکمه پیش فرز در نظر گرفته شود. دکمه پیش فرز در نمایش مشخص تر از بقیه دکمه ها است و کلید Enter زده شده صفحه کلید را به طور پیش فرز می گیرد. برای قرار دادن دکمه convertTemp به عنوان دکمه پیشی فرز از کد زیر استفاده می کنیم.

```
converterFrame.getRootPane().setDefaultButton(convertTemp);
```

RootPane در هر ظرف کامپوننتهای گرافیکی برای مدیریت جزئیات مثل پانل محتوا و منوها است.

ایجاد متن فرمت شده: در برنامه قبلی مبدل ما در جعبه متن می توانستیم الفبا را نیز وارد کنیم و این در زمان اجرا يك استثناء صادر می کرد. در این مثال به جای JTextField از JFormattedTextField کمک گرفته ایم که يك راه برای قانونمند کردن مجموعه ورودی های کاربر است. در کد زیر با استفاده از java.text.DecimalFormat مطمئن می شویم که ورودی کاربر فقط اعداد را قبول می کند.

```
//Create the format for the text field and the formatted text field
tempCelsius = new JFormattedTextField(
    new java.text.DecimalFormat("##0.0#"));
```

حال کد کامل این مثال را در پائین مشاهده می نماید.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.NumberFormatter;
import java.text.ParseException;
import java.text.DecimalFormat;
import java.net.URL;

public class CelsiusConverter2 implements ActionListener {
    JFrame converterFrame;
    JPanel converterPanel;
    JFormattedTextField tempCelsius;
    JLabel celsiusLabel, fahrenheitLabel;
    JButton convertTemp;

    public CelsiusConverter2() {
        //Create and set up the window.
        converterFrame = new JFrame("Convert Celsius to Fahrenheit");
        converterFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        converterFrame.setSize(new Dimension(120, 40));

        //Create and set up the panel.
        converterPanel = new JPanel(new GridLayout(2, 2));

        //Add the widgets.
        addWidgets();

        //Set the default button.
        converterFrame.getRootPane().setDefaultButton(convertTemp);

        //Add the panel to the window.
        converterFrame.getContentPane().add(converterPanel,
        BorderLayout.CENTER);

        //Display the window.
        converterFrame.pack();
        converterFrame.setVisible(true);
    }
}
```

```

private void addWidgets() {
    //Create the widgets.
    ImageIcon convertIcon = createImageIcon("images/convert.gif",
        "Convert temperature");

    //Create the format for the Celsius text field.
    tempCelsius = new JFormattedTextField(new
DecimalFormat("##0.0#"));

tempCelsius.setFocusLostBehavior(JFormattedTextField.COMMIT_OR_REVERT
);

    //Set and commit the default temperature.
    try {
        tempCelsius.setText("37.0");
        tempCelsius.commitEdit();
    } catch(ParseException e) {
        //Shouldn't get here unless the setText value doesn't agree
        //with the format set above.
        e.printStackTrace();
    }

    celsiusLabel = new JLabel("Celsius", SwingConstants.LEFT);
    convertTemp = new JButton(convertIcon);

    fahrenheitLabel = new JLabel("Fahrenheit", SwingConstants.LEFT);

    celsiusLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    fahrenheitLabel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

    //Listen to events from the Convert button and
    //the tempCelsius text field.
    convertTemp.addActionListener(this);
    tempCelsius.addActionListener(this);

    //Add the widgets to the container.
    converterPanel.add(tempCelsius);
    converterPanel.add(celsiusLabel);
    converterPanel.add(convertTemp);
    converterPanel.add(fahrenheitLabel);
}

public void actionPerformed(ActionEvent event) {
    String eventName = event.getActionCommand();
    //Parse degrees Celsius as a double and convert to Fahrenheit.
    int tempFahr = (int)((Double.parseDouble(tempCelsius.getText()))
        * 1.8 + 32);
}

```

```

//Set fahrenheitLabel to the new value and set font color
//based on the temperature.
if (tempFahr <= 32) {
    fahrenheitLabel.setText("<html><font Color=blue>" +
        tempFahr + "&#176 </font> Fahrenheit</html>");
} else if (tempFahr <= 80) {
    fahrenheitLabel.setText("<html><font Color=green>" +
        tempFahr + "&#176 </font> Fahrenheit</html>");
} else {
    fahrenheitLabel.setText("<html><font Color=red>" +
        tempFahr + "&#176 </font> Fahrenheit</html>");
}
}

/** Returns an ImageIcon, or null if the path was invalid. */
protected static ImageIcon createImageIcon(String path,
        String description) {
    java.net.URL imgURL = CelsiusConverter2.class.getResource(path);
    if (imgURL != null) {
        return new ImageIcon(imgURL, description);
    } else {
        System.err.println("Couldn't find file: " + path);
        return null;
    }
}

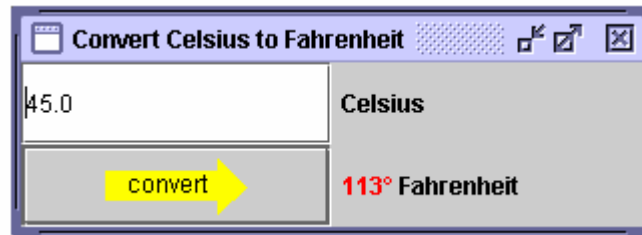
private static void createAndShowGUI() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    CelsiusConverter2 converter = new CelsiusConverter2();
}

public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}
}

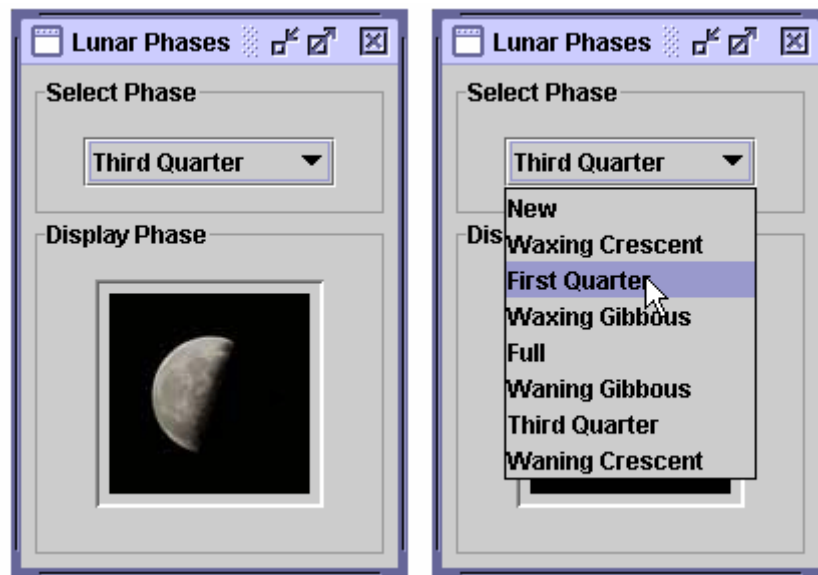
```

بعد از اجرا فرم به شکل زیر ظاهر خواهد شد.

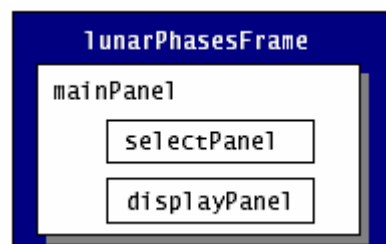


مثال فازهاي ماه

در این برنامه کاربر فاز های مختلف تکمیل شده ماه را از combo box انتخاب می کند و همان طور که در شکل زیر مشاهده میکنید آن فاز در قسمت مربوط به نمایش فاز نشان داده می شود.



این برنامه همان طور که در شکل زیر مشاهده می کنید از سه پانل تشکیل شده.



کدهای که در پائین نشان داده در داخل سازنده برنامه سه پانل ایجاد کرده سو اضافه کردن دو زیر پانل (پانل انتخاب و پانل نمایش فاز) به پانل اصلی است. کار پانل اصلی با استفاده از مدیر طرح بندی (layout manager) قرار دادن موقعیت اجزاء زیر پانلها است. مدیر طرح بند پیش فرز برای JPanel طرح بند FlowLayout است و به طور ساده اجزای گرافیکی را پشت سر هم می چیند.

```
//Create the phase selection and display panels.
selectPanel = new JPanel();
```

```
displayPanel = new JPanel();

//Add various widgets to the sub panels.
addWidgets();

//Create the main panel to contain the two sub panels.
mainPanel = new JPanel();
mainPanel.setLayout(new BorderLayout(mainPanel, BorderLayout.PAGE_AXIS));
mainPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));

//Add the select and display panels to the main panel.
mainPanel.add(selectPanel);
mainPanel.add(displayPanel);
```

در کدهای بالا از مدیر طرح بندی BorderLayout برای تنظیم دقیق زیر پانل ها استفاده کرده ایم.

استفاده از مدیر طرح بندی:

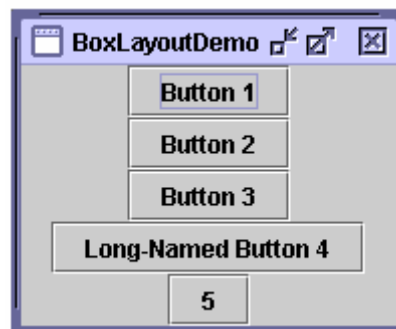
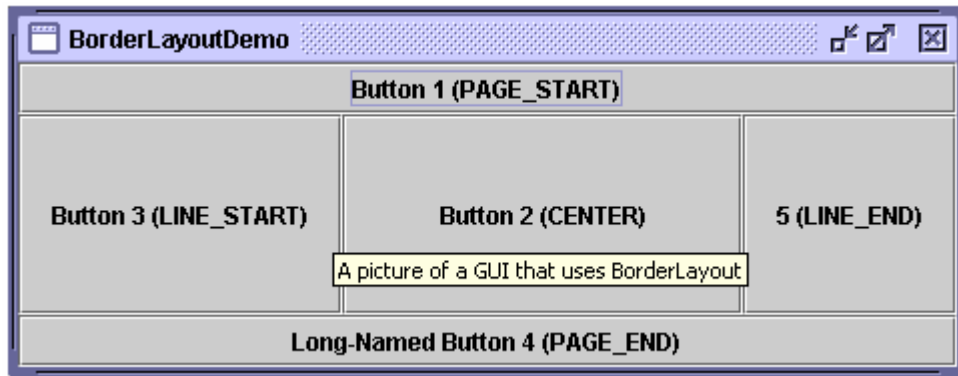
در جاوا از سه نوع مدیر طرح بند شامل BorderLayout، BoxLayout، و GridLayout، GridBagLayout و SpringLayout پشتیبانی می شود. همان طور که قبلاً گفته شد به طور پیش فرض تمام اشیاء از نوع JPanel به طور پیش فرض از BorderLayout استفاده می کنند. همین طور سایر نگه دارنده های محتوا مثل ظرف اصلی در اشیاء JApplet، JDialog و JFrame به صورت پیش فرض از BorderLayout استفاده می کند. زمانی که شما درباره مدیر طرح بندی تفکر می کنید یک روش وجود دارد و آن قاعده چگونه گی ایجاد یک JPanel یا اضافه کردن کامپوننت برای یک نگه دارنده محتوا می باشد. یک روش دیگر متفاوت استفاده از متد setLayout() برای مشخص کردن نحوه کار نگه دارنده محتوا است. برای مثال کد زیر ایجاد پانل به کمک BorderLayout را نشان داده:

```
JPanel pane = new JPanel(new BorderLayout());
```

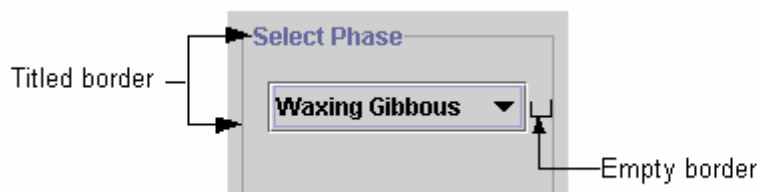
مثال زیر درباره تنظیم مدیر طرح بندی نگه دارنده محتوای پیش فرض است.

```
Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
```

زمانی که شما به یک پانل یا نگه دارنده محتوا کامپوننت گرافیکی اضافه می کنید، شما می توانید روش اضافه کردن را جدا از مدیر طرح بندی تعیین کنید. در زیر سه مثال از انواع مدیر طرح بندی شامل BorderLayout، BoxLayout و FlowLayout برای پنج دکمه در یک فریم به صورت پشت سر هم نشان داده ایم:



لبه های مرکب: در مثالهای گذشته ما لبه های ساده را برای ایجاد یک میانگیر فاصله بین دور اجزاء استفاده کردیم. در این مثال هر دو زیر پانل selectPanel و displayPanel از لبه های مرکب استفاده کرده و شامل لبه عنوان دار (شامل لبه حاشیه به همراه یک عنوان) و لبه خالی (یک لبه با فضای اضافی) است. این دو لبه در شکل زیر نشان داده شده.



کد مورد نیاز برای selectPanel به فرم زیر است. پانل displayPanel به همین روش لبه خود را تنظیم می کند.

```
// Add border around the select panel
selectPanel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createTitledBorder("Select Phase"),
    BorderFactory.createEmptyBorder(5,5,5,5)));
```

Combo Box : در این مثال از جعبه متن ترکیبی (combo box) برای نشان دادن یک گروه از انتخابهای استفاده می کنیم. بجای combo box می توان از جعبه متن برای نوشتن هر کدام استفاده کرد. کدهای زیر در مثال یک combo box غیر قابل ویرایش را ایجاد می کند.

```
JComboBox phaseChoices = null;
...
//Create combo box with lunar phase choices.
String[] phases = { "New", "Waxing Crescent", "First Quarter",
                    "Waxing Gibbous", "Full", "Waning Gibbous",
                    "Third Quarter", "Waning Crescent" };
phaseChoices = new JComboBox(phases);
phaseChoices.setSelectedIndex(START_INDEX);
```

شما به جای آرایه ای از رشته می توانید از آرایه ای از icon ها یا ساختار داده منظم در ایجاد combo box استفاده کنید. متد setSelectedIndex() مشخص می کند که در زمان شروع اجرای برنامه کدام یک از قلمها باید نمایش داده شود. زمانی که کاربر یک قلم را از combo box انتخاب می کند یک رخداد اتفاق می افتد و کد زیر در مثال وظیفه ثبت و کامل کردن یک شنود کننده رخداد روی combo box را دارد. زمانی که قلم جدید انتخاب می شود، تصویر این قلم نشان داده شده و برچسب به روز رسانی می شود.

```
phaseChoices.addActionListener(this);
...
public void actionPerformed(ActionEvent event) {
    if ("comboBoxChanged".equals(event.getActionCommand())) {
        //Update the icon to display the new phase
        phaseIconLabel.setIcon(images[phaseChoices.getSelectedIndex()]);
    }
}
```

تصاویر چند گانه: در مثال قبلی نحوه اضافه کردن یک تصویر تنها به یک دکمه را مشاهده کردید. در این مثال از هشت تصویر استفاده شده و در هر زمان تصویر یکی از آنها به نمایش در می آید. همچنین می توانیم تمام این تصاویر یکجا بار گذاری کرده یا در زمان لازمه یک از آنها را بار گذاری کنیم. در زیر نحوه بار گذاری یکجای هشت تصویر را ملاحظه می کنید.

```
final static int NUM_IMAGES = 8;
final static int START_INDEX = 3;

ImageIcon[] images = new ImageIcon[NUM_IMAGES];
...
//Get the images and put them into an array of ImageIcon.
for (int i = 0; i < NUM_IMAGES; i++) {
    String imageName = "images/image" + i + ".jpg";
    System.out.println("getting image: " + imageName);
    URL iconURL = LunarPhases.class.getResource(imageName);
    ImageIcon icon = new ImageIcon(iconURL);
```

```
images[i] = icon;
}
```

استفاده از متد `setResource()` برای جستجوی مسیر کلاسهای برنامه بمنظور پیدا کردن فایل‌های تصویر است. کد کامل برنامه در زیر نوشته شده.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.URL;

public class LunarPhases implements ActionListener {
    final static int NUM_IMAGES = 8;
    final static int START_INDEX = 3;
    ImageIcon[] images = new ImageIcon[NUM_IMAGES];
    JPanel mainPanel, selectPanel, displayPanel;
    JComboBox phaseChoices = null;
    JLabel phaseIconLabel = null;

    public LunarPhases() {
        //Create the phase selection and display panels.
        selectPanel = new JPanel();
        displayPanel = new JPanel();

        //Add various widgets to the sub panels.
        addWidgets();
        //Create the main panel to contain the two sub panels.
        mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel,
        BoxLayout.PAGE_AXIS));
        mainPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
        //Add the select and display panels to the main panel.
        mainPanel.add(selectPanel);
        mainPanel.add(displayPanel);
    }

    // Get the images and set up the widgets.
    private void addWidgets() {
        //Get the images and put them into an array of ImageIcons.
        for (int i = 0; i < NUM_IMAGES; i++) {
            images[i] = createImageIcon("/images/image" + i + ".jpg");
        }

        //Create a label for displaying the moon phase images
        phaseIconLabel = new JLabel();
        phaseIconLabel.setHorizontalAlignment(JLabel.CENTER);
        phaseIconLabel.setVerticalAlignment(JLabel.CENTER);
        phaseIconLabel.setVerticalTextPosition(JLabel.CENTER);
    }
}
```

```

phaseIconLabel.setHorizontalTextPosition(JLabel.CENTER);
phaseIconLabel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLoweredBevelBorder(),
    BorderFactory.createEmptyBorder(5,5,5,5)));

phaseIconLabel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createEmptyBorder(0,0,10,0),
    phaseIconLabel.getBorder()));

//Create a combo box with lunar phase choices.
String[] phases = { "New", "Waxing Crescent", "First Quarter",
    "Waxing Gibbous", "Full", "Waning Gibbous",
    "Third Quarter", "Waning Crescent" };
phaseChoices = new JComboBox(phases);
phaseChoices.setSelectedIndex(START_INDEX);

//Display the first image.
phaseIconLabel.setIcon(images[START_INDEX]);
phaseIconLabel.setText("");

//Add a border around the select panel.
selectPanel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createTitledBorder("Select Phase"),
    BorderFactory.createEmptyBorder(5,5,5,5)));
//Add a border around the display panel.
displayPanel.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createTitledBorder("Display Phase"),
    BorderFactory.createEmptyBorder(5,5,5,5)));
//Add moon phases combo box to select panel and image label.
displayPanel.add(phaseIconLabel);
selectPanel.add(phaseChoices);
//Listen to events from the combo box.
phaseChoices.addActionListener(this);
}

public void actionPerformed(ActionEvent event) {
    if ("comboBoxChanged".equals(event.getActionCommand())) {
        //Update the icon to display the new phase.
        phaseIconLabel.setIcon(images[phaseChoices.getSelectedIndex()]);
    }
}

/** Returns an ImageIcon, or null if the path was invalid. */
protected static ImageIcon createImageIcon(String path) {
    java.net.URL imageURL = LunarPhases.class.getResource(path);

    if (imageURL == null) {
        System.err.println("Resource not found: "

```

```

        + path);
    return null;
} else {
    return new ImageIcon(imageURL);
}
}

private static void createAndShowGUI() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);

    //Create a new instance of LunarPhases.
    LunarPhases phases = new LunarPhases();

    //Create and set up the window.
    JFrame lunarPhasesFrame=new JFrame("Lunar Phases");
    lunarPhasesFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    lunarPhasesFrame.setContentPane(phases.mainPanel);

    //Display the window.
    lunarPhasesFrame.pack();
    lunarPhasesFrame.setVisible(true);
}

public static void main(String[] args) {
    //Schedule a job for the event-dispatching thread:
    //creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}
}

```

مثال رای دادن:

در این مثال بیشتر روی دکمه های رادیویی (radio buttons) و جعبه های گفتگو (dialogs) کار خواهیم کرد. در این مثال کابر می تواند یک دکمه رادیویی را انتخاب کرده و به کمک دکمه رای یک رای را صادر کند. سپس یک جعبه گفتگو با پیام ظاهر شده و اطلاعات را نشان می دهد.

دکمه های رادیویی: در این مثال یک شنود کننده رخداد کلیک کردن دکمه رای را در ظرف کنترل می کند و هر زمان که آن کلیک شد برنامه تعیین می کند که کدام دکمه رادیویی کلیک شده و دستورات لازم آن را انجام می دهد. برای هر گروه از دکمه های رادیویی شما نیاز دارید به ایجاد نمونه شیء از ButtonGroup و اضافه کردن هر دکمه رادیویی به داخل آن. این شیء دکمه رادیویی را که قبلاً انتخاب شده را زمانی که دکمه رادیویی دیگری انتخاب می کنیم غیر فعال می کند. دکمه های رادیویی می توانند دکمه انتخاب شده پیش فرز نداشته باشند.

کدهاي زير دکمه هاي راديويي و شي ButtonGroup را ايجاد کرده و از html براي فرمت دهی متنها استفاده می کند. متد setActionCommand() با توجه به قلم هر دکمه راديويي متن جعبه گفتگو را مشخص می کند.

```
final int numButtons = 4;
JRadioButton[] radioButtons = new JRadioButton[numButtons];
final ButtonGroup group = new ButtonGroup();
...
final String defaultMessageCommand = "default";
final String yesNoCommand = "yesno";
final String yeahNahCommand = "yeahnah";
final String yncCommand = "ync";

radioButtons[0] = new JRadioButton("<html>Candidate 1:
    <font color=red>Sparky the Dog</font></html>");
radioButtons[0].setActionCommand(defaultMessageCommand);

radioButtons[1] = new JRadioButton("<html>Candidate 2:
    <font color=green>Shady Sadie</font></html>");
radioButtons[1].setActionCommand(yesNoCommand);

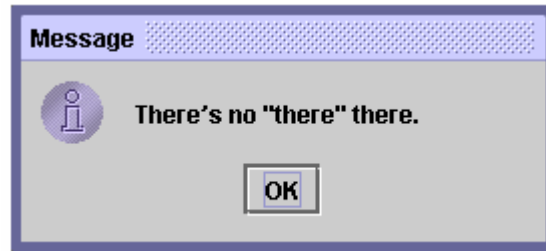
radioButtons[2] = new JRadioButton("<html>Candidate 3:
    <font color=blue>R.I.P. McDaniels</font></html>");
radioButtons[2].setActionCommand(yeahNahCommand);

radioButtons[3] = new JRadioButton("<html>Candidate 4:
    <font color=maroon>Duke the Java<font size=-2><sup>TM</sup>
    </font size> Platform Mascot</font></html>");
radioButtons[3].setActionCommand(yncCommand);

for (int i = 0; i < numButtons; i++) {
    group.add(radioButtons[i]);
}
//Select the first button by default.
radioButtons[0].setSelected(true);
```

جعبه هاي گفتگو: در مثالهاي قبلي ظرف سطح بالاي شما هميشه يك JFrame بود. براي ايجاد جعبه هاي گفتگوي مختلف و استاندارد بايد از کلاس JOptionPane استفاده کرد و اين جعبه هاي گفتگو بصورت مقيد (modal) می باشند. زمانی که يك جعبه گفتگوي modal در حال نمایش است از وارد کردن اطلاعات کاربر به ساير پنجره هاي برنامه جلوگیری می شود. کد زير يك مثال کوچک در باره کلاس JOptionPane است و جعبه گفتگوي حاصل از آن نشان داده شده.

```
JOptionPane.showMessageDialog(frame, "There's no \"there\" there.");
```



```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class VoteDialog extends JPanel {
    JLabel label;
    JFrame frame;
    String simpleDialogDesc = "The candidates";
    public VoteDialog(JFrame frame) {
        super(new BorderLayout());
        this.frame = frame;
        JLabel title;
        //Create the components.
        JPanel choicePanel = createSimpleDialogBox();
        System.out.println("passed createSimpleDialogBox");
        title = new JLabel("Click the \"Vote\" button"
            + " once you have selected a candidate.",
            JLabel.CENTER);
        label = new JLabel("Vote now!", JLabel.CENTER);
        label.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        choicePanel.setBorder(BorderFactory.createEmptyBorder(20,20,5,20));
        //Lay out the main panel.
        add(title, BorderLayout.NORTH);
        add(label, BorderLayout.SOUTH);
        add(choicePanel, BorderLayout.CENTER);
    }

    void setLabel(String newText) {
        label.setText(newText);
    }

    private JPanel createSimpleDialogBox() {
        final int numButtons = 4;
        JRadioButton[] radioButtons = new JRadioButton[numButtons];
        final ButtonGroup group = new ButtonGroup();
        JButton voteButton = null;
        final String defaultMessageCommand = "default";
        final String yesNoCommand = "yesno";
        final String yeahNahCommand = "yeahnah";
        final String yncCommand = "ync";
    }
}

```

```

radioButtons[0] = new JRadioButton(
    "<html>Candidate 1: <font color=red>Sparky the
Dog</font></html>");
radioButtons[0].setActionCommand(defaultMessageCommand);

radioButtons[1] = new JRadioButton(
    "<html>Candidate 2: <font color=green>Shady
Sadie</font></html>");
radioButtons[1].setActionCommand(yesNoCommand);

radioButtons[2] = new JRadioButton(
    "<html>Candidate 3: <font color=blue>R.I.P.
McDaniels</font></html>");
radioButtons[2].setActionCommand(yeahNahCommand);

radioButtons[3] = new JRadioButton(
    "<html>Candidate 4: <font color=maroon>Duke the Java<font
size=-2><sup>TM</sup></font size> Platform Mascot</font></html>");
radioButtons[3].setActionCommand(yncCommand);

for (int i = 0; i < numButtons; i++) {
    group.add(radioButtons[i]);
}
//Select the first button by default.
radioButtons[0].setSelected(true);

voteButton = new JButton("Vote");
voteButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String command = group.getSelection().getActionCommand();

        //ok dialog
        if (command == defaultMessageCommand) {
            JOptionPane.showMessageDialog(frame,
                "This candidate is a dog. Invalid vote.");

            //yes/no dialog
        } else if (command == yesNoCommand) {
            int n = JOptionPane.showConfirmDialog(frame,
                "This candidate is a convicted felon. \nDo you still want to
vote for her?",
                "A Follow-up Question",
                JOptionPane.YES_NO_OPTION);
            if (n == JOptionPane.YES_OPTION) {
                setLabel("OK. Keep an eye on your wallet.");
            } else if (n == JOptionPane.NO_OPTION) {
                setLabel("Whew! Good choice.");
            }
        }
    }
});

```

```

        } else {
            setLabel("It is your civic duty to cast your vote.");
        }
//yes/no (with customized wording)
    } else if (command == yeahNahCommand) {
        Object[] options = {"Yes, please", "No, thanks"};
        int n = JOptionPane.showOptionDialog(frame,
            "This candidate is deceased. \nDo you still want to vote for
him?",
            "A Follow-up Question",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null,
            options,
            options[0]);
        if (n == JOptionPane.YES_OPTION) {
            setLabel("I hope you don't expect much from your
candidate.");
        } else if (n == JOptionPane.NO_OPTION) {
            setLabel("Whew! Good choice.");
        } else {
            setLabel("It is your civic duty to cast your vote.");
        }
//yes/no/cancel (with customized wording)
    } else if (command == yncCommand) {
        Object[] options = {"Yes!",
            "No, I'll pass",
            "Well, if I must"};
        int n = JOptionPane.showOptionDialog(frame,
            "Duke is a cartoon mascot. \nDo you "
            + "still want to cast your vote?",
            "A Follow-up Question",
            JOptionPane.YES_NO_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE, null, options,
            options[2]);
        if (n == JOptionPane.YES_OPTION) {
            setLabel("Excellent choice.");
        } else if (n == JOptionPane.NO_OPTION) {
            setLabel("Whatever you say. It's your vote.");
        } else if (n == JOptionPane.CANCEL_OPTION) {
            setLabel("Well, I'm certainly not going to make you vote.");
        } else {
            setLabel("It is your civic duty to cast your vote.");
        }
    }
}
return;
}
});

```

```

        System.out.println("calling createPane");
        return createPane(simpleDialogDesc + ":",
                           radioButton,
                           voteButton);
    }

    private JPanel createPane(String description,
                              JRadioButton[] radioButtons,
                              JButton showButton) {
        int numChoices = radioButtons.length;
        JPanel box = new JPanel();
        JLabel label = new JLabel(description);

        box.setLayout(new BoxLayout(box, BoxLayout.PAGE_AXIS));
        box.add(label);

        for (int i = 0; i < numChoices; i++) {
            box.add(radioButtons[i]);
        }
        JPanel pane = new JPanel(new BorderLayout());
        pane.add(box, BorderLayout.NORTH);
        pane.add(showButton, BorderLayout.SOUTH);
        System.out.println("returning pane");
        return pane;
    }

    private static void createAndShowGUI() {
        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);
        JDialog.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new GridLayout(1,1));
        contentPane.add(new VoteDialog(frame));

        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}

```

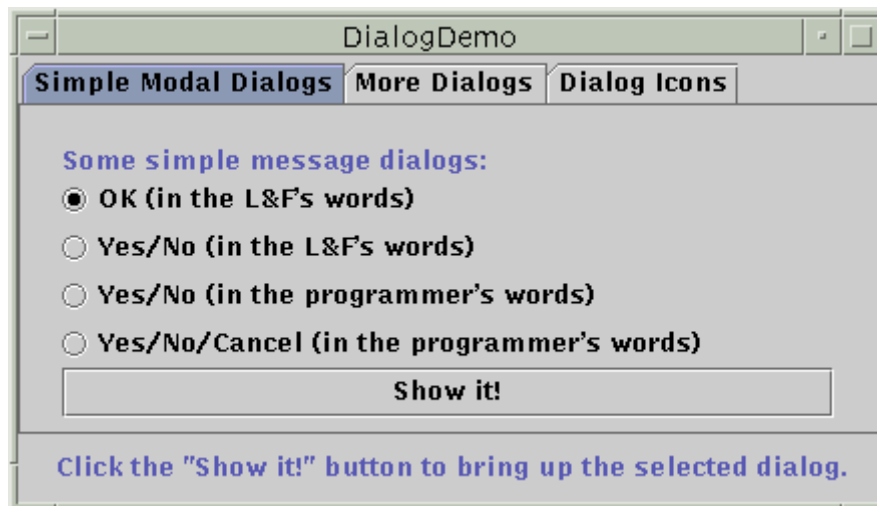
پس از اجرا فرم به صورت زیر ظاهر خواهد شد.



نحوه ایجاد جعبه های گفتگو

همان طور که گفته شد برای ایجاد جعبه های گفتگوی محاوره‌ای از کلاس استاندارد JOptionPane استفاده می‌کنیم. کلاس ProgerssMonitor همانند جعبه گفتگو وضعیت پیشرفت اعمال را نشان می‌دهد. همچنین دو کلاس JColorChooser و JFileChooser جعبه های محاوره ای استاندارد برای انتخاب رنگ و فایل هستند. هر جعبه گفتگو وابسته به يك frame است و زمانیکه شیء frame خراب می‌شود (از حافظه خارج می‌شود) جعبه های گفتگوی وابسته به آن هم خراب می‌شوند. جعبه های گفتگو بصورت مقید (modal) می‌باشند یعنی زمانی که يك جعبه گفتگوی modal در حال نمایش است از وارد کردن اطلاعات کاربر به سایر پنجره های برنامه جلوگیری می‌شود.

کلاس JDialog يك زیر کلاس از کلاس java.awt.Dialog بوده و يك جعبه Dialog را به نگه دارنده محتوای ریشه اضافه می‌کند و عمل بستن پیش‌فرز را پشتیبانی می‌کند. استفاده از کلاس JDialog بسیار شبیه به استفاده از کلاس JFrame بوده و همان قابلیت‌ها را دارد. زمانی که از JOptionPane برای جعبه های گفتگو استفاده می‌کنید در پیش‌زمینه همانند این است که از JDialog استفاده می‌کنید. در شکل زیر يك Dialog نمونه از يك برنامه را مشاهده می‌کنید.



قابلیتهای JOptionPane

با استفاده از این کلاس می‌توانید چندین نوع از جعبه‌های گفتوپی مختلف و دلخواه را بسازید و مشخصات آنها را مانند icon های جعبه، عنوان و متن جعبه، دکمه‌ها به طور دلخواه تعیین کنید. همچنین این امکان نیز وجود دارد که مکان نمایش آنها را نیز به طور دلخواه مشخص کنید. حتی می‌توان جعبه گفتگوی option pane را طوری تنظیم کرد که به جای گذاشتن آن در JDialog آن را در یک فریم داخلی یا JInternalFrame گذاشت. زمانیکه یک جعبه گفتگو ایجاد می‌شود به طور خودکار کد های مربوط به Look&Feel برای طرح بندی (layout) مربوط به طرف اصلی به آن اضافه می‌شود. کلاس JOptionPane از چهار icon برای چهار حالت سوال، اطلاعات، اخطار و خطا استفاده می‌کند. در شکل زیر چهار icon را مشاهده می‌کنید.



ایجاد و نمایش جعبه‌های گفتگوی ساده:

بمنظور ایجاد و نمایش جعبه‌های گفتگو از متد `showXxxDialog()` در کلاس `JOptionPane` استفاده کرده و اگر جعبه گفتگو به صورت یک فریم داخلی است یک `Internal` به آن اضافه می‌کنید همانند `showInternalMessageDialog`. اگر شما نیاز به کنترل جعبه گفتگو در هنگام بستن پنجره دارید و یا می‌خواهید جعبه بصورت modal نباشد شما می‌توانید به صورت مستقیم از معرفی `JOptionPane` استفاده کرده و آن را به یک نمونه `JDialog` اضافه کرده و به کمک متد `setVisible(true)` ظاهر شدن آن را کنترل کنید.

دو متد مفید و پر استفاده از `showXxxDialog` متدهای `showMessageDialog` و `showOptionDialog` هستند. متد اول به صورت ساده اطلاعات را به همراه یک دکمه نشان داده. متد دوم یک جعبه گفتگوی دلخواه را نشان داده که می‌تواند شامل دکمه‌ها، متن‌ها و اجزای گرافیکی دلخواه باشد. دو متد باقی مانده که کمتر از دو متد قبلی استفاده می‌شوند متدهای `showInputDialog` و `showConfirmDialog` هستند. متد اول برای سوال کردن از کاربر در باره تأیید چیزی است و دکمه‌های محدودی مثل `yes/no` دارد. متد دوم بمنظور گرفتن اطلاعات از کاربر به شکلهای متنی یا از `combo box` به صورت modal است.

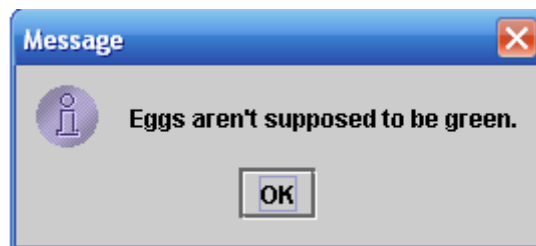
در اغلب این متدها اولین آرگومان نام frame یا ظرف نگهدارنده محتوا است و دومین آرگومان پیغامی که به کاربر نمایش داده خواهد شد است. سومین آرگومان همیشه نوع پیغام را مشخص می کند و همچنین چهارمین پارامتر icon را برای جعبه گفتگوی مورد نظر تعیین می کند.

حال به چند مثال درباره هر کدام از این متدها به همراه شکل‌های آنها در زیر می پردازیم.

: showMessageDialog

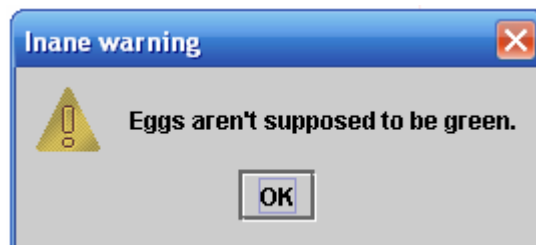
//default title and icon

```
JOptionPane.showMessageDialog(frame,"Eggs aren't supposed to be green.");
```



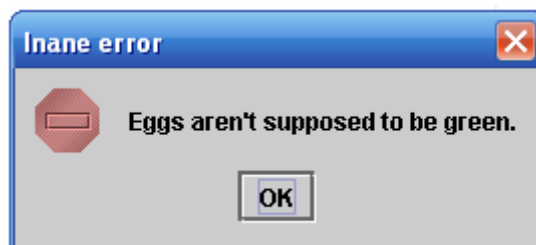
//custom title, warning icon

```
JOptionPane.showMessageDialog(frame,"Eggs aren't supposed to be green.",  
    "Inane warning",  
    JOptionPane.WARNING_MESSAGE);
```



//custom title, error icon

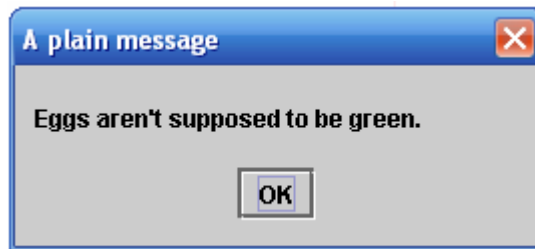
```
JOptionPane.showMessageDialog(frame,"Eggs aren't supposed to be green.",  
    "Inane error",  
    JOptionPane.ERROR_MESSAGE);
```



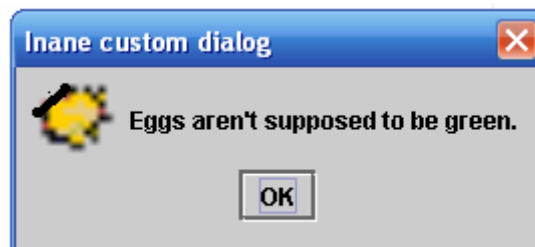
//custom title, no icon

```
JOptionPane.showMessageDialog(frame,"Eggs aren't supposed to be green.",  
    "A plain message",
```

```
JOptionPane.PLAIN_MESSAGE);
```

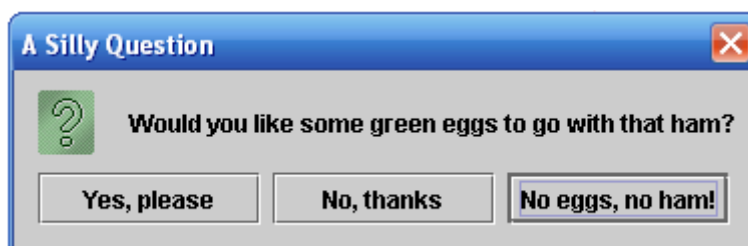


```
//custom title, custom icon
JOptionPane.showMessageDialog(frame,"Eggs aren't supposed to be green.",
    "Inane custom dialog",
    JOptionPane.INFORMATION_MESSAGE,icon);
```



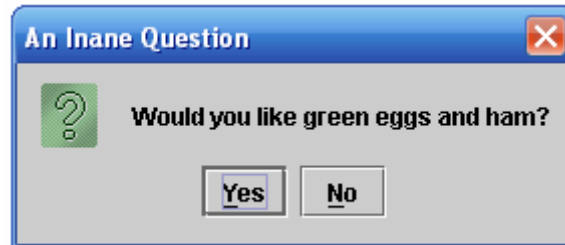
: showOptionDialog

```
//Custom button text
Object[] options = {"Yes, please",
    "No, thanks",
    "No eggs, no ham!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like some green eggs to go "
    + "with that ham?",
    "A Silly Question",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[2]);
```

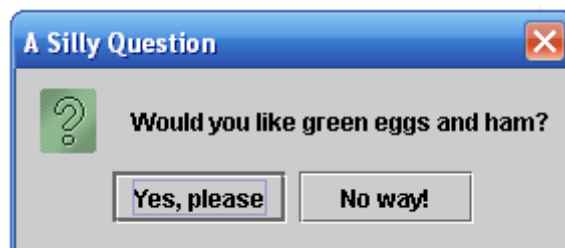


: JOptionPane

```
//default icon, custom title
int n = JOptionPane.showConfirmDialog(
    frame, "Would you like green eggs and ham?",
    "An Inane Question", JOptionPane.YES_NO_OPTION);
```

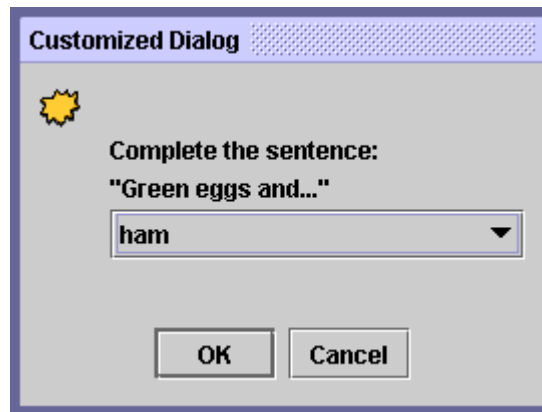


```
Object[] options = {"Yes, please",
    "No way!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like green eggs and ham?",
    "A Silly Question",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null, //don't use a custom Icon
    options, //the titles of buttons
    options[0]); //default button title
```

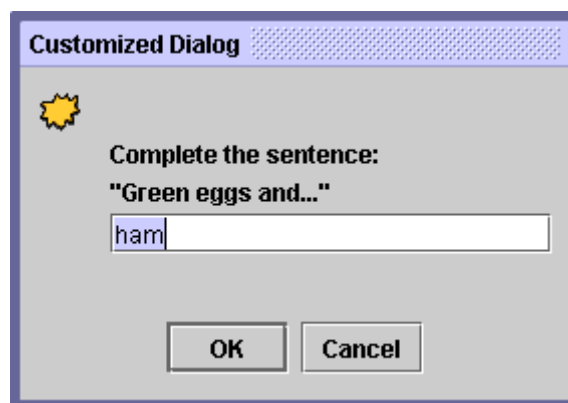


: showInputDialog

```
Object[] possibilities = {"ham", "spam", "yam"};
String s = (String)JOptionPane.showInputDialog(
    frame,
    "Complete the sentence:\n"
    + "\"Green eggs and...\\"",
    "Customized Dialog",
    JOptionPane.PLAIN_MESSAGE,
    icon,
    possibilities,
    "ham");
```

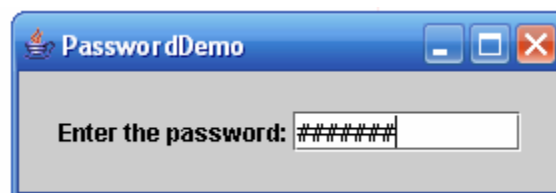


اگر در متد بالا آرایه possibilities را حذف کنیم جعبه گفتگوي ورودی به شکل زیر خواهد بود.



گرفتن Password

یکی از زیر کلاسهای کلاس JTextField کلاس JPasswordField می باشد. از این کلاس برای گرفتن کلمه های عبور در برنامه های که امنیت بالا نیاز دارند استفاده می شود. در هنگام وارد کردن کلمه عبور کارکترهای که کاربر وارد می کند را نشان نمی دهد و به جای کارکترها یک کارکتر ثابت نشان می دهد. یک فرم که در آن از JPasswordField استفاده شده در زیر مشاهده می نماید.



کدهای زیر نحوه ایجاد و تنظیم یک password field را نشان می دهد.

```
passwordField = new JPasswordField(10);
passwordField.setEchoChar('#');
passwordField.addActionListener(this);
```

عدد 10 در اولین خط نشان دهنده حداکثر طول کلمه عبور است. کارکتر # در داخل متد setEchoChar() کارکتر ظاهر شونده به جای کلمه عبور را مشخص می کند.

نحوه استفاده از فریم های داخلی و منوها

به کمک کلاس JInternalFrame شما می توانید یک JFrame را در داخل یک فریم دیگر نمایش دهید. معمولاً یک فریم داخلی به یک desktop pane اضافه می شود. خود desktop pane به عنوان یک نگه دارنده محتوا در یک JFrame استفاده می شود و یک نمونه از شیء JDesktopPane می باشد. کلاس JDesktopPane یک زیر کلاس از JLayeredPane است و در داخل JLayeredPane یک سری API برای فریم های داخلی موجود می باشد. کار سوییچ بین فریم های داخلی یا فریم اصلی به کمک این کلاسها ساده شده و کدهای زیر یک desktop به همراه فریم داخلی آن ایجاد می کند.

```
desktop = new JDesktopPane();
createFrame(); //Create first window
setContentPane(desktop);

...
//Make dragging a little faster but perhaps uglier.
desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);

...
protected void createFrame() {
    MyInternalFrame frame = new MyInternalFrame();
    frame.setVisible(true); //necessary as of 1.3
    desktop.add(frame);
    try {
        frame.setSelected(true);
    } catch (java.beans.PropertyVetoException e) {}
}

...//In the constructor of MyInternalFrame, a JInternalFrame subclass:
static int openFrameCount = 0;
static final int xOffset = 30, yOffset = 30;

public MyInternalFrame() {
    super("Document #" + (++openFrameCount),
        true, //resizable
        true, //closable
        true, //maximizable
        true); //iconifiable
    //...Create the GUI and put it in the window...
    //...Then set the window size or call pack...

    ...
    //Set the window's location.
    setLocation(xOffset*openFrameCount, yOffset*openFrameCount);
}
```

فریم های داخلی ظرفهای سطح بالا نیستند و این تمایز بین آنها و فریم های معمولی است. برای مثال شما یک فریم داخلی را به یک ظرف مثل JDesktopPane اضافه می کنید و آنها نمی توانند رخداد های مربوط به یک پنجره خارجی را ایجاد کنند. بدلیل این که فریم های داخلی با کدهای مستقل از زیرساخت (platform) تکمیل شده اند

قابلیتهای دارند که در فریمهای معمولی این قابلیتها وجود ندارند. يك از این قابلیتها کنترل بیشتر روی حالت و توانای فریم است، همانند دادن شمایل یا icon و بزرگ کردن (maximizing) به صورت برنامه نویسی شده.

قواعد استفاده از Internal Frames
در زیر قواعد استفاده از فریم های داخلی تك تك بیان شده.

شما باید اندازه فریم داخلی را تنظیم کنید: اگر اندازه فریم داخلی را مشخص نکنید اندازه آن مقدار صفر خواهد گرفت و هرگز قابل دید نخواهد بود. برای تنظیم اندازه فریم داخلی از یکی سه متد `setSize()`, `pack()`, `setBounds()` می توانید استفاده کنید.

محل فریم داخلی باید مشخص گردد: اگر این کار را انجام ندهید فریم داخلی در سمت چپ بالاترین نقطه ظرف یعنی 0,0 قرار خواهد گرفت. به کمک متد `setLocation()` یا `setBounds()` می توان محل فریم در داخل ظرف را تعیین کرد.

برای اضافه کردن يك جزء یا کامپوننت گرافیکی به يك فریم داخلی آنها را باید به نگه دارنده محتوای فریم های داخلی اضافه کنید: شبیه به اضافه کردن به يك `JFrame`.

جعبه های گفتگو برای فریم داخلی به کمک دو متد `JOptionPane()` و `JInternalFrame()` تکمیل می شوند: استفاده از `JDialog` ناممکن است.

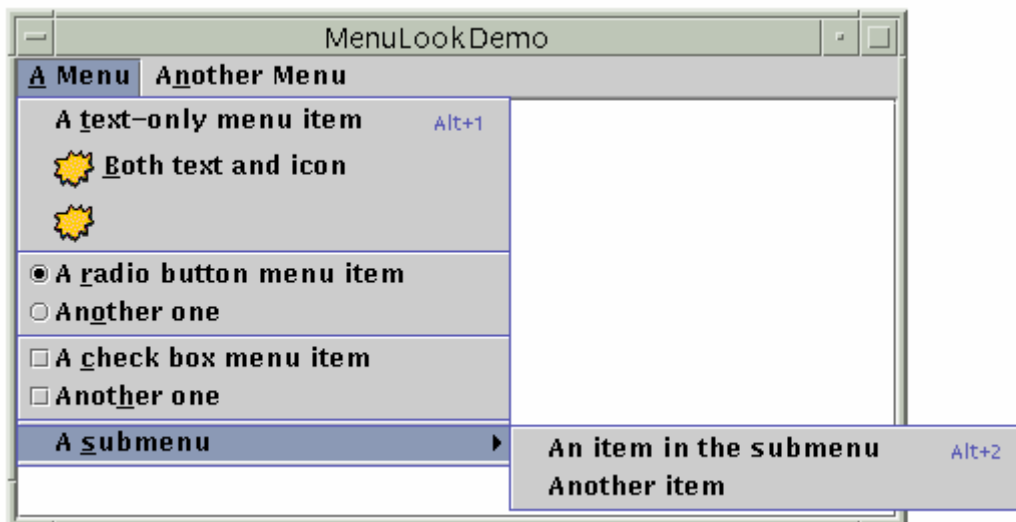
يك فریم داخلی به يك ظرف اضافه می شود: اگر فریم های داخلی را به داخل يك ظرف مثل `JDesktop` اضافه نکنید آنها به نمایش در نمی آیند. بمنظور ظاهر شدن باید متد مخصوص این کار را فراخوانی کنید: شما باید یکی از دو متد `setVisible(true)` یا `show()` را روی فریم داخلی فراخوانی کنید.

زمانی که در داخل يك desktop فریم های داخلی زیادی موجود است به نظر می رسد که انتقال بین هر کدام از آنها ممکن است که کند باشد. انتقال به شکل Outline يك از راه ها برای اجتناب از این امر است. تا زمانی که که وضعیت جدید در کشیدن (dragging) دکمه موس به وجود نیامده فریم داخلی دوباره سازی نمی شود و به محض ثابت شدن دکمه موس محتویات فریم دوباره سازی می شود. اگر از این نوع انتقال استفاده نکنیم تا هر زمان که دکمه موس حرکت می کند کار دوباره سازی انجام می شود و این کار فریمهای داخلی را بسیار کند می کند. تنظیم مدل کشیدن به کمک متد `setDragMode` است و در زیر يك نمونه نشان داده شده.

```
desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
```

منوها

با استفاده از منو کاربر می تواند گزینه های زیادی را برای انتخاب داشته و فضای بسیار کمتری نسبت به سایر روشها اشغال می کند. يك منو معمولاً در يك نگه دارنده منو یا menu bar ظاهر می شود. يك menu bar شامل يك یا چند منو بوده و دارای مکان مستقل از زیرساخت در بالای يك پنجره است، این به معنی است که اگر در يك سیستمی مکان منو در جای مشخص بطور مثال سمت راست پنجره در برنامه شما مکان منو همان مکانی است که شما تعیین کرده اید. در داخل هر منو يك سری قلم (item) از گزینه ها وجود دارد و هر کدام از آنها می توانند يك میانبر برای دسترسی داشته باشند. علاوه بر میانبر ارقام يك منو می توانند شامل checkbox، radio button، icon، و زیر منو باشند. شکل زیر يك نمونه از این منوها است.



بطور مثال در کد زیر نگره دارنده منو شامل يك منو به نام Document دارد. اقلام در این منو new و quit هستند.

```
JMenuBar menuBar = new JMenuBar();
    JMenu menu = new JMenu("Document");
    menuBar.add(menu);
    JMenuItem menuItem = new JMenuItem("New");
    menuItem.setActionCommand("new");
    menu.add(menuItem);
    menuItem = new JMenuItem("Quit");
    menu.add(menuItem);
```

برای تعریف میانبر به منوی مورد نظر باید از متد `setMnemonic()` استفاده کرد. بطور مثال `setMnemonic(KeyEvent.VK_N)` کلید Alt-n را بعنوان میانبر تعریف می کند.

حال کد کامل برنامه را در ادامه ملاحظه می کنید.

```
import javax.swing.JInternalFrame;
import javax.swing.JDesktopPane;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JFrame;
import javax.swing.KeyStroke;
import java.awt.event.*;
import java.awt.*;

public class InternalFrameDemo extends JFrame
    implements ActionListener {
    JDesktopPane desktop;

    public InternalFrameDemo() {
        super("InternalFrameDemo");
```

```

//Make the big window be indented 50 pixels from each edge of the screen.
    int inset = 50;
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setBounds(inset, inset, screenSize.width - inset*2,
              screenSize.height - inset*2);

    //Set up the GUI.
    desktop = new JDesktopPane(); //a specialized layered pane
    createFrame(); //create first "window"
    setContentPane(desktop);
    setJMenuBar(createMenuBar());
    //Make dragging a little faster but perhaps uglier.
    desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
}

protected JMenuBar createMenuBar() {
    JMenuBar menuBar = new JMenuBar();
    //Set up the lone menu.
    JMenu menu = new JMenu("Document");
    menu.setMnemonic(KeyEvent.VK_D);
    menuBar.add(menu);
    //Set up the first menu item.
    JMenuItem menuItem = new JMenuItem("New");
    menuItem.setMnemonic(KeyEvent.VK_N);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
        KeyEvent.VK_N, ActionEvent.ALT_MASK));
    menuItem.setActionCommand("new");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    //Set up the second menu item.
    menuItem = new JMenuItem("Quit");
    menuItem.setMnemonic(KeyEvent.VK_Q);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
        KeyEvent.VK_Q, ActionEvent.ALT_MASK));
    menuItem.setActionCommand("quit");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    return menuBar;
}

//React to menu selections.
public void actionPerformed(ActionEvent e) {
    if ("new".equals(e.getActionCommand())) { //new
        createFrame();
    } else { //quit
        quit();
    }
}

```

```

    }

    //Create a new internal frame.
    protected void createFrame() {
        MyInternalFrame frame = new MyInternalFrame();
        frame.setVisible(true); //necessary as of 1.3
        desktop.add(frame);
        try {
            frame.setSelected(true);
        } catch (java.beans.PropertyVetoException e) {}
    }

    //Quit the application.
    protected void quit() {
        System.exit(0);
    }

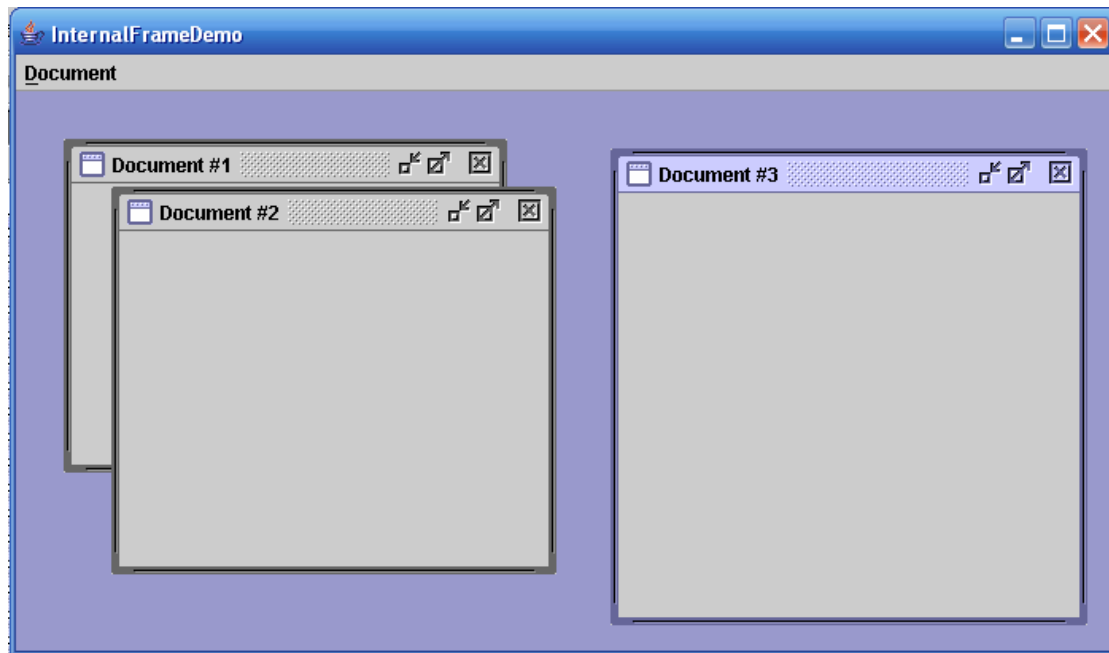
    private static void createAndShowGUI() {
        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);
        //Create and set up the window.
        InternalFrameDemo frame = new InternalFrameDemo();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Display the window.
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}

```

پس از اجرا برنامه به شکل زیر خواهد بود.



استفاده از برگه ها (Tab)

برای ایجاد برگه ها از کلاس JTabbedPane استفاده می کنیم و با استفاده از برگه ها می توان چندین جز (components) گرافیکی را در داخل یک فضای مشترک قرار داد. در داخل هر برگه اجزاء گرافیک گذاشته می شود و برای ایجاد هر برگه از متد addTab() استفاده می شود.

در این مثال چهار برگه به همراه tool tip و یک میانبر برای هر کدام ایجاد کرده ایم. در هنگام نمایش برگه ای که رنگ عنوان آن با سایر برگه ها تمایز دارد به عنوان برگه کاری محسوب شده و برای تغییر و رفتن به دیگر برگه ها از متد setTabPlacement() استفاده می کنیم. در مثال توجه داشته باشید که نوشتن کدهای دستگیر رخداد لازم نمی باشد و شی JTabbedPane به طور خودکار رخداد های صفحه کلید و موس را کنترل می کند.

متد addTab() هر کدام از برگه های tabbedPane را با آرگومانهای عنوان، icon، نام و tool tip ایجاد می کند. البته هر دو آرگومان icon و متن می توانند مقدار null داشته باشند. روش دیگر برای ایجاد برگه استفاده از متد insertTab() است. متد setMnemonicAt() میانبرها را برای هر کدام از برگه ها تعریف کرده و میانبر همه برگه ها از Alt به همراه شماره برگه استفاده می کنند. توجه داشته باشید اندازه همه برگه ها یکسان است و برابر با بزرگترین اندازه برگه در میان برگه ها است. کد کامل این مثال در زیر نمایش داده شده.

```
/* TabbedPaneDemo.java is a 1.4 example that requires one additional file:
 * images/middle.gif */
import javax.swing.JTabbedPane;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JPanel;
```

```

import javax.swing.JFrame;
import javax.swing.JComponent;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.KeyEvent;

public class TabbedPaneDemo extends JPanel {
    public TabbedPaneDemo() {
        super(new GridLayout(1, 1));

        JTabbedPane tabbedPane = new JTabbedPane();
        ImageIcon icon = createImageIcon("images/middle.gif");

        JComponent panel1 = makeTextPanel("Panel #1");
        tabbedPane.addTab("Tab 1", icon, panel1,
            "Does nothing");
        tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

        JComponent panel2 = makeTextPanel("Panel #2");
        tabbedPane.addTab("Tab 2", icon, panel2,
            "Does twice as much nothing");
        tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);

        JComponent panel3 = makeTextPanel("Panel #3");
        tabbedPane.addTab("Tab 3", icon, panel3,
            "Still does nothing");
        tabbedPane.setMnemonicAt(2, KeyEvent.VK_3);

        JComponent panel4 = makeTextPanel(
            "Panel #4 (has a preferred size of 410 x 50).");
        panel4.setPreferredSize(new Dimension(410, 50));
        tabbedPane.addTab("Tab 4", icon, panel4,
            "Does nothing at all");
        tabbedPane.setMnemonicAt(3, KeyEvent.VK_4);

        //Add the tabbed pane to this panel.
        add(tabbedPane);

        //Uncomment the following line to use scrolling tabs.
        //tabbedPane.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);
    }
    protected JComponent makeTextPanel(String text) {
        JPanel panel = new JPanel(false);
        JLabel filler = new JLabel(text);
        filler.setHorizontalAlignment(JLabel.CENTER);
        panel.setLayout(new GridLayout(1, 1));
        panel.add(filler);
    }
}

```

```

        return panel;
    }

    /** Returns an ImageIcon, or null if the path was invalid. */
    protected static ImageIcon createImageIcon(String path) {
        java.net.URL imgURL = TabbedPaneDemo.class.getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }
    }

    private static void createAndShowGUI() {
        //Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);

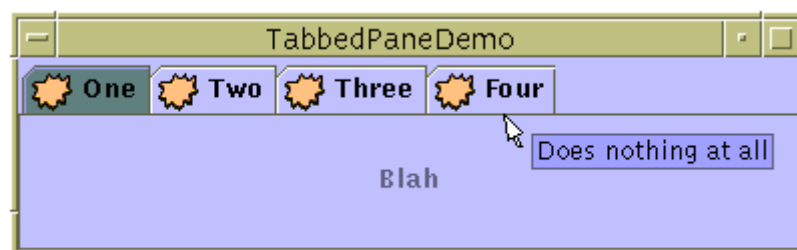
        //Create and set up the window.
        JFrame frame = new JFrame("TabbedPaneDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Create and set up the content pane.
        JComponent newContentPane = new TabbedPaneDemo();
        newContentPane.setOpaque(true); //content panes must be opaque
        frame.getContentPane().add(new TabbedPaneDemo(),
                                   BorderLayout.CENTER);
        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

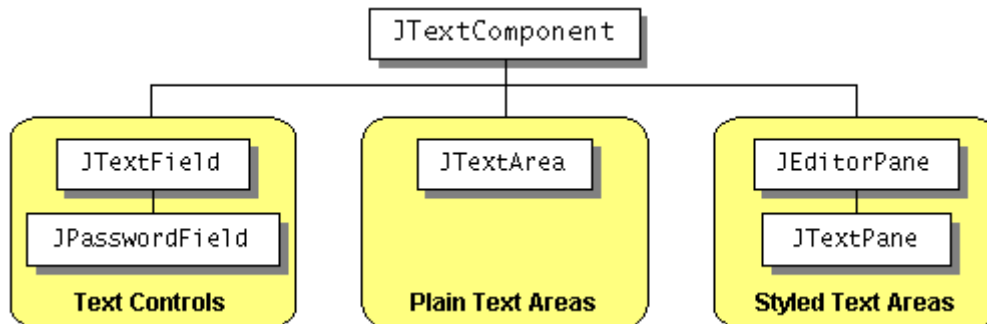
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}

```

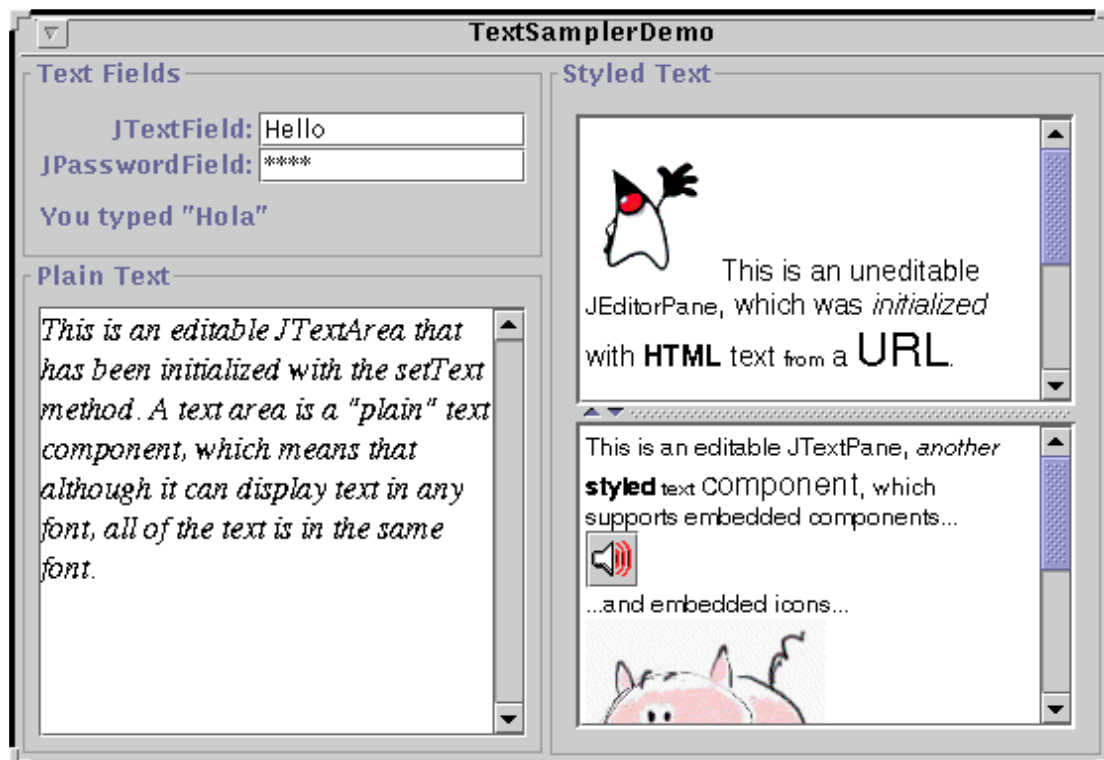
پس از اجرا نتیجه خروجی به شکل زیر خواهد بود.



استفاده از کامپوننت‌های متن
 کامپوننت‌های متنی برای نمایش و ویرایش متن‌ها که در برنامه‌ها لازم هستند ایجاد شده‌اند و تعداد این کامپوننت‌ها شش عدد می‌باشد. همه این کامپوننت‌ها از کلاس `JTextComponent` به ارث برده شده‌اند و می‌توان آنها را به سه دسته عمده تقسیم کرد. این سلسله مراتب در شکل زیر نشان داده شده.



در تصویر زیر نیز یک برنامه که از هر شش نوع کامپوننت متنی استفاده کرده به همراه توضیحات نشان داده شده.



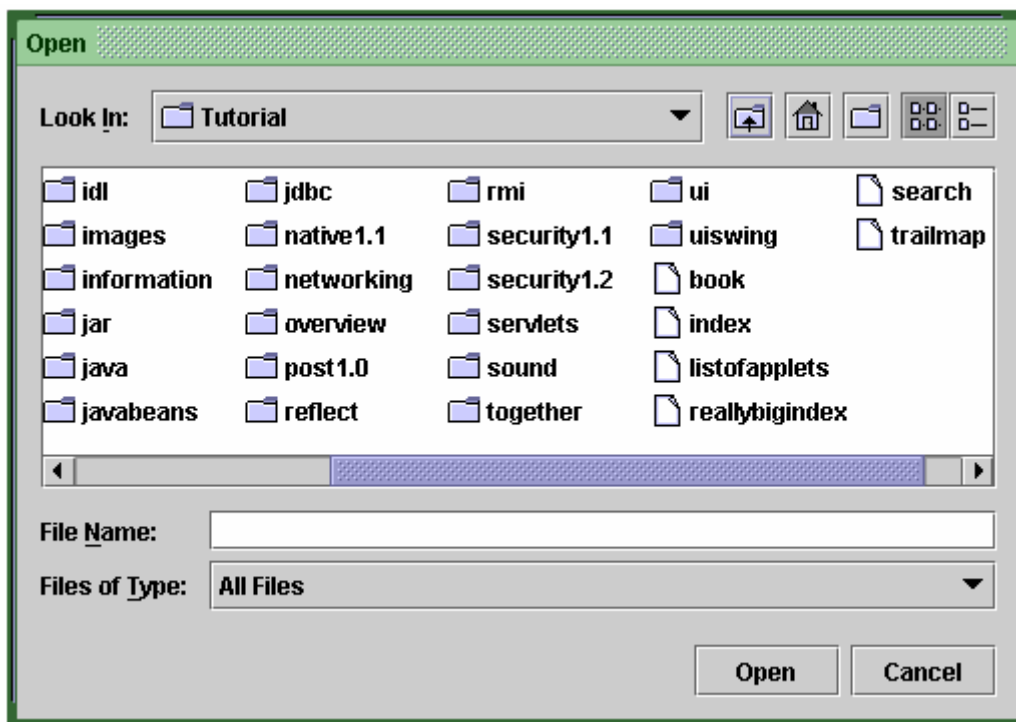
در دسته اول کامپوننت‌ها یعنی `Text Controls` ما فقط می‌توانیم یک خط را نمایش داده و ویرایش کنیم، شبیه به دکمه‌ها. استفاده از آنها متنی کمتری را نمایش داده و زمانی که متن به اندازه طول رسید دستگیر رخداد آن فراخوانی می‌شود. این دسته شامل سه کلاس `JTextField`، `JPasswordField` و `JFormattedTextField` است.

دسته دوم `Plain Text Areas` است که در آن می‌توان متن‌های بیش از چند خط را نمایش داد و ویرایش کرد. اگرچه `text area` متن را با هر `font` نمایش می‌دهد اما توجه داشته باشید که تمام متن باید با آن `font` نشان داده شوند. این دسته فقط

شامل يك كلاس JTextArea است كه به كاربر اين امكان را مي دهد كه متن فرمت نشده نامحدودي را وارد يا تماشا كند. دسته سوم Styled Text Areas هستند. در اين دسته مي توان متن را در بيش از يك font نمايش داد و ويرايش كرد. برخي از اين كامپوننتها اين اجازه را مي دهند كه تصاوير و ساير كامپوننتها را به همراه متن نمايش داد. به دليل قدرت و سازگاري اين كامپوننتها براي برنامه ها با متنهاي طرح دار مثل راهنما ها (help) بسيار مفيد هستند. اين دسته شامل دو كلاس JEditorPane و JTextPane به همراه زير كلاسهايشان است.

مثال انتخاب كردن فايلها

انتخاب كنند فايل يك رابط كاربر مناسب بمنظور ارتباط با سيستم فايلها و انتخاب يك فايل يا فهرست مي باشد. براي نمايش انتخاب كننده فايل شما بايد از API مخصوص اين كار يعني JFileChooser كنيد. روش ديگر استفاده از انتخاب كننده فايل قرار دادن نمونه شئي JFileChooser در داخل يك ظرف است. برنامه شما مسئول انجام كار بر روي فايل انتخاب شده است، همانند باز كردن و ذخيره كردن آن فايل. تعين استاندارد جعبه انتخاب كننده فايل به كمك Look&Feel مشخص مي شود. در جاوا Look&Feel جعبه باز كردن فايل و ذخيره فايل يكسان مي باشد و فقط در عنوان متن ظاهر شونده در بالاي جعبه و دكمه متمايز مي باشند. در زير تصوير يك جعبه باز كردن فايل با Look&Feel استاندارد را مشاهده مي كنيد.



گرفتن يك جعبه باز كننده فايل نياز به دو خط كد دارد، همانند زير:

```
//Create a file chooser
final JFileChooser fc = new JFileChooser();
...
//In response to a button click:
int returnVal = fc.showOpenDialog(aComponent);
```

آرگومان ورودی به متد `showOpenDialog` یک کامپوننت گرافیکی والد را برای جعبه باز کردن فایل مشخص می کند. یک کامپوننت والد فریمی که جعبه باز کردن فایل در داخل آن می باشد و وضعیت قرار گرفتن جعبه را مشخص می کند. بطور استاندارد جعبه در روی کامپوننت والد می باشد. زمانی که پنجره فریم والد `Minimize` باشد، این جعبه نیز نشان داده نخواهد شد و به آن وابسته خواهد بود. بطور پیش فرز یک انتخاب کننده فایل تمام فایلها در فهرست جاری کاربر (`home directory`) را نشان می دهد. شما می توانید فهرست اولیه انتخاب کننده فایل را را تغییر دهید. برای این کار از متد `setCurrentDirectory()` استفاده می کنیم. زمانی که دکمه `Open` از انتخاب کننده فایل فشرده می شود متد `actionPerformed()` از شنود کننده اعمال فراخوانی می شود، همانند زیر :

```
public void actionPerformed(ActionEvent e) {
    //Handle open button action.
    if (e.getSource() == openButton) {
        int returnVal = fc.showOpenDialog(FileChooserDemo.this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            //This is where a real application would open the file.
            log.append("Opening: " + file.getName() + "." + newline);
        } else {
            log.append("Open command cancelled by user." + newline);
        }
    }
    ...
}
```

متدهای `showXxxDialog()` یک مقدار عددی صحیح را که به فایل انتخاب شده کاربر اشاره دارد بر می گردانند. این عدد کافی است برای چک کردن اینکه آیا مقدار برگردانده شده با مقدار `APPROVE_OPTION` (تائید با زدن دکمه `Open`) برابر است یا نه و اگر مقدار دیگری باشد (زدن `Cancel` یا دکمه دیگر) کاری انجام نخواهد شد. بمنظور گرفتن فایل انتخاب شده (یا فهرست انتخاب شده در انتخاب کردن فهرست با همان کلاس) از متد `getSelectedFile()` از `JFileChooser` استفاده می کنیم. خروجی این متد یک شیء نمونه از کلاس `File` است. این مثال نام یک فایل را گرفته و در پیغام `log` آن را نشان می دهد. به کمک متدهای `getPath()`، `isDirectory()` و `exists()` از شیء `File` اطلاعاتی درباره فایل گرفته و همچنین سایر متدها مثل `delete()` و `rename()` برای تغییر در فایل استفاده کنید. بمنظور خوانده و نوشتن از فایل از کلاسهای `FileReader` و `FileWriter` استفاده کنید. یک مثال کوچک در این باره در زیر به نمایش درآمده.

```
import java.io.*;
```

```
public class Copy {
    public static void main(String[] args) throws IOException {
        File inputFile = new File("farrago.txt");
        File outputFile = new File("outagain.txt");

        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
```

```

int c;

while ((c = in.read()) != -1)
    out.write(c);

in.close();
out.close();
}
}

```

این مثال محتوای فایل farrago.txt را به داخل فایل outagain.txt کپی می‌کند. در زیر کد کامل برنامه را مشاهده می‌کنید.

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;

public class FileChooserDemo extends JPanel
    implements ActionListener {
    static private final String newline = "\n";
    JButton openButton, saveButton;
    JTextArea log;
    JFileChooser fc;

    public FileChooserDemo() {
        super(new BorderLayout());
        //Create the log first, because the action listeners need to refer to it.
        log = new JTextArea(5,20);
        log.setMargin(new Insets(5,5,5,5));
        log.setEditable(false);
        JScrollPane logScrollPane = new JScrollPane(log);

        //Create a file chooser
        fc = new JFileChooser();
        //Create the open button. We use the image from the JLF
        //Graphics Repository (but we extracted it from the jar).
        openButton = new JButton("Open a File...",
                                createImageIcon("images/Open16.gif"));
        openButton.addActionListener(this);

        //Create the save button. We use the image from the JLF
        //Graphics Repository (but we extracted it from the jar).
        saveButton = new JButton("Save a File...",
                                createImageIcon("images/Save16.gif"));
        saveButton.addActionListener(this);

        //For layout purposes, put the buttons in a separate panel
    }
}

```

```

JPanel buttonPanel = new JPanel(); //use FlowLayout
buttonPanel.add(openButton);
buttonPanel.add(saveButton);

//Add the buttons and the log to this panel.
add(buttonPanel, BorderLayout.PAGE_START);
add(logScrollPane, BorderLayout.CENTER);
}

public void actionPerformed(ActionEvent e) {

    //Handle open button action.
    if (e.getSource() == openButton) {
        int returnVal = fc.showOpenDialog(FileChooserDemo.this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            //This is where a real application would open the file.
            log.append("Opening: " + file.getName() + "." + newline);
        } else {
            log.append("Open command cancelled by user." + newline);
        }
        log.setCaretPosition(log.getDocument().getLength());

        //Handle save button action.
    } else if (e.getSource() == saveButton) {
        int returnVal = fc.showSaveDialog(FileChooserDemo.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            //This is where a real application would save the file.
            log.append("Saving: " + file.getName() + "." + newline);
        } else {
            log.append("Save command cancelled by user." + newline);
        }
        log.setCaretPosition(log.getDocument().getLength());
    }
}

/** Returns an ImageIcon, or null if the path was invalid. */
protected static ImageIcon createImageIcon(String path) {
    java.net.URL imgURL = FileChooserDemo.class.getResource(path);
    if (imgURL != null) {
        return new ImageIcon(imgURL);
    } else {
        System.err.println("Couldn't find file: " + path);
        return null;
    }
}

```

```

private static void createAndShowGUI() {
    //Make sure we have nice window decorations.
    JFrame.setDefaultLookAndFeelDecorated(true);
    JDialog.setDefaultLookAndFeelDecorated(true);

    //Create and set up the window.
    JFrame frame = new JFrame("FileChooserDemo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Create and set up the content pane.
    JComponent newContentPane = new FileChooserDemo();
    newContentPane.setOpaque(true); //content panes must be opaque
    frame.setContentPane(newContentPane);
    frame.pack();
    frame.setVisible(true);
}

public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI();
        }
    });
}
}

```

بطور پیش فرز انتخاب کننده فایل فقط فایلها را انتخاب می کند. شما به کمک متد `setFileSelectionMode()` این را تغییر داده و به انتخاب کننده هر دو فایل و فهرست یا فقط فهرست تبدیل کنید. نمونه این کار را در زیر ملاحظه می کنید.

```

fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

```

همان طور که در کد برنامه ملاحظه می کنید برای ذخیره فایل از همان کلاس `JFileChooser` بمنظور ایجاد جعبه ذخیره استاندارد استفاده می شود. استفاده از یک انتخاب کننده فایل یعنی `fc` برای نشان دادن باز کردن و ذخیره فایل همان طور که در زیر ملاحظه می کنید دو مزیت دارد. مزیت اول اینکه فهرست جاری باز کننده فایل و ذخیره کننده فایل یکی می باشند و دوم اینکه شما برای هر دو باز کننده و ذخیره کننده فایل یک انتخاب کننده فایل را تنظیم دلخواه کرده اید. پس از اجرای برنامه فرم خروجی به شکل زیر خواهد بود.



ایجاد میله ابزار
 بمنظور ایجاد يك میله ابزار از کلاس JToolBar استفاده می کنیم. این کلاس يك ظرف برای نگهداری کامپوننت‌های گروهی مثل دکمه ها و icon ها در داخل يك سطر یا ستون است. همانند منوها میله ابزار يك راه برای دستیابی راحت‌تر به گزینه ها است. بطور پیش‌فرض کاربر می تواند يك میله ابزار را به مکان دیگری با کشیدن (drag) در فریم انتقال دهد. همچنین این قابلیت وجود دارد که يك میله ابزار خارج از ظرف خود مثل يك فریم قرار بگیرد. کدهای زیر يك میله ابزار را ایجاد کرده و آن را به ظرف اضافه می کند.

```
public class ToolBarDemo extends JPanel
    implements ActionListener {
    ...
    public ToolBarDemo() {
        super(new BorderLayout());
        ...
        JToolBar toolBar = new JToolBar("Still draggable");
        addButtons(toolBar);
        ...
        setPreferredSize(new Dimension(450, 130));
        add(toolBar, BorderLayout.PAGE_START);
        add(scrollPane, BorderLayout.CENTER);
    }
    ...
}
```

وضعیت مکانی میله ابزار به‌مراه میله لغزان هر دو در پانل به کمک طرح بند لبه (border layout) مشخص شده. مکان میله ابزار در شروع صفحه (PAGE_START) و مکان میله لغزنده در مرکز (CENTER) می باشد. به دلیل اینکه میله لغزان در مرکز قرار دارد و کامپوننت گرافیکی دیگری به غیر از میله ابزار در ظرف نمی باشد، میله ابزار می تواند به سایر لبه های ظرف کشیده شود مثلاً در سمت راست فریم قرار گیرد. همچنین می توان آنرا به خارج از ظرف کشید بطوری که در يك پنجره جدا نمایش داده می شود و عنوان آن پنجره "Still draggable" می باشد.

ایجاد دکمه های میله ابزار
 دکمه ها در داخل میله ابزار از نوع JButton بوده و در داخل آنها از تصویر استفاده می شود. هر تصویر در ابعاد 16 در 16 و 24 در 24 است. این به این دلیل است که Look&Feel پیش‌فرض از آن استفاده می کند. در زیر نحوه اضافه کردن يك دکمه به يك میله ابزار نشان داده شده.

```
//third button
button = new JButton(new ImageIcon("images/right.gif"));
button.setToolTipText("This is the right button");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        displayResult("Action for third button");
    }
});
toolBar.add(button);
```

کد کامل برنامه را در زیر نشان داده شده.

```
import javax.swing.JToolBar;
import javax.swing.JButton;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import java.awt.*;
import java.awt.event.*;

public class ToolBarDemo extends JFrame {
    protected JTextArea textArea;
    protected String newline = "\n";

    public ToolBarDemo() {
        //Do frame stuff.
        super("ToolBarDemo");
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        //Create th toolbar.
        JToolBar toolBar = new JToolBar();
        addButtons(toolBar);

        //Create the text area used for output.
        textArea = new JTextArea(5, 30);
        JScrollPane scrollPane = new JScrollPane(textArea);

        //Lay out the content pane.
        JPanel contentPane = new JPanel();
        contentPane.setLayout(new BorderLayout());
        contentPane.setPreferredSize(new Dimension(400, 100));
```

```

        contentPane.add(toolBar, BorderLayout.NORTH);
        contentPane.add(scrollPane, BorderLayout.CENTER);
        setContentPane(contentPane);
    }

    protected void addButtons(JToolBar toolBar) {
        JButton button = null;

        //first button
        button = new JButton(new ImageIcon("images/left.gif"));
        button.setToolTipText("This is the left button");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                displayResult("Action for first button");
            }
        });
        toolBar.add(button);

        //second button
        button = new JButton(new ImageIcon("images/middle.gif"));
        button.setToolTipText("This is the middle button");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                displayResult("Action for second button");
            }
        });
        toolBar.add(button);

        //third button
        button = new JButton(new ImageIcon("images/right.gif"));
        button.setToolTipText("This is the right button");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                displayResult("Action for third button");
            }
        });
        toolBar.add(button);
    }

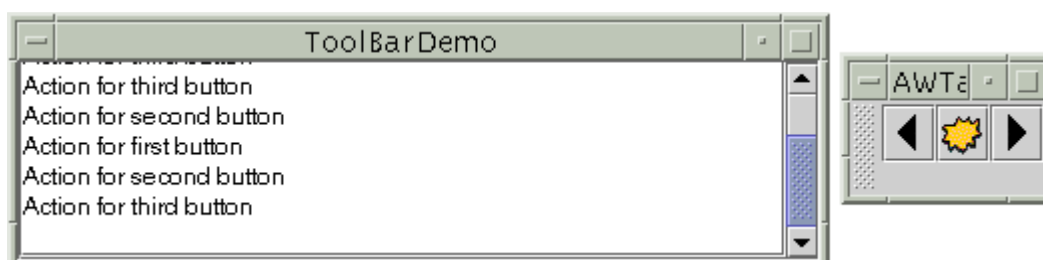
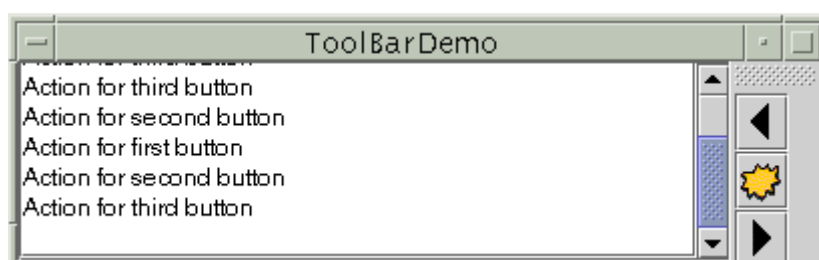
    protected void displayResult(String actionDescription) {
        textArea.append(actionDescription + newline);
    }

    public static void main(String[] args) {
        ToolBarDemo frame = new ToolBarDemo();
        frame.pack();
        frame.setVisible(true);
    }

```

}

پس از اجرا میله ابزار به شکل‌های زیر خواهد بود. در اولین شکل میله ابزار به سمت راست کشیده شده و در دومین شکل میله ابزار به بیرون از ظرف خود کشیده شده.



فصل چهارم

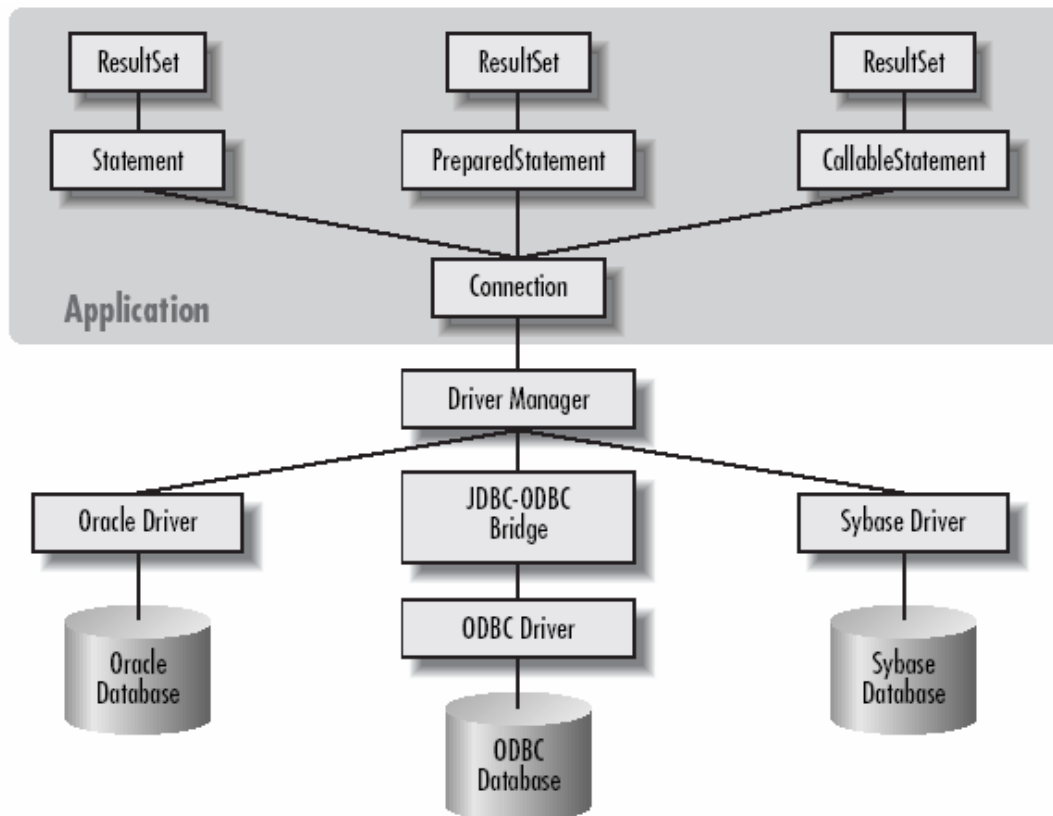
JDBC(Java Data Base Connectivity)

پایگاه داده رابطه ای

یک جدول در پایگاه داده رابطه ای شامل اطلاعاتی در مورد یک موجودیت که که کمک یک کلید با مقدار یکتا به سطرهای آن جدول دسترسی پیدا میکنیم . اطلاعات داخل جداول میتوانند به کمک دستورات SQL حذف ، اضافه ، و به هنگام سازی شوند .
JDBC API که برای اولین بار در JDK 1.1 برای برقراری ارتباط به پایگاه داده و اجرایی دستورات SQL استاندارد ارائه شد

JDBC API

JDBC یک سری API مه به شما اجازه اجرای دستورات SQL و گرفتن نتایج از آنها را میدهد. خود این API ها مجموعه ای از رابطها و کلاسهای طراحی شده برای اجرای اعمال روی هر گونه از پایگاه داده رابطهای موجود است . شکل زیر نحوه تعامل JDBC با بانک اطلاعاتی ODBC Connector , Sybase , oracle را نشان میدهد.



JDBC Drivers

JDBC API در پکیج java.sql و شامل تعدادی کلاسهای به هم مرتبط است. بیشتر این کلاسها بصورت رابط (interface) هستند و تکمیلی آنها ارائه نشده . هرکدام از بانکهای اطلاعاتی به کمک یک JDBC Driver ویژه که در رابط java.sql.Driver میباشد مورد دستیابی قرار می گیرند . این درایورها برای اکثر بانکهای اطلاعاتی معمول به صورت رایگان موجود هستند . شرکت سان JDBC-ODBC به عنوان پل در JDK برای اتصال به منابع داده ODBC استاندارد مثل Microsoft Access ارائه داده.

برقراری ارتباط

اولین قدم استفاده از ODBC Driver برای گرفتن اتصال به پایگاه داده است و لود کردن کلاس درایور به ماشین مجازی که برنامه در آن اجرا میشود. این لود کردن درایور را بعداً برای ما همگامی که می خواهیم متصل شویم قابل دسترس میکند . یک روش راحت برای لود کردن کلاس درایور استفاده از متد Class.forName() است. برای مثال :

```
Class.forName("sun.jdbc.odbc.JdbcodbcDriver");
```

وقتی که درایور به حافظه لود میشود خود را با کلاس java.sql.DriverManager رجیستر می کند.

گام بعدی سوال کردن از کلاس DriverManager برای باز کردن اتصال به بانک اطلاعاتی است . اتصال به کمک تابع getConnection() از DriverManager که ورودی آن یک رشته به فرم URL برای اطلاعات لازم بانک اطلاعاتی است .

متد getConnection() یک کلاس از نوع java.sql.Connection برمی گرداند
 Connection con=

```
DriverManager.getConnection("jdbc:odbc:somedb","user","password");
```

يك JDBC URL براي شناسايي يك بانك اطلاعاتي است و درايور هاي ويژه نياز به اطلاعات جدا در URL خود دارند. معمولاً JDBC URL با `jdbc:subprotocol:subname` شروع مي شوند. براي مثال JDBC Thin Driver براي ORACLE از URL به فرم `jdbc:oracle:thin:@dbhost:port:sid;` براي پل به ODBC شكل كلي URL به فرم `jdbc:odbc:datasourcename;odbcoptions` استفاده مي کند.

در طي زمان اجراي `getConnection()` كلاس `DriverManager` از درايور در مورد قبول يا شناسايي URL سوال مي کند و اگر جواب بله باشد مدير درايور از آن درايور براي ايجاد `Connection` استفاده مي کند.

```
Connection con = null;
```

```
try {
// Load (and therefore register) the JDBC-ODBC Bridge
// Might throw a ClassNotFoundException

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
// Get a connection to the database
// Might throw an SQLException

con = DriverManager.getConnection("jdbc:odbc:somedb", "user", "passwd");
// The rest of the code goes here.
}
catch (ClassNotFoundException e) {

// Handle an error loading the driver
}

catch (SQLException e) {
// Handle an error getting the connection
}
finally {
// Close the Connection to release the database resources immediately.
try {
if (con != null) con.close();
}
catch (SQLException ignored) { }
}
```

اجراي پرس وجو از بانك اطلاعاتي

استفاده از بانك اطلاعاتي نيازمند نوشتن `Queries` و راه ساده براي اجراي `Query` استفاده از كلاس `java.sql.Statement` است. شي `Statement` هيچ وقت به طور مستقيم ايجاد نمي شود بلكه برنامه با فراخواني متد `createStatement()` از شي `Connection` يك شي `Statement` ايجاد مي کند.

```
Statement stmt = con.createStatement();
```

وقتي يك Query وقتي ايجاد شد براي گرفتن داده بايد متد `executeQuery()` از شي `Statement` فراخواني كرد. اين متد دستور را اجرا كرده و در داخل كلاس `jav.sql.ResultSet` مي ريزد

```
ResultSet rs = stmt.executeQuery("SELECT * FROM CUSTOMERS");
```

اينگونه ميتوان فرض كرد كه شي `ResultSet` ردیف یا سطرهای از داده ها در يك زمان از جدول بانك است . براي رفتن به سطر بعدي در ميان سطرها با متد `next()` از `Statement` را اجرا كرده و براي گرفتن فيلدهاي سطر بسته به نوع داده از متد سازگار با آن استفاده ميكنيم. متد هاي `getString()` و `getObject()` بشير از ساير متدها معمولاً استفاده مي شوند.

```
while(rs.next()) {  
  
String event = rs.getString("event");  
  
Object count = (Integer) rs.getObject("count");  
}
```

همان طور كه ميدانيد `ResultSet` به كلاس والد خود يعني `Statement` رابطه دارد و اگر شي `Statement` بسته شود يعني دستور ديگري اجرا شود به طور اتوماتيك شي `ResultSet` بسته ميشود .
مثال زير نشان دهنده اتصال به بانك اطلاعاتي Oracle با `JDBC Driver` و اجراي پرس جوي ساده براي چاپ نام و شماره تلفن باز جدول كارمندان است. فرض كنيد بانك اطلاعاتي داراي جدولي به نام `EMPLOYEES` با دو فيلد `NAME` , `PHONE` است.

```
import java.io.*;  
import java.sql.*;  
  
public class GetEmployees {  
public static void main(String args[]) {  
  
Connection con = null;  
Statement stmt = null;  
ResultSet rs = null;  
try {  
  
// Load (and therefore register) the Oracle Driver  
  
Class.forName("oracle.jdbc.driver.OracleDriver");  
  
// Get a Connection to the database  
con = DriverManager.getConnection(  
"jdbc:oracle:thin:dbhost:1528:ORCL", "user", "passwd");  
  
// Create a Statement object
```

```

stmt = con.createStatement();

// Execute an SQL query, get a ResultSet
rs = stmt.executeQuery("SELECT NAME, PHONE FROM EMPLOYEES");
// Display the result set as a list

while(rs.next()) {
System.out.println( rs.getString("name") + " " + rs.getString("phone"));
}
}
catch(ClassNotFoundException e) {

System.out.println("Couldn't load database driver: " + e.getMessage());
}
catch(SQLException e) {

System.out.println("SQLException caught: " + e.getMessage());
}
finally {
// Always close the database connection.
try {
if (con != null) con.close();
}
catch (SQLException ignored) { }
}
}
}
}

```

Handle یا دستگیر کردن استثناء های SQL

در کد های بالا با دو بلوک Catch دو استثنا ClassNotFoundException و SQLException را دستگیر (handle) میکنیم. اولی از متد Class.forName وقتی که درایور JDBC نتواند لود شود و دومی وقتی که خطای از هرکدام از متدهای JDBC بر بزند.

SQLException همانند دیگر استثناءها با یک خصوصیت اضافی و آن اینکه میتواند زنجیره سازی شود. SQLException یک متد به نام getNextException() دارد و اجازه میدهد استثناءهای اضافی را نیز دستگیر کنیم. برای کامل کردن کد قبلی میتوانید کد زیر را به catch از SQLException اضافه کنید.

```

catch (SQLException e) {

out.println(e.getMessage());

while((e = e.getNextException()) != null) {

System.out.println(e.getMessage());
}
}
}

```

این کد ابتدا استثناء اول را نشان داده سپس بقیه استثناءها اگر موجود باشند را نیز نشان میدهد.
 برای فهمیدن تعداد سطرهای گرفته شده ResultSet میتوان از ResultSetMetaData به شکل زیر استفاده کرد.

```
ResultSetMetaData rsmd = rs.getMetaData();
```

```
int numcols = rsmd.getColumnCount();
```

همچنین میتوان با متد getColumnLabel() از ResultSetMetaData نام هر ستون جدول را گرفت. مثال زیر این عمل را نشان میدهد.

```
for (int i = 1; i <= numcols; i++) {
```

```
System.out.println("column "+i+"is :"+ rsmd.getColumnLabel(i));  
}
```

به منظور دریافت اطلاعات از هر سطر بدون توجه به نام آن از متد getObject() استفاده کرد که بطور خود کار نوع داده ستون را تشخیص داده و معادل با آن را برمیگرداند. با این روش کد نوشته شده شما قابلیت حمل بیشترش خواهد داشت . به کمک getObject() می توان ستون را به هر نوع حتی به غیر از نوع خود دریافت کرد. بطور مثال getObject().toString() .

در جدول زیر متدهای که می توانیم به کمک آنها انواع داده در SQL را دریافت نمایم مشخص میباشد . در ستون اول نام آن داده در SQL استاندارد و در ستون دوم نوع داده که بطور خودکار توسط getObject برگردانده میشود. اگر شما نوع ستونها در جدول SQL خود را میدانید میتوانید بسته به نوع ستون از متدهای سطر سوم استفاده کنید.

SQL Data Type	Java Type Returned by getObject()	Recommended Alternative to getObject()
CHAR	String	String getString()
VARCHAR	String	String getString()
LONGVARCHAR	String	InputStream getAsciiStream() InputStream getUnicodeStream()
NUMERIC	java.math.BigDecimal	java.math.BigDecimal getBigDecimal()
DECIMAL	java.math.BigDecimal	java.math.BigDecimal getBigDecimal()
BIT	Boolean	boolean getBoolean()
TINYINT	Integer	byte getByte()
SMALLINT	Integer	short getShort()
INTEGER	Integer	int getInt()
BIGINT	Long	long getLong()
REAL	Float	float getFloat()
FLOAT	Double	double getDouble()
DOUBLE	Double	double getDouble()
BINARY	byte[]	byte[] getBytes()
VARBINARY	byte[]	byte[] getBytes()
LONGVARBINARY	byte[]	InputStream getBinaryStream()
DATE	java.sql.Date	java.sql.Date getDate()
TIME	java.sql.Time	java.sql.Time getTime()
TIMESTAMP	java.sql.Timestamp	java.sql.Timestamp getTimestamp()

Handle کردن فیلد های خالی (null)

Handle کردن فیلد های null در JDBC با کمی نیرنگ همراه است (زمانی که محتوای یک فیلد در بانک پوچ یا null است متد گیرنده داده مثل getObject() مقدار null بر می گرداند) و متد هیچ شیئی بر نمی گرداند. هیچ راهی برای تشخیص null بودن مقدار یک فیلد قبل از خواندن آن وجود ندارد. متد wasNull() در ResultSet() با مقدارهای true, false که برمی گرداند مشخص میکند که آیا مقدار آخرین ستون خوانده شده null می باشد یا نه. این به این معنی است که شما می بایست ابتدا داده را از ResultSet خوانده و به داخل متغیر ریخته و متد wasNull() را فراخوانی میکنیم. در زیر این عمل نشان داده شده است.

```
int age = rs.getInt("age");
```

```
if(!rs.wasNull())
```

```
System.out.println("age : "+age);
```

روش دیگر کنترل کردن متد getObject() که آیا مقدار آن null می باشد یا نه.

به روز کردن پایگاه داده

در برنامه ها علاوه بر دیدن محتوای جداول نیاز به تغییر محتوای آنها نیز داریم . برای اجرای دستورات UPDATE , INSERT , DELETE می توانید به غیر از ResultSet از متد executeUpdate() Statement استفاده کنید. آن تعدادی سطر تغییر داده شده با Statement را بر می گرداند. بطور مثال :

```
int count =
stmt.executeUpdate("DELETE FROM CUSTOMERS WHERE CUSTOMER_ID =
5");
```

بعد از اجرای دستور SQL کلاسی همانند ResultSet (تعدادی سطر) برگرداننده میشود. به کمک متد execute() Statement میتوانیم متوجه شویم که آیا دستور مورد نظر ما یک یا چند ResultSet ایجاد کرده یا نه و true به معنی ایجاد و false هم به معنی عدم ایجاد می باشد.

```
Boolean b= stmt.execute(sql);
```

متدهای getResultSet() , getUpdateCount() از Statement برای دسترس به نتایج دستور execute() می باشند.

در عمل درج در جداول اولین عمل اجرای متد moveToInsertRow() از resultSet مورد نظر است که امکان درج کردن سطرها را فراهم می کند. سپس با متدهای مخصوص هر فیلد که اول آنها با update شروع شده مثل updateString اقدام به ایجاد سطر برای درج میکنیم . برای درج واقعی دربانک پس از ایجاد سطر درج متد insertRow() را فراخوانی میکنیم .

از TYPE_SCROLL_SENSITIVE برای نشان دادن(اعمال کردن تغییرات سطرها در بانک به resultSet مورد استفاده ما) تغییرات دربانک در برنامه کمک میگیریم.

مثال زیر در مورد عمل درج در بانک اطلاعاتی است.

```
import java.sql.*;
```

```
public class InsertRows {
```

```
public static void main(String args[]) {
```

```
String url = "jdbc:mySubprotocol:myDataSource";
```

```
Connection con;
```

```
Statement stmt;
```

```
try {
```

```
Class.forName("myDriver.ClassName");
```

```
} catch(java.lang.ClassNotFoundException e) {
```

```
System.err.print("ClassNotFoundException: ");
```

```
System.err.println(e.getMessage());
```

```
}
```

```
try {
```

```
con = DriverManager.getConnection(url,
```

```
"myLogin", "myPassword");
```

```

stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);

ResultSet uprs = stmt.executeQuery(
    "SELECT * FROM COFFEES");

uprs.moveToInsertRow();

uprs.updateString("COF_NAME", "Kona");
uprs.updateInt("SUP_ID", 150);
uprs.updateFloat("PRICE", 10.99f);
uprs.updateInt("SALES", 0);
uprs.updateInt("TOTAL", 0);

uprs.insertRow();

uprs.updateString("COF_NAME", "Kona_Decaf");
uprs.updateInt("SUP_ID", 150);
uprs.updateFloat("PRICE", 11.99f);
uprs.updateInt("SALES", 0);
uprs.updateInt("TOTAL", 0);

uprs.insertRow();

uprs.beforeFirst();

System.out.println(
    "Table COFFEES after insertion:");
while (uprs.next()) {
    String name = uprs.getString("COF_NAME");
    int id = uprs.getInt("SUP_ID");
    float price = uprs.getFloat("PRICE");
    int sales = uprs.getInt("SALES");
    int total = uprs.getInt("TOTAL");
    System.out.print(
        name + " " + id + " " + price);
    System.out.println(
        " " + sales + " " + total);
}

uprs.close();
stmt.close();
con.close();

} catch(SQLException ex) {
    System.err.println(

```

```

        "SQLException: " + ex.getMessage());
    }
}

```

در اینجا میتوان کد زیر را نیز هر با در برنامه به کار برد.

```

stmt.executeUpdate("INSERT INTO COFFEES " +
    "VALUES ('Kona', 150, 10.99, 0, 0)");

```

برای عمل حذف باید از متد `deleteRow()` و برای مشخص کردن رکورد خاص (بطور مثال رکورد چهارم) با توجه به ترتیب قرار گیری (معمولاً بر اساس کلید اصلی) با متد `absolute(4)` بر روی آن رفته و عمل حذف را انجام میدهیم.

استفاده از دستورات SQL آماده (Prepared)

شی `PreparedStatement` شبیه قاعده `Statement` بوده و در آن می توانیم دستورات SQL را اجرا کنیم. تفاوت اساسی این است که در `PreparedStatement` دستور توسط بانک برای سرعت بیشتر پیش کامپایل میشود. زمانی که `PreparedStatement` کامپایل میشود آن میتواند با استفاده از پارامترها دلخواه تنظیم شود. این قابلیت برای اجرای دستورات کلی SQL در برنامه مفید است.

متد `PreparedStatement` از شی `Connection` بوده و ؟ برای ارسال پارامتر در بعدها استفاده میکند.

در کد زیر نحوه استفاده از `PreparedStatement` نشان داده شده.

```

PreparedStatement pstmt = con.prepareStatement(
    "INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, TOTAL) VALUES
    (?, ?, ?)");
// Other code
pstmt.clearParameters(); // clear any previous parameter values
pstmt.setInt(1, 2); // set ORDER_ID
pstmt.setInt(2, 4); // set CUSTOMER_ID
pstmt.setDouble(3, 53.43); // set TOTAL
pstmt.executeUpdate(); // execute the stored SQL

```

در نهایت دستور زیر بطور واقعی اجرا میشود.

```

INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, TOTAL) VALUES
(2,4,53.43)

```

متد `clearParameters()` تمام پارامترهای قبلی استفاده شده را حذف میکند. پس از مقدار دهی به فیلدهای رکورد توسط متدهای `setXXX()` به منظور درج واقعی در بانک از متد `executeUpdate()` استفاده میکنیم.

در درج مقادیر متنی در فیلد آن ممکن است بعضی از کاراکترها باشند که بطور تصادفی در دستورات SQL معنای خاصی دارند. به عنوان مثال کاراکتر (') در عبارت "John d'Artagan" به عنوان کاراکتر کنترلی در نظر گرفته میشود. برای حل این موضوع

دو روش وجود دارد. روش اول گذاشتن کد گریز (escape) که با (' ') مشخص می شوند و همچنین در این روش باید سایر کارکترها واحتمالات را در نظر گرفت و کار مشکلی می باشد. روش دوم و بهتر این که با تابع () setString از PreparedStatement به طور اتوماتیک تمام کنترل های ممکن را روی رشته انجام دهیم و نوشتن با این روش مستقل از بانک اطلاعاتی و کدهای مختلف است. مزیت دیگر این روش زمانی که در برنامه های وب از بانک اطلاعاتی استفاده می کنیم از هک شدن جلوگیری می کند.

```
PreparedStatement pstmt = con.prepareStatement(
"INSERT INTO MUSKETEERS (NAME) VALUES (?)");
```

```
pstmt.setString(1, "John d'Artagan");
```

```
pstmt.executeUpdate();
```

استفاده مجدد از اشیاء بانک اطلاعاتی

در برنامه مختلف همانند برنامه های سرویس دهنده وب و سرویس دهنده برنامه بعد از استفاده JDBC Connection و بستن آن برای کاربران دیگر نیاز به باز کردن Connection دیگر داریم و این خود چند ثانیه وقت سرویس دهنده را تلف می کند. برای حل مشکل از اشیاء مخصوص بانک اطلاعاتی دوباره برای کاربران دیگر استفاده می کنیم.

این روش وابسته به نوع برنامه که استفاده می کنیم است و بطور مثال در Servlet ها به کمک متد () init از کلاس Servlet ارتباط به بانک را برای استفاده های متوالی را برقرار می کنیم. متد () init وقتی یک شی Servlet برای اولین بار ایجاد میشود فراخوانی میشود. در EJB (Enterprise JavaBean) اینکار را به کمک متد () ejbLoad از آن Bean انجام می دهیم. مثال مربوط به استفاده مجدد از بانک اطلاعاتی در فصل Servlet موجود است.

تراکنشها (Transactions)

در بانکهای رابطه ای مدرن امروزی یک از قابلیت های مهم پشتیبانی از تراکنشها میباشد و در برنامه های سرویس گرا مثل برنامه مالی بانک به صورت online به بیش از یک دستور select در کار با پایگاه داده نیاز داریم. فرض کنید میخواهیم مقداری پول را از حسابی به حسابی دیگر جابه جا کنیم.

برای اینکار وجود دو حساب چک شده سپس مقدار موجودی حساب اول که برای انتقال آن مقدار بررسی و اگر کمتر باشد عملیات متوقف میشود و در غیر این صورت ادامه عملیات و بقیه ماجرا.

در این جا دو راه حل وجود دارد یک گذاشتن تمام این کنترلها به عهده پایگاه داده رابطه ای که بطور خود کار تمام این مراحل را میتواند کنترل کند و در صورت وجود حالت نامعتبر عملیات Rollback را انجام دهد.

راه حل دوم اینکه از داخل برنامه خود بتوانیم عملیات داخل پایگاه داده را کنترل کنیم. مدیریت تراکنش JDBC در شی Connection بوده و به صورت پیش فرض وقتی که Connection اتصال برقرار میکند آن در مد auto-commit قرار دارد. این به این معنی است که وقتی که دستور SQL اجرا میشود یک تراکنش به بانک اطلاعاتی commit (ارسال معتبر) میشود. بدعی است اگر ارسال commit نباشد تغییرات انجام شده توسط دستور SQL ما نامعتبر بوده و به حالت قبلی خواهند برگشت. برای کنترل ارسال معتبر یا نامعتبر به کمک متد () setAutoCommit(false) بر روی شی Connection

میتوان دستورات را نامعتبر کرد. به منظور چک کردن وضعیت auto-commit متد `getAutoCommit()` را به کار میبریم.

زمانی که تمام دستورات برنامه و SQL شما بطور کامل اجرا شد برای نهایی کردن تراکنش با فراخوانی متد `commit()` تمام رکورد ها در بانک اطلاعاتی بصورت ماندگار ثبت میشوند. حال اگر در برنامه، شما با خطای مواجه شدید به کمک متد `rollback()` میتوانید اطلاعات قبل از تراکنش را برگردانید.

در مثال زیر نحوه استفاده از تراکنش را مشاهده میکنید. در بانک اطلاعاتی جدول INVENTORY (شامل ID محصول و قیمت آن) و جدول SHIPPING (شامل ID محصول ، شماره ترتیب و هزینه حمل shipped) را داریم. با متد `chargeCard()` (در این جا کد آن نوشته نشده) کارت اعتباری مشتری را کنترل میکنیم و در داخل آن اگر خطای به وجود آید یک استثناء صادر می کند و موجب میشود Rollback عملیات Update انجام شده در بانک را به حالت قبل از آن برمی گردانند.

```
import java.io.*;
import java.sql.*;

public class transaction {
    public static void main(String args[]) {

        Connection con = null;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con = DriverManager.getConnection("jdbc:odbc:ordersdb", "user", "passwd");
            // Turn on transactions
            con.setAutoCommit(false);
            Statement stmt = con.createStatement();
            stmt.executeUpdate(
                "UPDATE INVENTORY SET STOCK = (STOCK - 10) WHERE PRODUCTID = 7");
            stmt.executeUpdate(
                "UPDATE SHIPPING SET SHIPPED = (SHIPPED + 10) WHERE PRODUCTID = 7");
            chargeCard(); // method doesn't actually exist...
            con.commit();
            System.out.println("Order successful! Thanks for your business!");
        }
        catch (Exception e) {
            // Any error is grounds for rollback
            try {
                con.rollback();
            }
            catch (SQLException ignored) { }
            System.out.println("Order failed. Please contact technical support.");
        }
        finally {
            // Clean up.
```

```

try {
if (con != null) con.close();
}
catch (SQLException ignored) { }
}
}
}
}

```

سایر امکانات JDBC

در اینجا کمی به روالهای ذخیره شده در بانک اطلاعاتی، نحوه گرفتن داده ها با نوع پیچیده و کار با متنهای طولانی را بیان خواهیم کرد.

روالهای ذخیره شده (Stored Procedures)

اکثر بانکهای اطلاعاتی رابطهای یک زبان برنامه نویسی داخلی دارند همانند PL/SQL در Oracle و این امکان را به توسعه دهندگان بانک اطلاعاتی میدهند تا روالهای مورد نیاز در برنامه خود را در داخل بانک اطلاعاتی بنویسند و آنها را از برنامه فراخوانی کنند.

این روالها برای انجام اعمال واقعی در بانک اطلاعاتی بسیار مناسب میباشند و خود بانکهای اطلاعاتی از آنها بسیار استفاده می کنند. در کد زیر روال نوشته شده در Oracle PL/SQL را مشاهده میکنید.

```

CREATE OR REPLACE PROCEDURE sp_interest
(id IN INTEGER bal IN OUT FLOAT) IS
BEGIN
SELECT balance
INTO bal
FROM accounts
WHERE account_id = id;
bal := bal + bal * 0.03;
UPDATE accounts
SET balance = bal
WHERE account_id = id;
END;

```

این روال با اجرای دستور SQL عمل محاسباتی را بر روی فیلد bal جدول accounts بانک انجام داده و آن را به روز میکند.

نوشتن این روالها آسان بوده و مزایای بسیاری دارند. از جمله :

- روالهای ذخیره شده در بانک اطلاعاتی پیش کامپایل شده و سرعت آنها از dynamic SQL بیشتر می باشد.
- روالهای ذخیره در داخل RDBMS اجرا میشوند و میتوان با درخواستهای مختلف جدا آنها را اجرا کرد بدون ترافیک شبکه.
- روالهای ذخیره در بانک اطلاعاتی یک بار نوشته میشوند و بارها توسط برنامه های کاربردی مختلف به زبانهای مختلف برنامه نویسی استفاده می شوند.

- اگر در جدول بانک تغییری ایجاد شود فقط نیاز به تغییر کد روالهای ذخیره شده داریم و لازم نیست کد برنامه ها تغییر کند.
 هر کدام از بانکهای اطلاعاتی برای دستیابی به روالهای ذخیره شده نحوه خاص خود را دارد. در JDBC يك استاندارد برای دستیابی به روالهای ذخیره شده در کلاس `java.sql.CallableStatement` وجود دارد.
 روش استفاده از روالی که مقداری بر نمیگرداند به فرم `call { procedure_name(?,?) }` و همچنین روالی که يك مقدار خروجی برمی گرداند `call { procedure_name(?,?) = }` می باشد.
 استفاده از `CallableStatement` همانند استفاده از کلاس `PreparedStatement` می باشد.

```
CallableStatement cstmt = con.prepareCall("{call sp_interest(?,?)}");
cstmt.registerOutParameter(2, java.sql.Types.FLOAT);
cstmt.setInt(1, accountID);
cstmt.execute();
System.out.println("New Balance: " + cstmt.getFloat(2));
```

در کد بالا با استفاده از متد `prepareCall()` از شی `Connection` يك `CallableStatement` ایجاد می کند. چون روال مورد نظر ما خروجی دارد با کمک متد `registerOutParameter()` از `CallableStatement` نوع خروجی شناسایی می شود که از نوع `Float` (فیلد شماره ۲ که خروجی است) میباشد. از `setInt()` برای دادن مقدار متغیر `accountID` به پارمتر ورودی (شماره ۱) استفاده می کنیم.
 سرانجام پس از اجرای روال با متد `getFloat()` خروجی را چاپ می کنیم. لازم به ذکر است که استفاده از متدهای `getXxx()` همانند استفاده آنها در `ResultSet` است.

کار با دادهای متنی و باینری

اغلب بانکهای اطلاعاتی انواع دادهها مانند متنها با چندین گیگابایت اندازه و اطلاعات باینری همانند فایلها `multimedia` را پشتیبانی می کنند.
 بانکهای اطلاعاتی مختلف با روشهای مختلفی این کار را انجام میدهند اما با متدهای `JDBC` میتوانیم آنها را در فرمت استاندارد در یافت و ارسال کنیم.
 متد `getAsciiStream()` برای گرفتن داده های متنی و متد `getBinaryStream()` برای گرفتن اشیاء دودویی با اندازه بالا از `ResultSet` قابل استفاده اند و هر دوی این متدها يك `InputStream` بر می گردانند.
 پشتیبانی از داده ها با اندازه بالا یکی از مسائل در `JDBC` است. برای این کار ابتدا مطمئن شوید که درایور شما درست کار میکند و آن را تست کنید. درایورهای `Oracle` `JDBC` برای داده با اندازه بالا در این زمینه مستعد مشکل میباشد.
 در کد زیر نحوه خواندن `ASCII String` نشان داده شده و ما فرض کرده ایم که `Connection` , `Statement` و بقیه قبلاً ایجاد شده اند.

```
try {
    ResultSet rs = stmt.executeQuery(
        "SELECT TITLE, SENDER, MESSAGE FROM MESSAGES WHERE MESSAGE_ID = 9");
    if (rs.next()) {
        System.out.println(rs.getString("title"));
        System.out.println("From: " + rs.getString("sender"));
        BufferedReader msgText = new BufferedReader(
            new InputStreamReader(rs.getAsciiStream("message")));
        while (msgText.ready()) {
            System.out.println(msgText.readLine());
        }
    }
}
```

```

}
}
}
catch (SQLException e) {
// Report it
}

```

در کد بالا فرض کرده ایم که يك جدول حاوي پيغامها(عنوان، فرستنده، پيغام) داریم و در ابتدا عنوان پيغام و فرستنده آن را چاپ کرده و سپس پيغام را با گرفتن InputStream در حلقه while خط به خط چاپ میکنيم.

در کد تا زمانی که از InputStream میخوانيم نباید مقدار ساير ستونها را از ResultSet گرفت و این بسيار مهم است چون هرگونه فراخواني متدهاي (getXXX()) از ResultSet جريان ورودی(InputStream) را خواهد بست. داده باینري با getBinaryStream() از ResultSet گرفته میشود و باید نوع محتوای آن با setContentType("image/gif") مشخص کرد. در این مثال نوع آن فایل تصویر با پسوند GIF است.

```

*****
Connection con;

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
"SELECT IMAGE FROM PICTURES WHERE PID = 7");

if (rs.next()) {
    BufferedInputStream gifData =
    new BufferedInputStream(rs.getBinaryStream("image"));
    byte[] buf = new byte[4 * 1024]; // 4K buffer
    int len;

    while ((len = gifData.read(buf, 0, buf.length)) != -1) {
        out.write(buf, 0, len);
    }
}
}

```

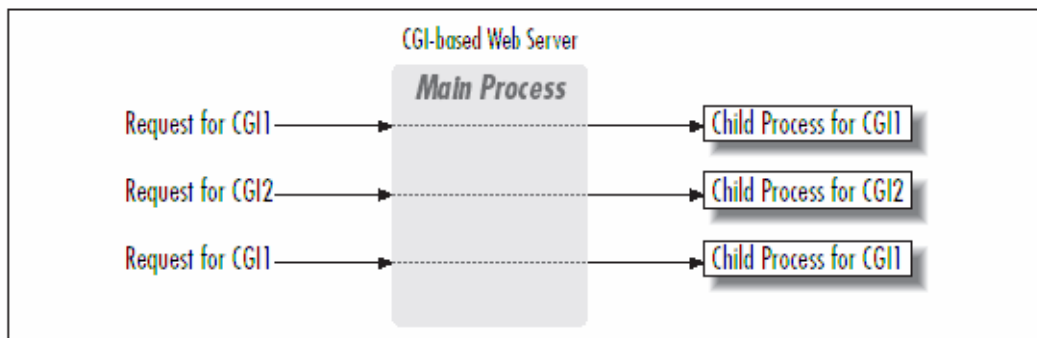

فصل پنجم

Servlet

در بعضی از مواقع در سرویس دهنده وب به علت اجرای صفحات در حالت مفسری ما می توانیم برای افزایش سرعت اجرا از برنامه های کامپایل شده استفاده کنیم
Servlet کامپوننت های کلیدی سمت سرویس دهنده جاوا می باشند که برای دسترسی به آنها از روش مشابه در CGI (Common Gateway Interface) استفاده میشود.

Common Gateway Interface

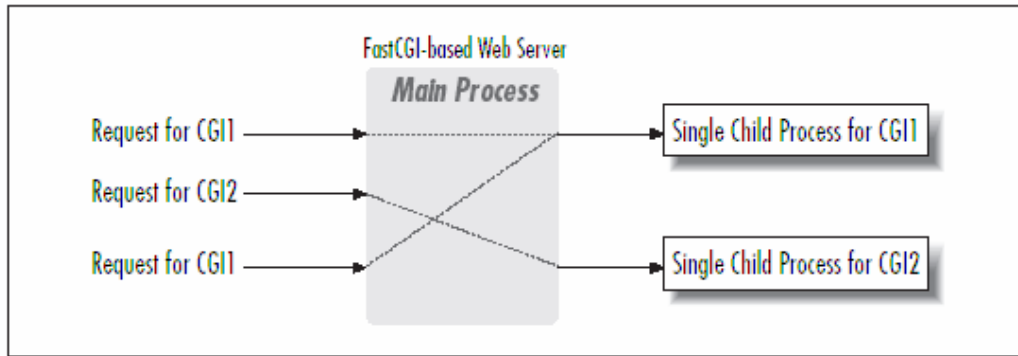
به کمک CGI یک سرویس دهنده وب می تواند Request فرستاده شده از طرف Client را به یک برنامه خارجی ارجاع دهد و خروجی آن برنامه به Client فرستاده شود. مزیت CGI در این است که میتوان می توان برنامه های وب پویا با برنامه های خارج از سرویس دهنده وب را ترکیب کرد و سرعت اجرا به ویژه در سرویس های وبی که خاصیت پردازشی و محاسباتی دارند به طور قابل ملاحظه ای بهبود بخشید.
شکل زیر نحوه ارسال درخواست سرویس گیرنده را نشان میدهد.



برنامه CGI میتواند با اغلب زبانهای برنامه نویسی نوشته شود. زبان Perl یکی از زبانهای رایج در نوشتن CGI می باشد و قابلیت پردازش متن را در خود دارد. همچنین مستقل از سخت افزار میباشد اما نیاز به اجرای مفسر Perl در هر درخواست Client است و مشکل زیاد شدن زمان پردازش و منابع اضافی را دارد.
در CGI برنامه CGI با سرویس دهنده وب تعامل لازم را ندارد چون هر دو در فرآیندهای جدا اجرا میشوند بطور مثال یک CGI script نمی تواند در یک فایل ثبت وقایع (Log) مربوط به سرویس دهنده وب بنویسد

FastCGI

FastCGI همانند CGI بوده با این تفاوت که فرآیند برای هر درخواست سرویس گیرنده را حذف کرده و برای تمام درخواستهای یک فرآیند ایجاد میکند ای عمل در شکل زیر نشان داده شده.



FastCGI نسبت به CGI سریعتر است اما باز هم نیاز به اجرای مفسر Perl برای هر اتصال سرویس گیرنده می باشد

mod_perl

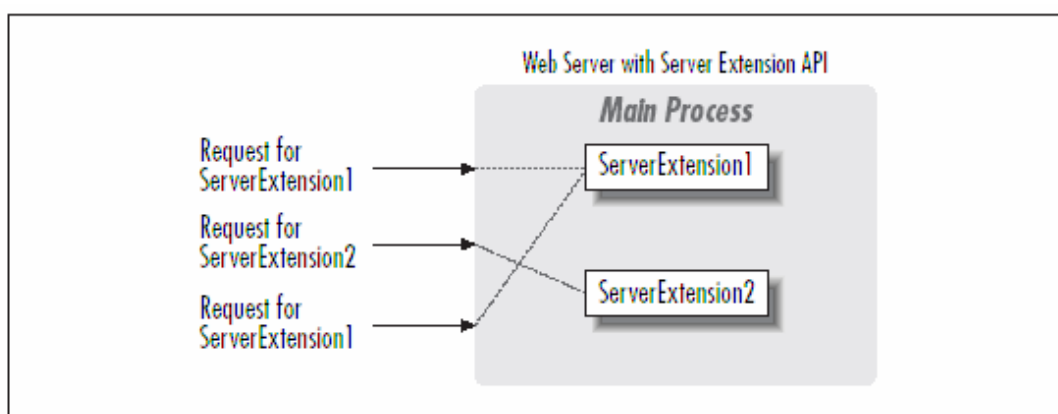
در روش اگر از سرویس دهنده وب Apache استفاده میکنید میتوانید یک کپی از مفسر Perl را در Apache جاسازی کنید و به قابلیت های Perl دسترسی مستقیم داشته باشید. به کمک mod_perl سرور خود CGI را تفسیر کرده و سرعت اجرا بالا تر میرود.

PerlEx

PerlEx از API های محلی سرور وب استفاده میکند و کارایی بهبود می یابد و در سرویس دهنده ویندوز NT یا (Microsoft Internet Information Server) قابل استفاده میباشد.

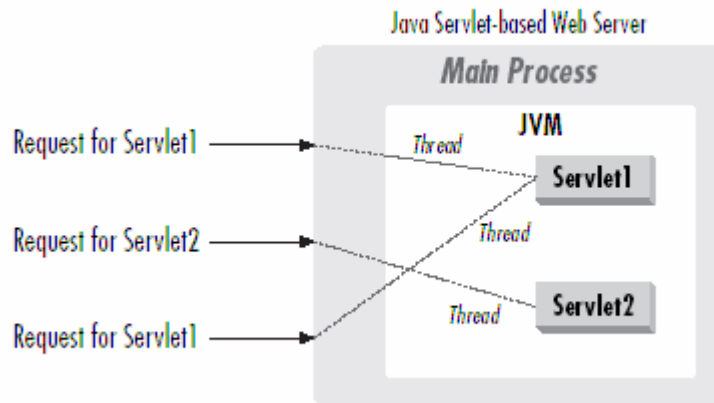
Server Extension APIs

بعضی از شرکتها شروع به ایجاد API های با قابلیت الحاق به سرویس دهنده وب میباشند. برای مثال Netscape یکسری API های داخلی به نام NSAPI که اکنون با نام WAI شناخته میشود. شرکت میکروسافت نیز ISAPI (Internet Service API) برای سرویس دهنده IIS خود ارائه کرد. با کمک این API ها می توان با کدهای C یا C++ ارتباط برقرار کرده و کدهای الحاقی باعث بهبود سرعت اجرا و استفاده مناسب از منابع میشود. در نوع server extension دیگر مثل perl نیاز به اجرای یک فرایند برای هر سرویس گیرنده نیست و یک server extension برای نیاز ما کافی میباشد. همچنین تمام server extension ها در داخل یک فرایند اجرا میشوند. این اعمال در شکل زیر کشیده شده.



Java Servlet

Servlet يك كلاس جاوا بصورت Server Extension است كه مي تواند به طور پويا داخل سرور شود و معمولاً در سرور وب جاوا JSP استفاده مي شود. Servlet همانند Server Extension گفته شده مي باشد با اين تفاوت كه Servlet در داخل JVM بر روي سرور اجرا ميشود (در شكل مشخص شده) Servlet همانند Applet نياز به پشتيباني جاوا در مرورگر ندارد به دليل اينكه در سمت سرور مي باشد.



در Servlet بجاي استفاده از فرآيند براي اجراي آن از نخ يا Thread استفاده ميشود برعكس CGI كه براي هر سرويس گيرنده فرآيند اجرا مي كرد . هر Servlet براي اجراي خود يك نخ در داخل فرآيند سرور اجرا ميشود و در تعامل نزديك با سرويس دهنده است . مزيت ديگر Servlet قابليت حمل آن است و قابل اجرا در تمام سيستم عامل ها است و بعضي بر اين باروند كه Java Servlet بهترين زير ساخت براي توسعه برنامه هاي وب است .

Servlet جاگزين مناسبتي براي CGI است و در انواع سرور ها ميتواند بسط يابد. براي مثال فرض كنيد كه يك سرويس دهنده FTP بر پايه جاوا داريم كه دستورات را با Servlet پشتيباني مي كنند. بمنظور چك كردن فايلها از نظر ويروس و دستورات جديد به راحتی مي توانيم Servlet جديد به سرور اضافه كنيم .

موتور Servlet

يك سرويس دهنده براي اجراي Servlet نياز به موتور يا پردازش گر مخصوص به خود دارد و در سرويس دهنده JSP از آن پشتيباني شده وبطور داخلي از Servlet پشتيباني ميكند . ميتوان براي ساير سرويس دهنده ها نيز از Servlet استفاده كرد فقط نياز به نصب موتور Servlet دارد. براي سرويس دهندهاي وب Apache , Netscape fastTrak , Microsoft IIS , IBM WebSphere موتور Servlet طراحي شده و ميتوان براي استفاده از Servlet آن را نصب كرد.

مزيت هاي Servlet

قابليت حمل:

به دليل اينكه Servlet به زبان جاوا بوده ، خوش تعريف بودن و استفاده از انواع كلاسهاي پايه جاوا قابليت حمل بالايي مي ان سيستم عامل ها دارد . يعني شما ميتوانيد Servlet را يكبار نوشته و در انواع سرويس دهنده ها اجرا كنيد .

قدرت

Servelt میتواند از API های مرکزی جاوا مثل API های networking , URL , multithreading , image manipulation , data compression , database connectivity , object serialization , CORBA , remote method invocation , و سایر امکانات را پشتیبانی میکند . همچنین اگر برنامه شما نیاز به ایجاد برنامه استفاده کننده از فهرست راهنما (directory service) بر پایه وب داشته باشد JNDI API کمک موثری میتواند باشد.

Servelt میتواند با استفاده از کامپوننتهای Enterprise JavaBeans همانند يك Application Server در سرویس دهنده وب استفاده شود.

راندمان و قابلیت تحمل

زمانیکه يك Servelt بازگذاری میشود به طور عام در حافظه سرویس دهنده وب همانند يك شي تنها باقي مانده و در مرحله بعد سرویس دهنده با يك فراخوانی ساده همانند فراخوانی تبع لز آن استفاده میکند . برعکس در CGI باید از فرآیند CGI این عمل را انجام داد که پیچیده تر میباشد .

همچنین Servelt برای فراخوانی های همزمان و یا چند تاي client فقط برای آنها نخ (thread) جدید اجرا کرده منابع کمتر به هدر میرود.

Servelt همانند اشیاء داخل حافظه هستند که حالت خود را مدیریت کرده و می توانند منابع خود را مثل اتصال دائم به بانک اطلاعاتی و غیره نگهداری کنند.

ایمنی

Servelt از مسائل برنامه نویسی امن پشتیبانی در سطوح مختلف پشتیبانی می کند. چون از زبان جاوا strong type safety به ارث می برد . بطور مثال زمانی که اقلب مقادیر در برنامه CGI شامل ارقام عددی شبیه شماره پورت سرویس دهنده است با آنها شبیه مقادیرهای String در ارسال برخورد میشود اما در Servelt همانند نوع واقعی خود برخورد شده .

همچنین اشغال جمع کن جاوا (garbage collector) در هنگام احتیاج نداشتن به Servelt آنها را به طور خودکار از حافظه برداشته تا از ارجاع نامعتبر اشاره گرها و اشغال حافظه جلوگیری شود.

Servelt به کمک مکانیزم java exception handling خطاها را دستگیر یا handle میکند تا از اجرای غیر قانونی کد جلوگیری شود.

یکپارچگی

Servelt ها بطور کامل با سرویس دهنده جاوا یکپارچه هستند و به Servelt امکان همکاری و هماهنگی با سرور را دارد برای مثال Servelt میتواند از امکان ترجمه مسیر فایلها در دایکتوری مجازی برای ثبت وقایع و log ها استفاده کند .

قابلیت انعطاف و توسعه پذیری

Servelt ها قابلیت توسعه بالایی دارند امروزه که از کلاسهای API های بهینه شده HTTP Servelt استفاده میکنیم در آینده می توانیم از همان HTTP Servelt بهینه (optimize) شده استفاده کنیم بدون تغییر در کد . شرکت سان سالانه تمام API های خود بهتر و جدید تر میکند و در JSDK ورژن جدید ارائه میدهد.

Servelt انعطاف بالایی دارد بطور مثال میتوانید Servelt را برای ایجاد صفحه وب استفاده کنید و به يك صفحه ایستا با تگ <SERVELT> آنها را اضافه کنید. این کار در شرکت های استفاده می شود که تعداد Servelt زیادی دارد و به کمک این روش محتوای آنها را فیلتر می کنند در از کد نویسی اضافی و پیچیده تر شدن جلوگیری شود. به این عمل Servelt Chain می گویند.

HTTP Servelt

در این قسمت با نحوه کار HTTP Servelt ، ایجاد صفحه HTML و دسترسی به آن توسط نام آن آشنا می شویم.

میتوان يك HTTP Servelt را در داخل يك صفحه HTML جا سازی کرد بطوری که متدهای آن در سمت سرور اجرا میشوند. برای ایجاد قدرت بیشتر و استفاده از تکنیک فیلتر کردن محتوا این قابلیت وجود دارد که Servelt ها به هم دیگر زنجیر شوند. وقتی که ما Servelt را در داخل صفحه HTML بنویسیم به این تکنولوژی Java Server Page میگویند.

درخواست و پاسخ (Request , Response)

ارتباط در اینترنت به صورت است که سرویس گیرنده با ارسال درخواست به سرویس دهنده وب آماده گی خود را برای دریافت اطلاعات یا سرویس اعلام میدارد سپس سرویس دهنده با توجه به شرایط به سرویس گیرنده پاسخ میدهد.

زمانیکه سرویس گیرنده درخواست را ارسال میکند اولین کار این است که در خواست به کمک متدی در سرور پردازش می شود این درخواست شامل اطلاعات در باره سرویس گیرنده میباشد که به سرور میگوید کدام کار باید انجام شود.

اولین خط درخواست شامل آدرس URL صفحه درخواستی و ورژن پرتکول HTTP است.

GET /intro.html HTTP/1.0

درخواست از متد GET برای سوال درباره صفحه intro.html استفاده میکند و پس از ارسال درخواست سرویس گیرنده اطلاعات اختیاری را ارسال می کند. این اطلاعات اختیاری میتواند شامل نام نرم افزار سرویس گیرنده یا مرورگر آن ، انواع فایل های که توانای نشان دادن آنها را دارد وسایر اطلاعات باشد.

User-Agent: Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)

Accept: image/gif, image/jpeg, text/*, */*

سرویس دهنده پس از گرفتن درخواست يك پاسخ ارسال میکند. اولین خط پاسخ يك خط وضعیت است برای مثال :

HTTP/1.0 200 OK

در خط وضعیت ابتدا ورژن پرتکول HTTP نوشته میشود و بعد از آن عدد 200 نشانه دریافت درخواست به صورت موفق است. کلمه "OK" به این معنی که صفحه درخواستی مورد نظر سرویس گیرنده در سرور موجود است. پس از ارسال خط وضعیت سرویس دهنده اطلاعاتی در مورد نرم افزار و انواع محتوا در سرور به سرویس گیرنده ارسال میکند.

Date: Saturday, 23-May-2003 03:25:12 GMT

Server: JavaWebServer/1.1.1

MIME-version: 1.0

Content-type: text/html

Content-length: 1029

Last-modified: Thursday, 7-May-2003 12:15:35 GMT

در ابتدا زمان سرویس دهنده و نوع نرم افزار سرور مشخص میشود. MIME انواع داده های media را که سرویس دهنده ارسال میکند را مشخص می کند. Content-type معرف شکل فایلی که درخواست شده است. در انتها تاریخ آخرین تغییر در فایل ذکر میشود.

GET, POST

سرویس گیرنده به دو روش می تواند درخواست را ارسال کند روش اول با متد GET که برای گرفتن اطلاعات با حجم کم و با امنیت کمتر مثل متن، چارت و یا نتیجه در خواست از يك بانک اطلاعاتی. روش دوم با POST برای ارسال اطلاعات با حجم یا امنیت بالاتر مثل شماره کارت اعتباری، اطلاعات شخصی و یا اطلاعات که در داخل بانک اطلاعاتی ذخیره می شوند.

اطلاعات در GET به صورت يك دنباله از کارکترها در شکل URL ارسال می شوند و به این نوع URL ها query string میگویند. به همین دلیل می باشد که در GET نمی توانیم اطلاعات با حجم بالا را ارسال کنیم چون حجم URL تا 240 کارکتر میباشد. متد POST از تکنولوژی جدا از GET استفاده میکند و می توانیم اطلاعات در حد مگا بایت را ارسال کنیم. در POST از ارتباط مستقیم با socket از پرتکول HTTP استفاده کرده و محدودیت ارسال نداریم.

سایر متدها

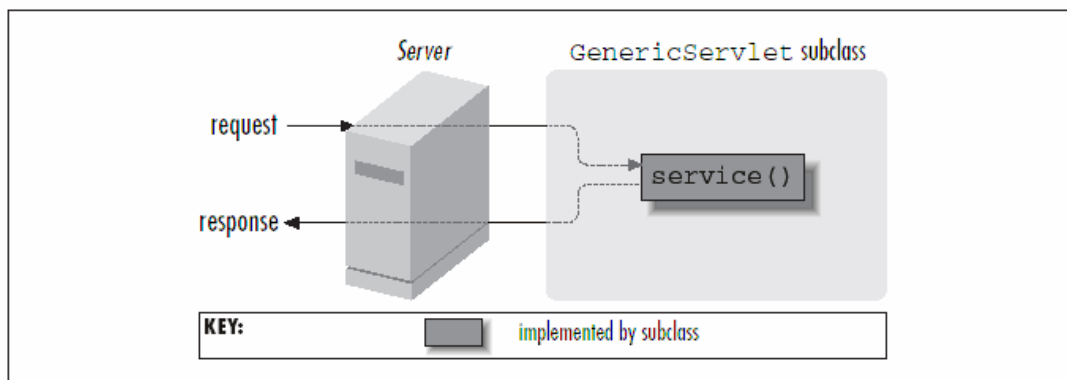
علاو بر POST, GET چند متد دیگر وجود دارند که کمتر استفاده میشوند. متد HEAD که توسط سرویس گیرنده ارسال میشود برای زمانی که فقط می خواهد سرآیند Response را ببیند یا اندازه سند ارسالی را تعیین کند و زمان را تغییر دهد و سایر موارد استفاده کند. متد OPTION برای سوال از سرویس دهنده در مورد قابلیت های اضافی که پشتیبانی می کند و منابعی که در آن قابل دسترسی میباشند. متد TRACE به اشکال گیری debugging درخواست ارسالی در مورد محتوای آن به سرویس گیرنده کمک میکند. متد PUT برای قرار دادن يك سند به طور مستقیم در روی سرویس دهنده است و متد DELETE برعکس متد PUT میباشد.

Servelt API

برای ایجاد HTTP Servlet باید از API های مخصوص این کار که در بسته های `javax.servlet`، `javax.servlet.http` قرار دارند استفاده کنیم. بسته `javax.servlet` شامل کلاسهای برای پشتیبانی از Servlet و مستقل از پرتکول است همچنین برای اضافه کردن قابلیت های ویژه HTTP بسته `javax.servlet.http` را به برنامه `extend` می کنیم. تمامی Servlet ها باید باید رابط `javax.servlet.Servlet` را کلاس خود تکمیل کنند. اکثر Servlet ها آن را با `extend` کردن از بین کلاس `javax.servlet.GenericServlet` یا `javax.servlet.http.HttpServlet` انجام میدهند. برای مستقل از پرتکول بودن Servlet باید از `GenericServlet` استفاده و برای به کار بردن قابلیت های HTTP از زیر کلاس `HttpServlet` داخل خود `GenericServlet` استفاده می کنیم.

در Servlet همانند Applet ما متد `main()` نداریم (در برنامه های معمولی جاوا باید حتماً موجود باشد). به جای آن متد، سرور در هنگام گرفتن `Request` از سرویس گیرنده متد `service()` از Servlet را فراخوانی میکند.

یک Servlet در حالت کلی باید متد `service()` را خود پیاده سازی (`override`) کند تا درخواست مخصوص سرویس گیرنده را دستگیر (`handle`) کند. متد `service()` دو پارامتر قبول میکند اولی یک شی `Request` و دومی یک شی `Response` است. شی `Request` برای گرفتن درخواست سرویس گیرنده و دادن آن به Servlet و شی `Response` برای فرستادن پاسخ به سرویس گیرنده می باشد. شکل زیر این موضوع را بیان میکند.

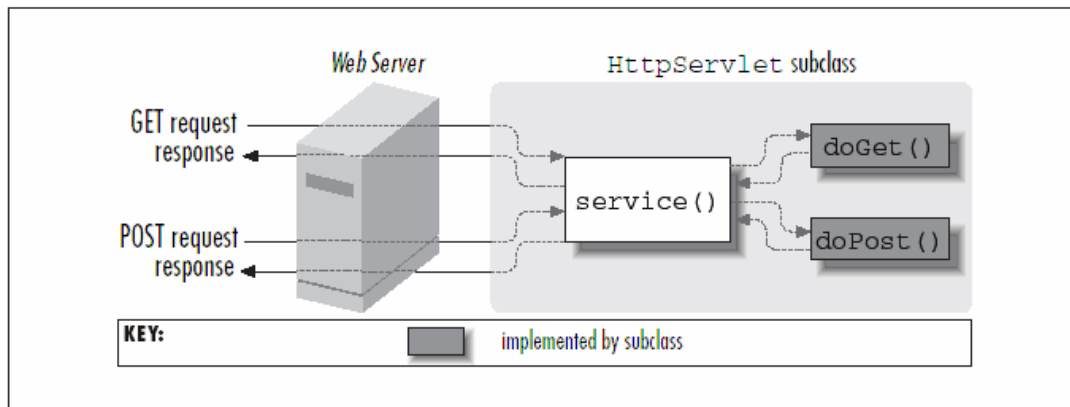


در مقابل `HttpServlet` متد `service()` را پیاده سازی نمی کند و به جای آن متد `doGet()` برای گرفتن درخواست های GET و از متد `doPost()` برای گرفتن درخواست های POST استفاده میکند.

بسته `HttpServlet` این قابلیت را دارد که هر دوی این متدها را پیاده سازی (`override`) کند و مستقل از نوع درخواست شود.

متد `service()` از `HttpServlet` تمام متدهای `doXXX()` را میتواند به کار برد و آن را `override` نمی کند.

در شکل زیر این موضوع نشان داده شده.



علاوه بر کلاسهای گفته شده در `javax.servlet` دو کلاس `ServletRequest` و `ServletResponse` برای دستیابی به اشیاء `Server`، `ServerResponse` و همچنین `Request` و `HttpServletResponse` در بسته `javax.servlet.http` برای دستیابی به `HTTP Request`، `HTTP Response` استفاده میشوند.

در بسته `javax.servlet.http` میتوانیم از کلاس `HttpSession`، `HttpCookie` برای پردازش جلسه کاری و داده های ذخیره شده توسط سرور روی سرویس گیرنده استفاده کنیم.

ایجاد صفحه

اکثر `HTTP Servlet` ها یک صفحه `html` ایجاد میکنند همانند اینکه یک `CGI Script` به اطلاعات دستیابی پیدا کرده و صفحه وب را ایجاد میکند اما با تفاوت که در `Servlet` میتوانیم همه وظایف را در داخل سرویس دهنده وب انجام دهیم نه بیرون آن. در اینجا اولین `Servlet` را که پیغام ساده `Hello World` را به سرویس گیرنده ارسال میکند نشان میدهیم.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Hello World</BIG>");
        out.println("</BODY></HTML>");
    }
}
```

این `Servlet` از `HttpServlet` استفاده کرده و متد `doGet()` را پیاده سازی می کند. هنگامی که سرویس دهنده وب یک `GET Request` دریافت می کند متد `doGet` را از `Servlet` فراخوانی کرده و آن درخواست را در شیء `HttpServletRequest` میگذارد. برای نوشتن خروجی و پاسخ به سرویس گیرنده `Servlet` از `HttpServletResponse` کمک گرفته و یک جریان خروجی (`PrintStream`) استاندارد با متد `getWriter()` از شیء ایجاد میکند.

PrintWriter وظیفه برگرداندن Unicode های کارکتر جاوا را به عهده دارد و در اینجا به صورت English میباشد.
استفاده از `setContentType("text/html")` برای تنظیم کردن محتوای خروجی و به صورت html از استانداردهای MIME می باشد.

اجرای Hello World

زمانیکه Servlet نوشته شد برای اجرای آن به دو چیز نیاز داریم اولی فایل کلاسهای Servlet API که برای کامپایل لازم هستند و دومی Servlet Engine که یک سرویس دهنده وب میباشد.

- برای فایل کلاسهای Servlet API چند راه وجود دارد.
- نصب (JSDK) Java Servlet Development Kit که در <http://java.sun.com/product/servlet> موجود است. در آن کدهای منبع، Servlet Engine و یک سرویس دهنده وب ساده موجود است.
- نصب یکی از Servlet Engine های موجود که هر کدام فایل کلاسهای Servlet API را در داخل خود دارد.

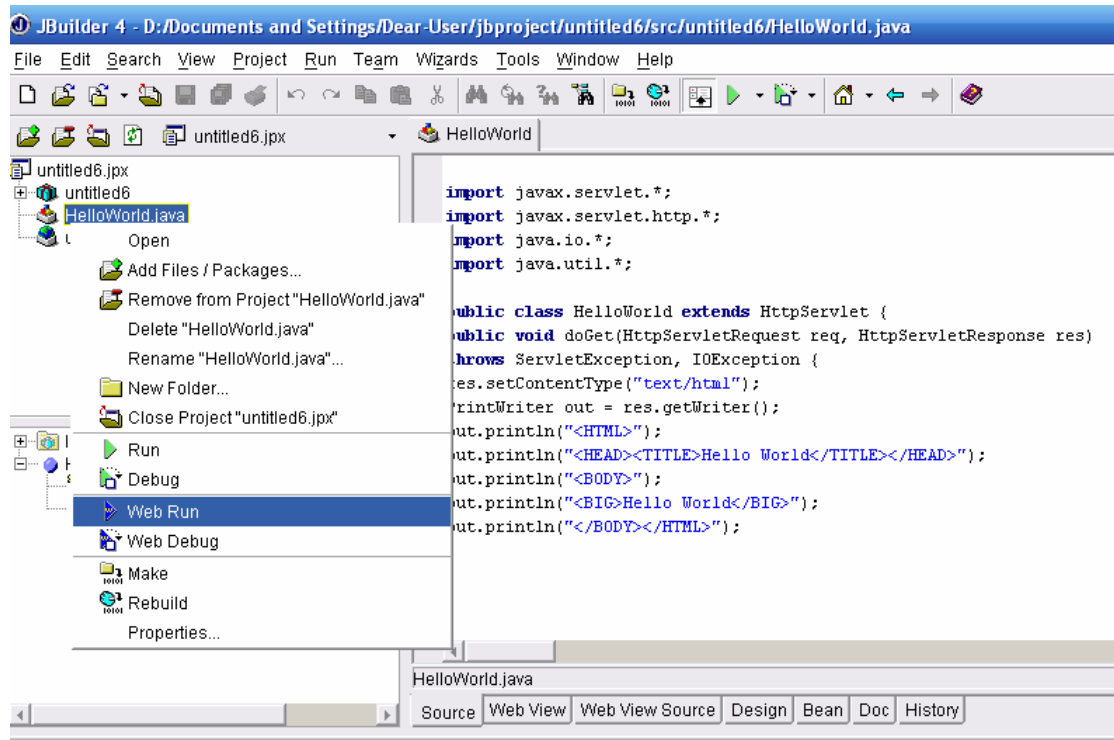
چندین دو جین از Servlet Engine در بازار موجود است که در قسمت موتور Servlet بیان شدند و میتوانید هر کدام از آنها را نصب کنید.
اگر شما از سرویس دهنده وب جاوا استفاده میکنید باید کد منبع Servlet را در پوشه `server_root/Servlet` قرار دهید (پوشه ای در محل نصب سرویس دهنده است) و این استاندارد محل فایل کلاسهای Servlet است. پس گذاشتن کد منبع در مکان درست باید آنرا کامپایل کنید.

به منظور کامپایل از کامپایلر استاندارد جاوا `javac` (یا با استفاده از IDE های جاوا همانند `jbuilder`) این کار را انجام میدهیم همچنین مطمئن شوید که بسته های `javax.servlet` , `javax.servlet.http` در مسیر مربوطه موجود باشند.
بعد از کامپایل باید سرویس دهنده را راه اندازی کنید تا Servlet را اجرا کنیم. برای اینکار فایل `httpd script` (در ویندوز `httpd.exe`) در `server_root/bin` پیدا کرده و آن را اجرا کنید و پیکره بندی پیش فرز در سرور روی پورت 8080 می باشد.

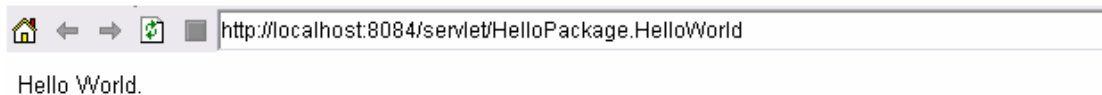
برای اجرای servlet چندین راه وجود دارد و یکی از آنها دستیابی به کمک URL است برای اینکار <http://server:8080/servlet/HelloWorld> را در مرورگر خود تایپ کنید.



اگر از IDE معمول مثل JBuilder استفاده میکنید کافی است بروی فایل جاوا ایجاد شده کلیک راست کرده و گزینه web run را همانند شکل زیر انتخاب کنید.



اگر Servlet قسمتی از یک بسته (package) باشد برای اجرا باید URL را به فرم `server_root/Servlet/package/name` تایپ کنید. بطور مثال : <http://server:8080/servlet/package.name.HelloWorld>



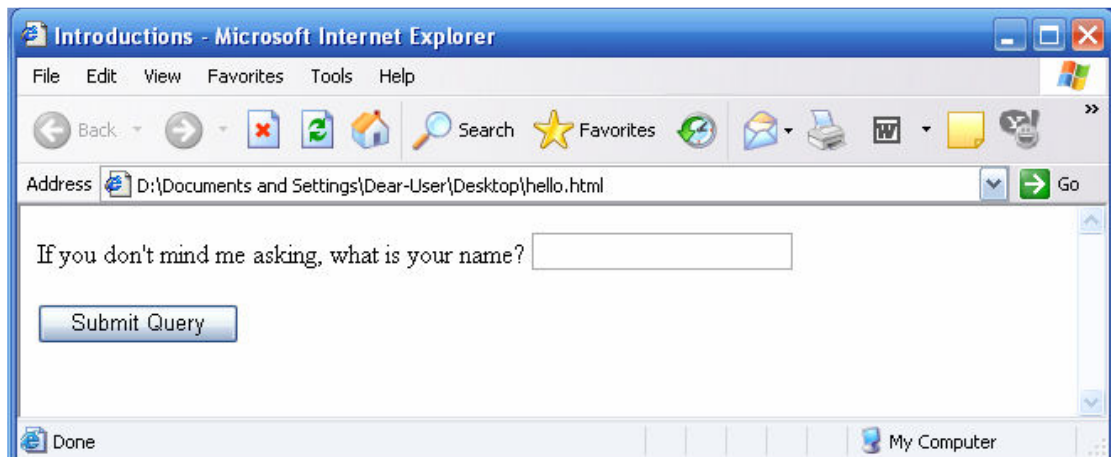
روش دیگر ارجاع به Servlet به کمک نام رجیستر شده می باشد و می تواند نام کلاس را روی خود نداشته باشد. در Java Web Server برای رجیستر کردن Servlet باید از Administration Tool که مدیریت سرور جاوا است و در پورت 9090 از سرور قابل دسترس است همانند <http://server:9090/> سپس در مدیریت Web Service قسمت Servlet انتخاب کنید و پس از آن Add New Servlet. در این قسمت می توانید نام ویژه ای را برای کلاس Servlet انتخاب کنید. بطور مثال اگر ما نام "hi" را برای HelloWorld به کمک URL <http://server:8080/servlet/hi> به دسترسی داریم.

روش سوم دسترسی به Servlet استفاده از Servlet Alias یا نام مستعار که شبیه به دیگر URL ها می باشد و تفاوت در این می باشد که سرویس دهنده با URL ویژه ای آن Servlet را اجرا میکند. به طور مثال برای HelloWorld Servlet از `http://server:8080/hello.html` استفاده می کنیم. استفاده از alias این امکان را می دهد که استفاده از Servlet را در سایت مخفی کنیم و یک Servlet به جای یک صفحه وب جایگزین شود. برای ایجاد نام مستعار در مدیریت Web Service قسمت setup را انتخاب کنید و سپس Servlet Aliases و در آخر از Add Alias.

گرفتن داده از فرمها

اولین Servlet یعنی HelloWorld چندان جالب نبود و حال می خواهیم يك Servlet ایجاد کنیم که از يك فرم html که کد آن در زیر موجود است نام کاربر را گرفته و در جواب به وي سلام دهد.

```
<HTML>
<HEAD>
<TITLE>Introductions</TITLE>
</HEAD>
<BODY>
<FORM METHOD=GET ACTION="/servlet/Hello">
If you don't mind me asking, what is your name?
<INPUT TYPE=TEXT NAME="name"><P>
<INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```



هنگامی که نام توسط کاربر ارسال می شود آن به Hello Servlet فرستاده می شود(به دلیل اینکه در قسمت ACTION فرم ورودی نام آن را روی سرور وارد کرده ایم). در اینجا ما متد GET در قسمت METHOD برای ارسال استفاده کرده ایم و برای مثال اگر شما نام mehdi را وارد کنید URL درخواست به فرم <http://server:8080/servlet/hello?name=mehdi> خواهد بود. در Servlet مورد نظر به کمک شیء HttpServletRequest به داده های ارسال شده از صفحه html دستیابی پیدا میکنیم. در زیر کد Hello Servlet نشان داده شده.

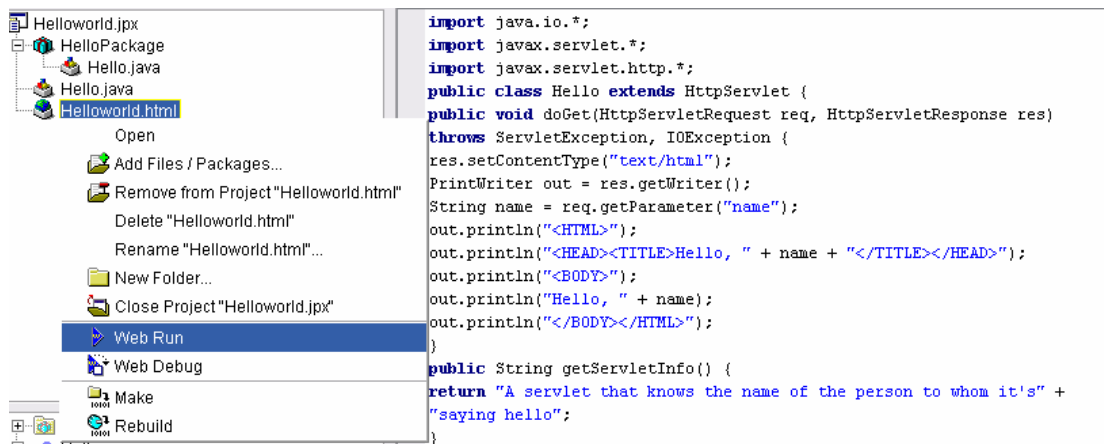
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Hello extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
```

```
throws ServletException, IOException {
res.setContentType("text/html");
PrintWriter out = res.getWriter();
String name = req.getParameter("name");
out.println("<HTML>");
out.println("<HEAD><TITLE>Hello, " + name + "</TITLE></HEAD>");
out.println("<BODY>");
out.println("Hello, " + name);
out.println("</BODY></HTML>");
}
public String getServletInfo() {
return "A servlet that knows the name of the person to whom it's" +
"saying hello";
}
}
```

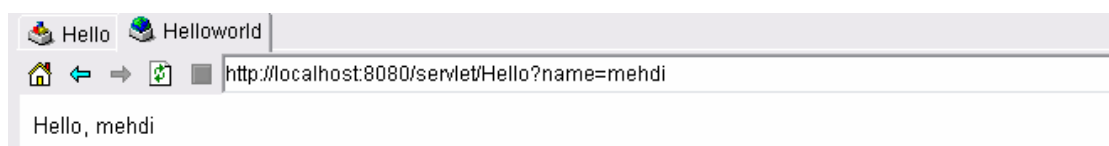
تنها تفاوت این Servlet با قبلی در این است که از متد `getParameter("name")` از شی Request استفاده کرده ایم. به کمک این متد به پارامتر ارسال شده از سرویس گیرنده توسط QueryString دستیابی پیدا می کنیم و اگر سرویس گیرنده پارامتر را خالی ارسال کند این متد null برمی گرداند.

متد `getServletInfo()` توسط Servlet برای برگرداندن اطلاعات درباره خود Servlet به سرویس دهنده وب می باشد. این اطلاعات می تواند راجع کاربرد، نویسنده، ورژن و سایر اطلاعات مربوط به یک Servlet باشد.

حال برای اجرای برنامه همورد نظر در محیط JBuilder روی فایل html ساخته شده کلیک راست کرده و گزینه web run را انتخاب کنید.



سپس در قسمت نام نام دلخواه را وارد کنید و روی دکمه کلیک کنید. نتیجه به شکل زیر خواهد بود.



گرفتن درخواستها به فرم POST

در Servlet قبلي براي گرفتن درخواست GET از متد doGet() استفاده كرديم حال براي گرفتن درخواست POST از متد doPost() بايد استفاده كنيم. تنها تغييرات لازم اين است كه متد doPost() به فرم زير را به Servlet قبلي اضافه كنيم.

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
```

```
doGet(req, res);
```

```
}
```

همچنين در فايل html متد را به POST تغيير دهيم.

```
<FORM METHOD=POST ACTION="/servlet/Hello">
```

در حالت كلي بهتر است كه در Servlet هر دو نوع متد doGet() , doPost() را براي قبول كردن هر دو نوع درخواست پياده سازي كنيم. مزيت doPost() در اين است كه قادر به دريافت داده ورودي با اندازه بالا است.

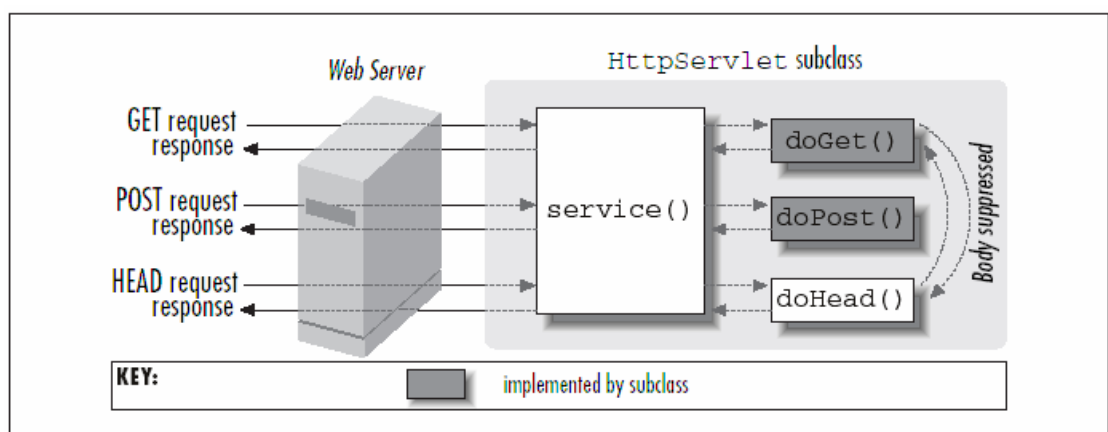
اگر متد doPost() در Servlet پياده سازي نشده باشد و سرويس گيرنده با اين متد ارسال كند از طرف سرويس دهنده يك پيغام خطا به سرويس گيرنده ارسال مي شود.

گرفتن درخواست به فرم HEAD

همان طور كه قبلاً گفته شد متد HEAD كه توسط سرويس گيرنده ارسال ميشود براي زماني كه فقط مي خواهيد سرآيند Response را ببينيد يا اندازه سند ارسالي را تعيين كند و زمان را تغيير دهد و ساير موارد استفاده كند. مثل دو متد قبلي در اينجا متد doHead() وجود ندارد و براي اينكار از متد doGet() استفاده ميكنيم.

متد service() از HttpServlet درخواست از نوع HEAD را شناسايي كرده و به طور ويژه با برخورد ميكند. در زمان رسيدن درخواست از نوع HEAD شي HttpServletResponse آن را دريافت كرده و آنرا به متد doGet() ارسال مي كند. متد doGet() آنرا به صورت عادي پردازش كرده و اما فقط قسمت سرآيند (header) به سرويس گيرنده ارسال مي شود. اين به اين معني است كه قسمت بدنه Response به دور انداخته مي شود.

شكل زير اين جريان را نشان ميدهد.



اگر چه اين استراتژي ساده و راحت است اما شما ميتوانيد كاري را بهبود دهيد با پيدا كردن درخواست HEAD در داخل متد doGet(). اين كار صرفه جويي را به همراه دارد چون وقت براي خروجي كه لازم نيست فرستاده شود صرف نمي شود.

مثال زیر همان Hello Servlet با استراتژی گفته شده است و از متد `getMethod()` در پیاده سازی کمک گرفته.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Hello extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        // Set the Content-Type header
        res.setContentType("text/html");
        // Return early if this is a HEAD
        if (req.getMethod().equals("HEAD")) return;
        // Proceed otherwise
        PrintWriter out = res.getWriter();
        String name = req.getParameter("name");
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello, " + name + "</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Hello, " + name);
        out.println("</BODY></HTML>");
    }
}
```

توجه کنید که همیشه Content-Type سرآیند تنظیم می شود حتی وقتی که درخواست HEAD Request می باشد. Content-Type نیز در سرآیند به سرویس گیرنده ارسال میشود.

اطمینان پیدا کنید که در آخر ارسال پاسخ به HEAD از دستور `return` استفاده می کنید و متد `System.exit(0)` را فراخوانی نکنید چون باعث خروج از سرویس دهنده وب خواهید شد.

ضمینه کردن در سمت سرور

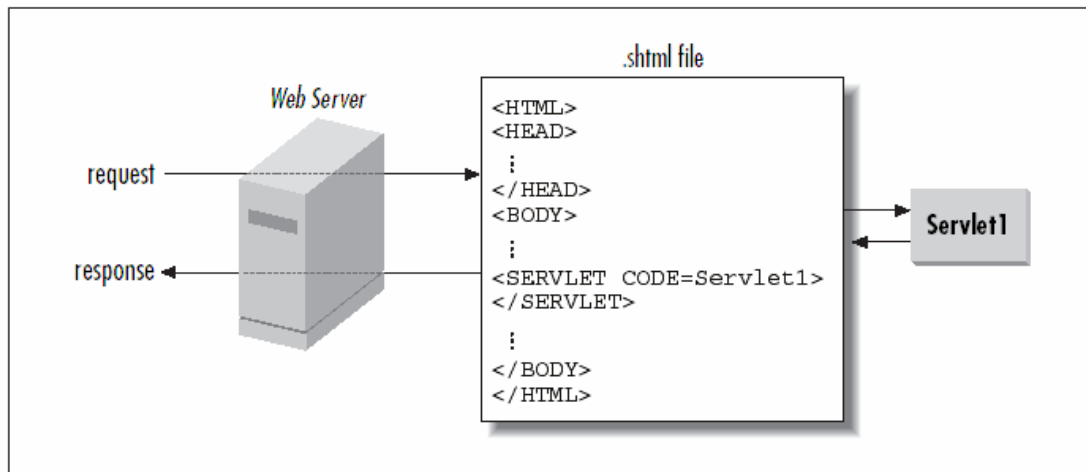
تمامی Servlet های که در قبل مشاهده کردید بطور کامل صفحه html را ایجاد می کردند و تمام آن به عهده Servlet بود. Servlet می تواند در داخل یک صفحه html قرار گیرد که به آن `server-side include (SSI)` می گویند.

اکثر سرویس دهنده ها اینگونه نوشتن را پشتیبانی میکنند و قبل از ارسال صفحه آنرا پردازش کرده و خروجی به سرویس گیرنده ارسال میشود. این روش همانند استفاده از Applet است.

```
<SERVLET CODE=ServletName CODEBASE=http://server.port/dir
initParam1=initValue1 initParam2=initValue2>
<PARAM NAME=param1 VALUE=value1>
<PARAM NAME=param2 VALUE=value2>
If you see this text, it means that the web server
providing this page does not support the SERVLET tag.
</SERVLET>
```

در تگ `<SERVLET>` مشخصه `CODE` برای نام کلاس Servlet یا نام رجیستر شده آن است. `CODEBASE` که اختیاری می باشد مکان لود شدن Servlet را نشان میدهد.

هر پارامتر برای ارسال شدن به Servlet باید از تگ <PARAM> استفاده کند و Servlet برای دریافت پارامترها از متد `getParameter()` در شی `ServletRequest` استفاده می کند. هر متغیر میتواند در انتهای تگ Servlet مقدار اولیه داشته باشد. اگر سرویس دهنده از SSI پشتیبانی کند تگ Servlet را پردازش کرده و اگر از SSI پشتیبانی نکند تگ Servlet را به سرویس گیرنده ارسال نخواهد کرد (فقط محتوای داخل آن ارسال می شود. به طور پیش فرض سرویس دهنده وب جاوا صفحات با پسوند `shtml` را با تگ Servlet پردازش می کند (در شکل زیر) و سرویس گیرنده تمام چیزهای بین <SERVLET> </SERVLET> را نخواهد دید بر عکس Applet.



نوشتن server-side Include

ضمینه کردن Servlet در صفحه وب زمانی مفید است که صفحه در اصل به صورت ایستا باشد اما شامل قسمتی پویا باشد. برای مثال فرض کنید که ما چندین صفحه داریم که باید زمان جاری را در آن نشان دهیم و این خود یک چالش است چون باید در آن ناحیه ای که سرویس گیرنده قرار دارد زمان را محاسبه کنیم. به کمک زمینه کردن در سمت سرویس دهنده و نوشتن راهنمای SSI که در داخل صفحه html کدهای جاوا مخصوص این کار را فراخوانی می کند میتوان این کار را انجام داد. کد زیر را در داخل یک فایل با پسوند `shtml` ذخیره کنید.

```

<HTML>
<HEAD><TITLE>Times!</TITLE></HEAD>
<BODY>
The current time here is:
<SERVLET CODE=CurrentTime>
</SERVLET>
<P>
The current time in London is:
<SERVLET CODE=CurrentTime>
<PARAM NAME=zone VALUE=GMT>
</SERVLET>
<P>
And the current time in New York is:
<SERVLET CODE=CurrentTime>
  
```

```
<PARAM NAME=zone VALUE=EST>
</SERVLET>
<P>
</BODY>
</HTML>
```

نام Servlet که از صفحه shtml فراخوانی می شود CurrentTime بوده و برای نشان دادن زمان در منطقه ای که به صورت پارامتر به آن ارسال می شود است. کد Servlet در زیر نوشته شده.

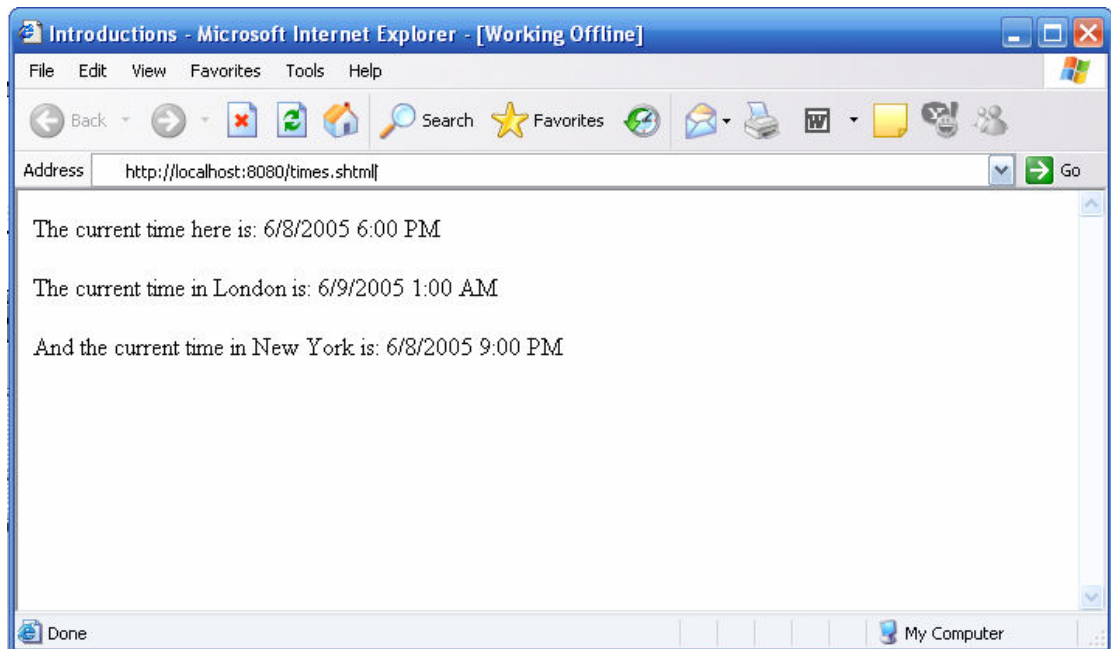
```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CurrentTime extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        Date date = new Date();
        DateFormat df = DateFormat.getInstance();
        String zone = req.getParameter("zone");
        if (zone != null) {
            TimeZone tz = TimeZone.getTimeZone(zone);
            df.setTimeZone(tz);
        }
        out.println(df.format(date));
    }
}
```

CurrentTime Servlet شبیه به Hello Servlet است و تفاوت چندانی بین آنها وجود ندارد. در CurrentTime Servlet همانند Servlet های قبلی از متد doGet() استفاده می کند. البته Servlet جاسازی شده در صفحه وب محدود می باشد چون همانند Request از طرف سرویس دهنده قابلیت سرآیند HTTP را ندارد.

برای نشان دادن زمان از کلاس Date , DateFormat استفاده می کنیم. برای تنظیم ناحیه باید از متد setTimeZone() در شی Dataformat کمک بگیریم.

برای مشخص کردن پارامتر های که توسط متد getParameter() از HttpServletRequest گرفته می شود در داخل فایل shtml از تگ <PARAM> استفاده می کنیم. اگر پارامتری به Servlet ارسال نشود به طور پیش فرز زمان سیستم به خروجی ارسال خواهد شد.

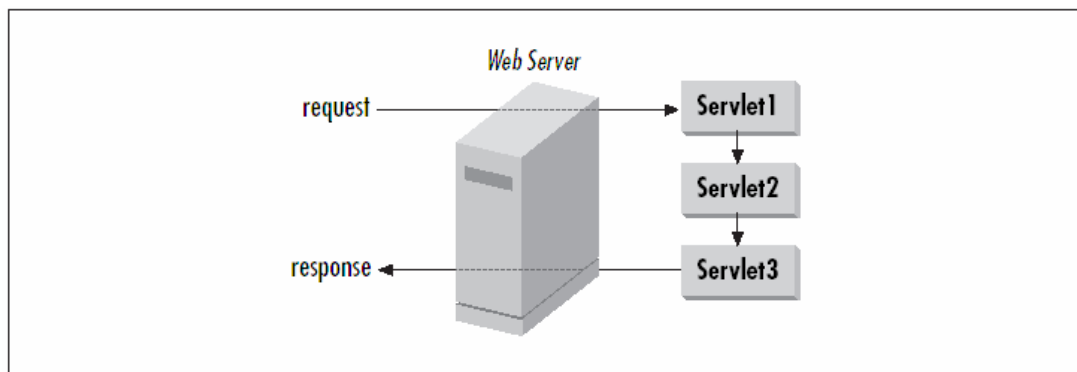
خروجی به شکل زیر خواهد بود.



زنجیره سازی و فیلتر کردن Servlet

همکاری Servlet ها برای ایجاد محتوای صفحات وب را زنجیر سازی (Servlet Chaining) می گویند

در بیشتر سرویس دهنده ها درخواست سرویس گیرنده به کمک دنباله ای از Servlet ها پاسخ داده میشود. درخواست client ابتدا به Servlet اول در زنجیر فرستاده می شود و پاسخ گرفته شده از آخرین Servlet در زنجیر به مرورگر سرویس گیرنده فرستاده می شود. در میان هر کدام از Servlet ها خروجی به Servlet به عنوان ورودی ارسال می شود. این عمل در شکل زیر نشان داده شده.



دو روش برای به راه انداختن زنجیره وجود دارد. در روش اول به طور صریح URL های واقعی زنجیر را برای سرور مشخص کرده و در روش دوم برای سرور مشخص می کنید که تمام خروجی Servlet ها به داخل يك Servlet ویژه رفته و در آخر کار خروجی آن Servlet به سرویس گیرنده ارسال شود.

زمانی که يك Servlet محتوا را به دیگر نوع محتوا تبدیل می کند به این عمل فیلتر (filtering) کردن می گویند.

زنجیره سازی کردن تفکر شما درباره محتوای واقعی وب را تغییر میدهد. در اینجا چیزهای وجود دارند که شما با آن را انجام دهید.

-تغیر سریع محتوای صفحات

بطور مثال برای بهبود سایت وب خود تمام تگهای <BLINK> صفحه وب را در سرویس دهنده که در مثال بعدی نشان داده شده پشتیبانی کنید. در واقع شما با کسانی که نمی توانند با زبان انگلیسی صحبت کنند به طور پویا صفحات را برای سرویس گیرنده به زبان خود ترجمه کنید. همچنین اگر نمی خواهید متنهای داخل صفحات وب را هر کاربری بخواند و بعضی متنها برای افراد خاص ظاهر شود. بطور مثال نام پروژه ای مخفی که هنوز نام آن فاش نشده.

شما می توانید کلمات(متنها) تازه را در سایت خود بالا ببرید مثلاً يك مجله خبری online و در اینجا نیاز به يك Servlet دارید که نام بالایی هزار شرکت را را بطور اتوماتیک پیدا کرده و در صفحه خانگی سایت برای آنها لینک ایجاد کند.

-ایجاد هسته برای محتوا و نمایش آن در فرمت ویژه

بطور مثال شما می توانید تگهای دلخواه را در صفحات وب جای داده و Servlet مخصوص که این جایگذاری با html انجام دهد. فرض کنید که يك تگ <SQL> مخصوص گرفتن محتوا با query از بانک اطلاعاتی بوده و نتیجه به جدولی در صفحه html می فرسد. در عمل این کار شبیه به استفاده از تگ <SERVLET> است.

پشتیبانی از دادهای که کمتر مورد استفاده اند

برای مثال شما می توانید از تصاویری که فرمت آنها کمتر پشتیبانی می شود به کمک يك فیلتر آنها را به انواع استاندارد مثل GIF, JPEG تبدیل کنید.

ممکن است شما از خود سوال کنید که چرا از زنجیره سازی استفاده کنید وقتی که می توانید به جای آن از script برای ویرایش فایلها در مکان ویژه استفاده کنید و Servlet به خاطر گرفتن درخواست يك سربرار اضافی دارد.

در زیر سه مزیت زنجیره سازی بیان شده:

- زنجیره سازی Servlet به راحتی قابل برگشت پذیری دارد زمانی که کاربران معترض هستند به استبداد شما در گذاشتن <BLINK> ، و شما می توانید سریعاً تغییرات را برگردانید و قشقرق را آرام کنید.
- این امکات وجود دارد که بطور اتوماتیک محتوا را ایجاد کرده و اطمینان پیدا کنید که محدودیت سازی برقرار است. بطور مثال تگهای ویژه جایگذاری شده و تصاویر ارسالی PostScript به تصویرهای معمولی تبدیل می شوند.
- آنها محتوای برای آینده را ایجاد می کنند و نیاز نیست هر بار برای اضافه کردن محتوای جدید script اجرا کنید.

ایجاد يك Servlet Chain

این مثال تمام تگ های <BLINK> را از صفحه html برمیدارد. اگر شما با این تگ آشنای ندارید باید بگویم که وقتی متنی در بین این تگ قرار میگیرد این متن بصورت درخشان و چشمک زن نشان داده می شود. مطمئن باشید که استفاده از آن مفید است، امتحان کنید. این مثله در بسیاری از نوشته ها گفته شده و یکی از جوکها در مورد html است. در مثال زیر با استفاده از زنجیره سازی تمام تگهای <BLINK> را از صفحات ایستای سایت حذف می کند. تمامی این کارها بطور اتوماتیک است و صفحاتی که در آینده اضافه می شوند را نیز در بر می گیرد. این Servlet متدهای getReader(), getContentType() را کامل میکند.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Deblink extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse res)
```

```

throws ServletException, IOException {
String contentType = req.getContentType(); // get the incoming type
if (contentType == null) return; // nothing incoming, nothing to do
res.setContentType(contentType); // set outgoing type to be incoming type
PrintWriter out = res.getWriter();
BufferedReader in = req.getReader();
String line = null;
while ((line = in.readLine()) != null) {
line = replace(line, "<BLINK>", "");
line = replace(line, "</BLINK>", "");
out.println(line);
}
}

public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
doGet(req, res);
}

private String replace(String line, String oldString, String newString) {
int index = 0;
while ((index = line.indexOf(oldString, index)) >= 0) {
// Replace the old string with the new string (inefficiently)
line = line.substring(0, index) +
newString +
line.substring(index + oldString.length());
index += newString.length();
}
return line;
}
}

```

در اینجا از دو متد doGet(),doPost() برای گرفتن هر دو نوع درخواست استفاده کرده ایم. در داخل متد doGet() ابتدا يك print writer گرفته سپس نوع محتوای دادهای دریافت شده با متد getContentType() مشخص می شود و از آن برای تنظیم کردن محتوای خروجی استفاده میشود.

اگر متد getContentType() مقدار null برگرداند این به این معنی است که داده ای برای deblink کردن نرسیده و فقط return می نویسیم. بمنظور خوانده داده از درخواست رسیده از شی BufferedReader با فراخواندن متد req.getReader() کمک می گیریم. Servlet صفحه html خروجی از Servlet قبلی از زنجیر را خط به خط خوانده و تمام <BLINK> یا </BLINK> با متد replace() برداشته و آنرا به سرویس گیرنده برمی گرداند(یا به Servlet بعدی در زنجیر ارسال می کند). ورژن قوی تر از این Servlet باید سرآیند HTTP رسیده را به همان شکل به سرویس گیرنده ارسال کند.

اجرا کردن Deblink Servlet

اگر از وب سرور جاوا استفاده می کند در قدم اول باید سرویس دهنده را از زنجیره ای بودن این Servlet آگاه سازید. برای این کار به قسمت مدیریت Web Service رفته و setup را انتخاب کنید سپس site را انتخاب کنید ودر آخر در قسمت option گزینه Servlet Chainnig را فعال کنید. On

این تکنیک که راه گریز نام دارد و این طور استنباط میشود که تمام کدهای HTML از فایلها با پسوند html. می آیند و این استنباط را تغییر می دهد. کدهای HTML می توانند از فایلها که این کدها را به طور پویا ایجاد می کنند آمده و این اجازه را می دهد که پسوندهای مختلف کار کنیم. پس ما نیاز به سومین تکنیک برای زنجیره کردن نیاز داریم که يك در اصل يك مدخل ایجاد میکند.

قصد ما این است که تمام فایلها text/html قادر به ارسال شدن به Deblink Servlet باشند. Administration Tool سرویس دهنده جاوا امکان این کار را به صورت گرافیکی ندارد و باید يك فایل Properties را ویرایش کنیم. این فایل در server_root/properties/server/javawebserver/mimeservlets.properties پیدا میشود. این فایل شامل رهنمود به فرم زیر است.

Java-internal/parsed-html=ssinclude

این رهنمود به این اشاره دارد که تمام پاسخها با سرآیند Content-type از Java-internal/parsed-html Servlet به internal/parsed-html ssinclude(server side include) ارسال خواهد شد. برای چه این کار لازم خواهد بود؟ بدون آن Servlet ssinclude فقط فایلها ایستا با پسوند shtml را پشتیبانی خواهد کرد. این همان راه گریز است که به طور خودکار صفحات شامل تگ <SERVLET> ایجاد کرده و پشتیبانی می کند. برای مشخص کردن ارسال محتوا بصورت text/html به Deblink باید رهنمود زیر را اضافه کنیم.

Text/html=Deblink

بعد از اعمال تغییرات باید سرویس دهنده را دوباره راه اندازی کرده و HelloWorld Servlet با تگ <BLINK> امتحان نماید.

Java Server Page

بعد از ظهور Servlet شرکت sun يك راه جدید برای استفاده از Servlet با نام Java Server Page(JSP) ارائه داد و توانایی ها و قابیتهای همانند شبهه به Active Server Page(ASP) شرکت میکروسافت دارد. JSP از بسیاری لحاظ شبهه به server side include عمل کرده و تفاوت اصلی این است که به جای جای دادن تگ <SERVLET> در صفحه html آنها به طور جزئی در کد Servlet قرار می دهیم. همانند زنجیره سازی و .

JSP همانند زنجیره سازی و server side include نیازی به تغییر Servlet API ندارد، اما نیاز به پشتیبانی ویژه از طرف سرویس دهنده دارد و این قابلیت در سرویس دهندهای ورژن 1.1.1 به بعد موجود است.

استفاده از JSP

تکنولوژی JSP به ما این امکان را می دهد که کد هي Servlet را در داخل يك صفحه ایستا HTML قرار دهیم. هر بلوک از کد Servlet در تگ <% %> محصور شده(با نام scriptlet شناسایی میشوند) و برای آسوده گی از چهار متغیر از قبل تعریف شده استفاده میکند. Request

يك Servlet Request که از نوع شي HttpServletRequest می باشد.

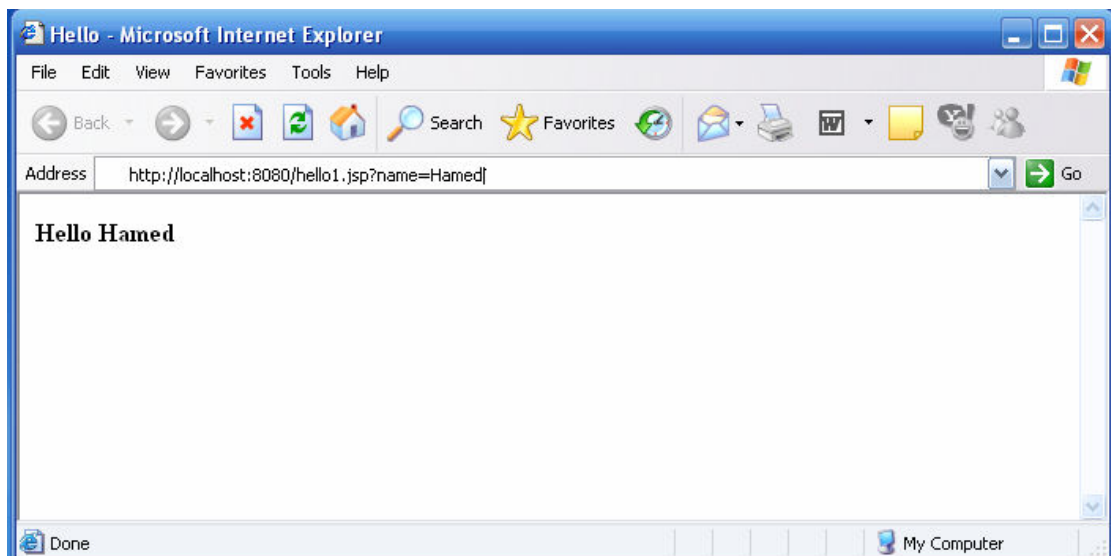
Response

يك Servlet Response که از نوع شي HttpServletResponse می باشد.

Out

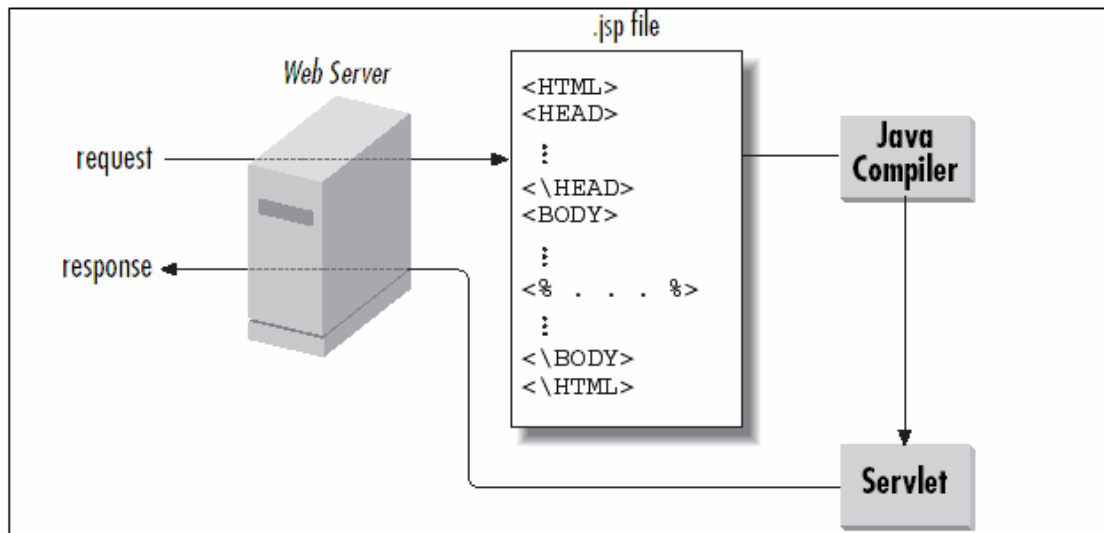
يك output writer كه از نوع شي PrintWriter مي باشد.
 in
 يك input writer كه از نوع شي BufferedReader مي باشد.
 در مثال بعدي يك صفحه JSP كه پيغام Hello را نشان مي دهد با كمّي تغيير نسبت به كد هاي قبلي كامل كرده ايم. لازم به ذكر است كه متغير هاي Request,out از قبل تعريف شده مي باشند.
 براي اجراي آنرا با نام hello1 و با پسوند jsp در فولدر ريشه سرور يعني server_root/public_html ذخيره كنيد. براي دستيابي به اين صفحه آدرس <http://server:port/hello1.jsp> را وارد كنيد. نحوه اجرا اين صفحه در شكل نشان داده شده.

```
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
<%
if (request.getParameter("name") == null) {
out.println("Hello World");
}
else {
out.println("Hello, " + request.getParameter("name"));
}
%>
</H1>
</BODY>
</HTML>
```



بقیه کار

نحوه کار JSP به این شکل می باشد که سرویس دهنده به طور خود کار Servlet برای ساختن محتوا ایجاد، کامپایل، لود و اجرا می کند. نحوه این کار در شکل زیر نشان داده شده.



این طور می توان فکر کرد که يك Servlet در پیش زمینه اجرا میشود و نام آن را Servlet زحمت کش گذاشته اند و صفحه html را با فراخوانی متد `out.println()` کامل میکند. ممکن است Servlet زحمت کش برای `hello1.jsp` به شکل زیر باشد.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class _hello1_xjsp extends HttpServlet {
public void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
BufferedReader in = request.getReader();
out.println("<HTML>");
out.println("<HEAD><TITLE>Hello</TITLE></HEAD>");
out.println("<BODY>");
out.println("<H1>");
if (request.getParameter("name") == null) {
out.println("Hello World");
}
else {
out.println("Hello, " + request.getParameter("name"));
}
out.println("</H1>");
out.println("</BODY></HTML>");
}
}

```

در هنگام دستیابی به صفحه JSP مدت زمانی برای ایجاد پاسخ صرف می شود و این لازم است تا Servlet پیش زمینه را ایجاد و کامپایل کند. به طور حتم در خواسته های بعدی برای آن صفحه JSP دارای سرعت بیشتری خواهند بود چون سرویس دهنده فقط يك بار Servlet پیش زمینه آن را کامپایل می کند آن هم در بار اول. برای هر تغییر نیاز به کامپایل دوباره داریم و هر گونه خطا در ضمن کامپایل به سرویس گیرنده ارسال خواهد شد.

عبارت و رهنمودها (expression ,directives)

در scriptlet داخل تگ `<% %>` می توانیم از عبارت ها و رهنمودها استفاده کنیم. مثلاً به جای استفاده از `out.println()` برای نوشتن در خروجی کافی است تگ را با علامت مساوی استفاده کنیم همانند `<%= foo%>` که مقدار متغیر در خروجی دیده خواهد شد. رهنمودهای JSP به فرم `<%@ %>` بوده و اجازه می دهند که Servlet پیش زمینه را کنترل کرده. مثلاً برای تنظیم کردن نوع محتوا ، اضافه کرده بسته های برنامه جاوا، تکمیل کردن یک رابط، extend کردن یک super class و گرفتن درخواستهای GET,POST . در تگهای رهنمود متغیر های کلیدی به فرم زیر می توانند مقدار بگیرند.

```
<%@ varname = "value" %>
```

در رهنمودها سه متغیر وجود دارد که شما می توانید به آنها مقدار دهید.

content-type

نوع محتوای صفحه کلی را مشخص میکند. بطور پیش فرز این مقدار "text/html" می باشد.

```
<%@ content-type = "text/plain" %>
```

import

لیست کلاسهای را که سرویس دهنده باید آنها را به برنامه وارد کند و اگر بیش از یک کلاس داشته باشیم می توانیم آنها را با کاما از هم جدا کنیم.

```
<%@ import = "java.io.*,java.util.Hashtable" %>
```

extend

مشخص کننده ابر کلاسهای که Servlet باید آنها extend کند.

```
<%@ extend = "CustomHttpServletSuperclass" %>
```

ابر کلاس پیش فرز HttpServlet میباشد.

Implements

مشخص کننده لیست رابطهای است که Servlet باید آنها تکمیل کند. رابطهای چند تایی با کاما از هم جدا می شوند.

```
<%@ implements = "Serializable" %>
```

method

مشخص کننده نوع متدی می باشد که سرویس گیرنده درخواست را با آن به Servlet ارسال می کند. به طور پیش فرز همه متدها را پشتیبانی می کند.

```
<@ method = "doPost" %>
```

language

مشخص کننده زبان scripting است و به صورت پیش فرز آن java است. شما می توانید سایر زبانهای مورد قبول را استفاده کنید.

```
<%@ language = "java" %>
```

مثال زیر صفحه Hello را با استفاده از عبارت ها و رهنمودها در JSP پیاده سازی کرده.

```
<%@ method = "doPost" %>
```

```
<HTML>
```

```

<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
<% if (request.getParameter("name") == null) { %>
Hello World
<% } else { %>
Hello, <%= request.getParameter("name") %>
<% } %>
</H1>
</BODY>
</HTML>

```

Servlet پیش زمینه این صفحه JSP به مثال قبلی بسیار نزدیک است با این تفاوت که متد doPost() را به جای service() کامل می کند.

اعلان کردن (Declaration)

در بعضی از مواقع ضروری است که در یک صفحه JSP متدها و متغیرهای غیر محلی را در Servlet پیش زمینه (زحمت کش) تعریف کنیم. به این عمل این عمل اعلان کردن در JSP می گویند.

یک اعلان کردن با `<SCRIPT RUNAT="server">` شروع شده و با تگ `</SCRIPT>` خاتمه می یابد. در میان این تگ شما می توانید کد هر Servlet را در آن زمینه (include) کنید که در داخل آن از متدهای آن Servlet استفاده کنید. در مثال زیر از متد `getName()` که ایجاد شده استفاده می شود.

```

<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
Hello, <%= getName(request) %>
</H1>
</BODY>
</HTML>
<SCRIPT RUNAT="server">
private static final String DEFAULT_NAME = "World";
private String getName(HttpServletRequest req) {
String name = req.getParameter("name");
if (name == null)
return DEFAULT_NAME;
else
return name;
}
</SCRIPT>

```

Servlet پیش زمینه که برای این Servlet ایجاد می شود، ممکن است به فرم زیر باشد.

```

import java.io.*;
import javax.servlet.*;

```

```

import javax.servlet.http.*;
public class _hello3_xjsp extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        BufferedReader in = request.getReader();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>");
        out.println("Hello, " + getName(request));
        out.println("</H1>");
        out.println("</BODY></HTML>");
    }
    private static final String DEFAULT_NAME = "World";
    private String getName(HttpServletRequest req) {
        String name = req.getParameter("name");
        if (name == null)
            return DEFAULT_NAME;
        else
            return name;
    }
}

```

کامپوننت‌های JSP و JavaBeans

یکی از مزیت‌های و قدرت‌های JSP در این است که می‌توانیم در آنها از کامپوننت‌های جاوا یعنی JavaBeans استفاده کنیم. این کامپوننت‌ها کلاس‌های جاوا می‌باشند که قابلیت استفاده مجدد داشته و متدها و متغیرهای آنها به کمک نام ویژه‌شان قابل دسترسی است. به کمک تگ <BEAN> می‌توان آنها را به طور مستقیم در صفحه JSP استفاده کرد. يك JavaBeans وظایف خوش تعریف مثل اجرا پرس جو از بانک اطلاعاتی، اتصال به سرویس دهنده mail، مدیریت اطلاعات درباره سرویس گیرنده و سایر را می‌تواند انجام دهد و نتایج را برای صفحه JSP با متدهای دسترسی قابل دسترسی می‌کند.

تفاوت میان JavaBeans گذاشته شده در داخل يك صفحه JSP و کلاس معمولی Servlet این است که سرویس دهنده وب با JavaBeans رفتار متفاوتی دارد. برای مثال سرویس دهنده به طور خود کار متغیرهای نمونه کامپوننت (propertyها) را با پارامتر رسیده از سرویس گیرنده تنظیم میکند (مقدار را در داخل آنها قرار می‌دهد). مثلاً متغیر name از JavaBeans که با متد setName(String name) تنظیم می‌شود و سرویس دهنده با فراخوانی این متد آنرا تنظیم می‌کند. توجه کنید که نیاز به متد ()getParameter ندارید.

همچنین يك bean می‌تواند توسط سرویس دهنده به طور خودکار مدیریت شود و برای درخواست‌های ویژه از سرویس گیرنده تخصیص یابد. برای این کار باید از جلسه کاری سرویس گیرنده یا Session اسفاده کرد یعنی زمانی که سرویس گیرنده متصل می‌شود جلسه کاری شروع می‌شود و هنگامی که سرویس گیرنده قصد قطع اتصال را دارد Session پایان می‌پذیرد.

يك bean حتي مي‌تواند در داخل يك Servlet استفاده شود! اگر سرویس دهنده implement يك bean را در داخل رابط javax.servlet.Servlet پیدا کند آنگاه متد ()service از bean را برای

هر درخواست سرویس گیرنده فراخوانی خواهد کرد، همچنین متد `init()` از `bean` را وقتی که آن برای اولین بار ایجاد می شود فراخوانی می کند. سودمندی این قابلیت قابل بحث است اما می تواند توسط `bean` برای بررسی چگونه گی آن پیش از گرفتن درخواست سرویس گیرنده استفاده گردد. فرم استفاده از `bean` باید به شکل زیر باشد.

```
<BEAN NAME="lookup name" VARNAME="alternate variable name"
TYPE="class or interface name" INTROSPECT="{yes|no}" BEANNAME="file
name"
CREATE="{yes|no}" SCOPE="{request|session}">
<PARAM property1= value1 property2= value2>
</BEAN>
```

صفات در تگ `<BEAN>` به صورت زیر تنظیم می شوند.

NAME

برای مشخص کردن نام `bean` مورد استفاده در محدوده درخواستهای سرویس گیرنده و اگر `bean` با این نام موجود باشد در این صفحه استفاده خواهد شد.

VARNAME

مشخص کننده نام متغیرهای `Bean` است. صفحه از این نامها برای ارجاع به `bean` و فراخوانی متد های مربوط به `property` استفاده می کند.

VARNAME="prefs"

TYPE

نام کلاس `bean` یا نوع رابط را مشخص می کند برای مثال:

TYPE="UserPreferencesBean"

به طور پیش فرز این مقدار `java.lang.Object` می باشد.

INTROSPECT

اگر سرویس دهنده بخواهد ویژه گیهای `bean` را با استفاده پارامترهای ارسال شده از سرویس گیرنده تنظیم کند این مقدار را باید برابر با "yes" قرار دهیم و در غیر این صورت با "no" این کار را انجام می دهیم.

BEANNAME

به منظور مشخص کردن فایل سریال شده یا فایل کلاس که شامل `bean` مورد نظر است و زمانی که برای اولین بار `bean` ایجاد می شود سرویس دهنده از آن استفاده می کند. این صفت بصورت اختیاری می باشد.

BEANNAME="hellobean.ser"

CREATE

اگر `bean` هنگام درخواست سرویس گیرنده موجود نباشد آنرا ایجاد کرده و دو مقدار "yes" , "no" را قبول می کند. اگر مقدار "no" را داشته باشد و نمونه از قبل ایجاد شده در سرور موجود نباشد به سرویس گیرنده پیغام خطا را ارسال خواهد کرد. بطور پیش فرز مقدار "yes" را در خود دارد یعنی به طور خودکار `bean` را ایجاد خواهد کرد.

SCOPE

برای مشخص کردن تخصیص `bean` ها به درخواستهای ویژه یا جلسه کار سرویس گیرنده(به طور خودکار تا زمانی قابل دسترس باشد که سرویس گیرنده به سرور متصل است).مقدار های قابل قبول دو مقدار `request` , `session` می باشند که پیش فرز `request` می باشد.

پارمترهاي كه به يك bean پاس مي شوند با ليستي از تگهاي <PARAM> در داخل تگ <BEAN> مشخص ميشوند.
 مثال زير كه با استفاده از HelloBean پيغام hello را به كاربر مي گويد نحوه استفاده از يك bean در صفحه JSP را نشان مي دهد.

```
<%@ import = "HelloBean" %>
<BEAN NAME="hello" TYPE="HelloBean"
INTROSPECT="yes" CREATE="yes" SCOPE="request">
</BEAN>
<HTML>
<HEAD><TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
Hello, <%= hello.getName() %>
</H1>
</BODY>
</HTML>
```

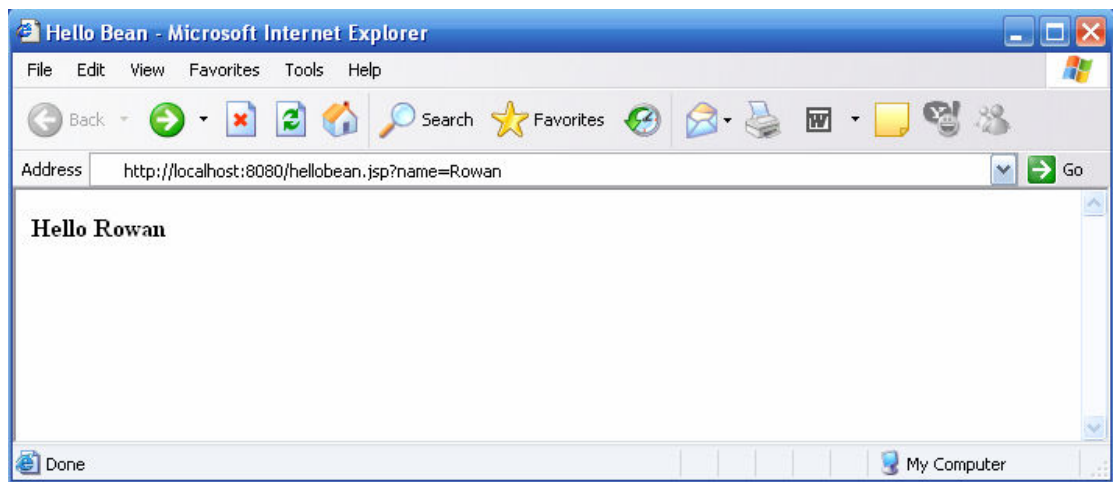
مشاهده مي كنيد كه استفاده از bean كد را در داخل صفحه JSP كد هاي را کاهش داده و باعث مي شوند كه كد داخل JSP وضوح و قدرت (كامپوننتها قدرت بعدي در ارتباط با ساير برنامه ها دارند) بعدي داشته باشد. حتي استفاده از API هاي خوش تعريف كه با bean تعامل دارند باعث مي شود كه افراي كه برنامه نويس حرفه اي نيستند صفحات JSP پيشرفته اي طراحي كنند.

HelloBean كه بايد در مسير كلاسهاي سرويس دهنده قرار گيرد همانند:
 server_root/classes

در زير كد HelloBean كامل شده.

```
public class HelloBean {
private String name = "World";
public void setName(String name) {
this.name = name;
}
public String getName() {
return name;
}
}
```

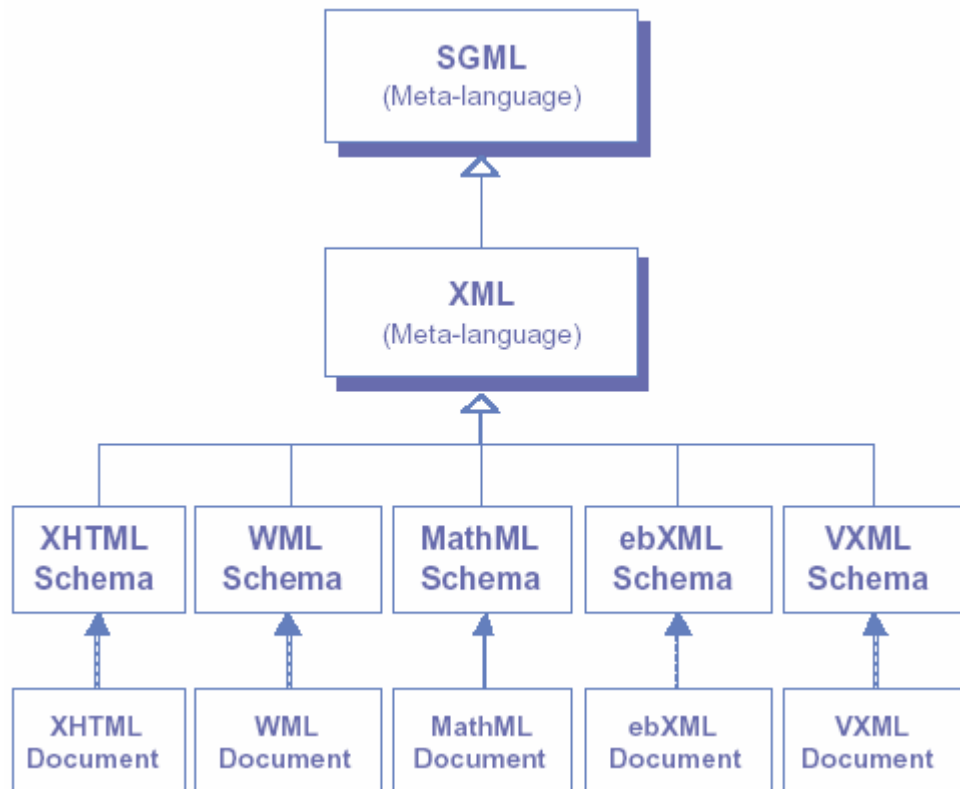
اين bean يك property به نام name داشته كه با متد setName() مقدار مي گيرد و با متد getName() مقدار داخل خود را برمي گرداند. به طور پيش فرز مقدار آن "world" است. براي تست كردن آدرس <http://server:port/hellobean.jsp> را در مرورگر خود وارد كنيد.



فصل ششم

Java XML

زبان (eXtensible Markup Language) XML در عمل يك زبان نمي باشد و يك فرازبان (matalanguage) براي ايجاد ساير زبانها ميباشد. از Xml براي ايجاد ساختار ساير زبانها و مستندات خود تعريف استفاده مي شود. Xml در جنبه اي گوناگون تجاري و صنعتي استانداردهاي فراواني پيدا کرده، همانند Mathematicla Markup Language (MathML)، Electronic Business XML (ebXML)، Voice Markup، Language (VXML) و ساير نماير آنها. اين مفاهيم در شكل زير نمايان ميباشد.



Xml شامل نشانه ها (markup) و محتوا می باشد. نشانه ها شامل يك سري تگها كه محتوای داخل مستندات را شرح میدهد میباشند. يك شرح قابل تطابق از داده ها به ما این امکان را می دهند كه به راحتی داده را ارسال و دریافت کرده و آنها را به سایر فرمتها تبدیل كنیم. به طور مثال صاحبان تجارت و صنایع از xml برای تعویض اطلاعاتی مثل قیمتها، دارای ها، محصولات با سایر شركتها استفاده می كنند. استفاده از xml بسیار گسترده شده به طوري كه رتبه اول را در جابه جاي اطلاعات در اینترنت دارد.

هر سند xml برای خود يك گرامر مخصوص به خود دارد كه مشخص كننده مجموعه قوانین و سازمان آن سند است. بطور مثال يك عنصر به نام قیمت در مستندات ebXML دارای معنی بوده اما در مستندات MathXML دارای معنی نمی باشد. پس تمام مستندات xml باید این گرامر را برای خواننده و ایجاد شدن داشته باشند. xml قابلیتهاي برای ایجاد این گرامرها و تطابق با مستندات رسیده در خود ایجاد کرده. Xml در خود يك پایه برای زبانهاي سطح بالاتر برای تعویض داده ها در بین کامپوننتهاي نرم افزاري، سیستمهاي enterprise دارد. امروزه اکثر زبانهاي برنامه نویسی امکاناتي برای تجزیه و ترجمه کردن مستندات xml و ابزار هاي برای ایجاد، دستکاري انواع اطلاعات بر پایه xml را در خود تعیبه کرده اند. سازمان دهی بر اساس xml امکان سطح استفاده مجدد بالاي برای کامپوننتها در سیستم هاي توزیع شده فراهم میکند. برای مثال در جاوا تمام کامپوننتها Deployment Descriptor با فرمت xml برای نصب در يك سرویس دهنده یا برنامه دارند و این كمك میکند كه در سیستمهاي مختلف به راحتی از آنها استفاده کرد. این مستندات مشخصات و ویژه گی هاي کامپوننت را برای استفاده در خود دارند. چون xml به عنوان يك استاندارد تبادل در صنعت کامپیوتر معرفی شده در بیشتر مواقع تبادل اطلاعات و پیغام رسانی از آن استفاده می کند و این زمانی مهم است كه سیستمهاي كه با هم تفاوت ساختار دارند.

از xml میتوان به عنوان يك ذخیره كننده داده هاي ماندگار نیز استفاده کرد به طور كه به جاي گذاشتن آنها در بانک هاي اطلاعاتي آنها را در فیاالهاي xml قرار میدهم. امروزه اکثر بانکهاي اطلاعاتي امکان import , export داده ها را به صورت xml دارند.

يك سند xml نمونه

برای تشریح بهتر xml و تکنولوژی هاي وابسته يك مثال عملي را ایجاد خواهیم کرد. از این مثال برای توضیح تکنولوژی هاي متفاوت xml استفاده خواهیم کرد. در زیر سند xml كه برای مشخص کردن کاتالوگ محصولات است را مشاهده میکنید.

```
<?xml version="1.0"?>
<product-catalog>
  <product sku="123456" name="The Product">
    <description locale="en_US">
      An excellent product.
    </description>
    <description locale="es_MX">
      Un producto excelente.
    </description>
```

```

<price locale="en_US" unit="USD">
  99.95
</price>
<price locale="es_MX" unit="MXP">
  9999.95
</price>
</product>
</product-catalog>

```

در بلا یك كاتالوگ محصول را مشاهده میکنید كه فقط يك محصول را نشان مي دهد. این محصول شامل اطلاعات شرح، قیمت، و شماره SKU است. در این جا توجه كنید كه شرح و قیمت این محصول برای مكانهاي متفاوت جدا مي باشد، مثلاً برای كشور مكزیك قیمت 9999.95 MXP به واحد پول این كشور است.

رده بندي تكنولوژي هاي XML

استانداردها و تكنولوژي هاي متفاوتي برای xml تا كنون به وجود آمده اند كه هر يك کاربرد خاص خود را دارد. اینها مخصوص جاوا یا زباني خاص نیستند و برای استفاده كردن ساده و مدیریت شده xml ساخته شده اند. در این بخش يك مرور خلاصه بر برخي از آنها خواهیم داشت. برای اطلاعات بیشتر در این زمینه از سایت <http://www.zvon.org> مي توانید آنها را در يافت كنید.

تكنولوژي اعتبار سنجي XML

نتایج يك سند xml به دو روش بازیابي مي شوند. يك روش با استفاده از شمای xml (xml schema) و روش دیگر استفاده از تعریف کننده نوع سند (Document Type Definition). هر کدام از دو روش مشخص کننده قوانین آن سند xml بوده و برای بازیابي از آنها استفاده مي شود و بار سنگيني را از دوش كد هاي برنامه شما برمي دارد.

تعریف کننده نوع سند (Document Type Definition)

اولین و قدیمی ترین مکانیزم تعریف زبان Document Type Definition یا به اختصار DTD مي باشد. DTD يك فایل متني شامل مجموعه قوانین در باره ساختار و محتوای يك سند xml بوده و مجموعه عناصری معتبر شامل مشخصات و ترتیب را كه ممكن است در در سند xml ظاهر شود لیست مي كند. يك DTD سلسله مراتب ساختاری يك سند را دیکته مي كند، آنهایی كه به شدت مهم مي باشند برای اعتبار سنجي ساختار xml. برای مثال عنصر نیمکت یا Couch ممكن است در داخل عنصر LivingRoom معتبر باشد، اما در عنصر BathRoom معتبر نباشد. DTD همچنین مشخصات عناصر مثل عناصر خیلی ویژه ، مقادیر قابل قبول و اختیاری یا اجباری را تعریف مي كند.

در زیر DTD مربوط به مثال كاتالوگ محصولات را مشاهده مي كنید.

```

<!ELEMENT product-catalog (product+)>
<!ELEMENT product (description+, price+)>
<!ATTLIST product
  sku ID #REQUIRED

```

```

    name CDATA #REQUIRED >
<!ELEMENT description (#PCDATA)>
<!--ATTLIST description
    locale CDATA #REQUIRED
    >
<!ELEMENT price (#PCDATA)>
<!--ATTLIST price
    locale CDATA #REQUIRED
    unit CDATA #REQUIRED
    >

```

این DTD توسط پارسر xml برای اعتبار سنجی سند استفاده می شود، ما میتوانیم یک DTD را در یک فایل جدا ذخیره کرده و اشاره گر به آن را در خود سند xml شبیه به زیر ذخیره کنیم.

```
<!DOCTYPE product-catalog SYSTEM "product-catalog.dtd">
```

استفاده از این دستور پارسر را وادار می کند که "product-catalog.dtd" را در همان فهرست فایل که سند xml وجود دارد جستجو کرده و از آن استفاده کند.

تعریف کننده الگو (schema)

اگر چه اولین مشخص کننده اعتبار سند DTD بود ولی در توسعه های enterprise محدودیتهای را با خود به همراه داشت. یکی از این محدودیتهای اصلی این بود که معتبر بودن خود DTD معلوم نبوده و بنا براین نیاز به ابزار های برای تجزیه کردن آن بود. یک عیب دیگر این بود که DTD ها محدودیت زیادی در ایجاد ساختار و محتوای مستندات xml داشتند. آنها قادر به حل تصادف نامها در فضای نامی و رابطه بین عناصر پیچیده نبوده و خاصیت پیمانه ای بودن را پشتیبانی نمی کردند. بنابر این کنسرسیوم W3C تصمیم به ایجاد تکنولوژی جایگزین برای آن در نظر گرفت. این تکنولوژی XML Schema Definition (XSD) نام گرفت. XSD خود بر پایه xml بوده و استفاده از آن رابطه بین عناصر و فضای نامی عناصر را پشتیبانی میکند. XSD مشخصات کاملتری از سند xml نسبت به DTD ارائه میکند. در XSD امکان به ارث بردن ویژه گی ها برای داده ها با نوع پیچیده وجود دارد. هم اکنون بسیاری از انواع داده های اولیه به صورت خودکار در XSD موجود می باشند.

در قسمت زیر الگوی XSD مربوط به مثال product catalog را مشاهده میکنید.

```

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element type="product-catalog"/>

  <xsd:complexType name="productCatalog">
    <xsd:element type="productType"
      minOccurs="1"/>
  </xsd:complexType>

```

```

<xsd:complexType name="productType">
  <xsd:element name="description"
    type="xsd:string" minOccurs="1">
    <xsd:attribute name="locale"
      type="xsd:string"/>
  </xsd:element>
  <xsd:element name="price"
    type="xsd:decimal" minOccurs="1">
    <xsd:attribute name="locale"
      type="xsd:string"/>
    <xsd:attribute name="unit"
      type="xsd:string"/>
  </xsd:element>
  <xsd:attribute name="sku"
    type="xsd:decimal"/>
  <xsd:attribute name="name"
    type="xsd:string"/>
</xsd:complexType>
</xsd:schema>

```

این XSD يك نوع داده پیچیده به نام productType را که بالاتر از دیگر نوع داده ها قرار دارد تعریف می کند. داده ها با نوع پیچیده شامل مشخصات و دیگر عناصر از آن داده می باشند. این مثال اشکارا سطح ساختار پیچیده را کمتر می کند تا شما بتوانید با XSD آنها را تعریف کنید.

همان طور که می بینید تمام عناصر نام، نوع و حداقل مورد نیاز آنها مشخص گردیده و یکی از قدرت XSD تعیین نوع و محدوده ورودی است. در اینجا به شما توصیه می شود که در تعریف کردن XSD ها ماهر شوید، چون بعدها از آنها در برنامه های خود استفاده خواهید کرد.

این نکته را باید بیان کرد که برخی تجزیه کننده ها (پارسرها) در همه مواقع برای اعتبار سنجی پیشنهاد نمی شوند. اعتبار سنجی مستندات xml در طی توسعه و تست کردن بسیار با اهمیت می باشد و وقتی که داده ها بین سیستم های enterprise به اشتراک گذاشته می شوند این امر حیاتی می شود. از این اطمینان پیدا کنید که ارسال و دریافت داده ها در فرمت معتبر صورت گیرد.

تکنولوژی تجزیه کردن XML

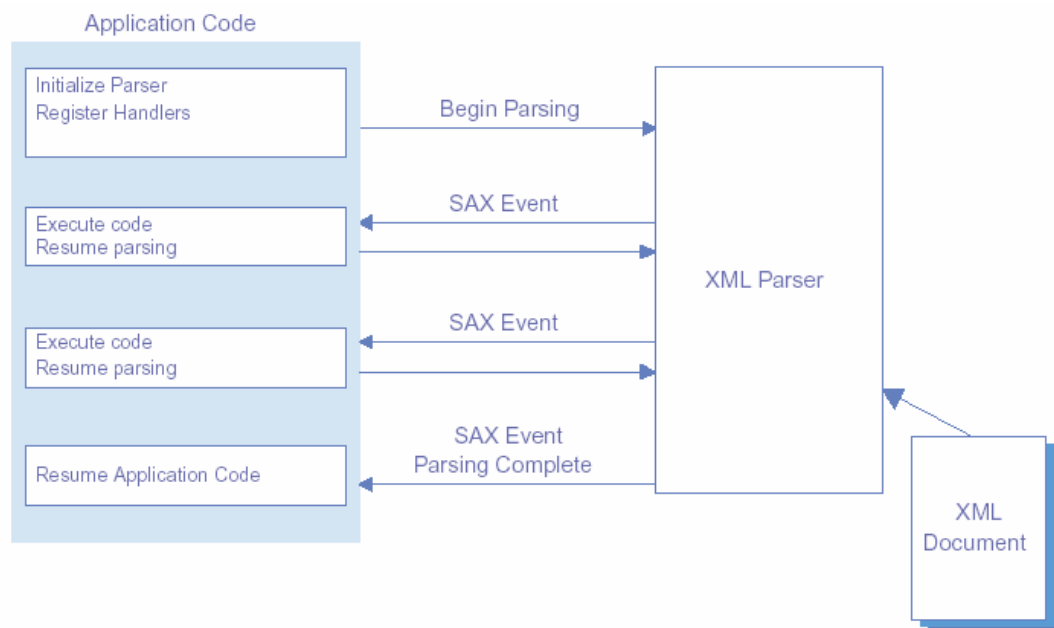
قبل از اینکه يك سند اعتبار سنجی و استفاده شود آن باید با یکی از نرم افزار های پارسر تجزیه شود. این پارسر ها کامپوننت نرم افزاری هستند که قابلیت خواندن مستندات xml را داشته و داده داخل این سند را برای برنامه کاربردی قابل دسترس میکنند.

پارسرهای مختلفی تا کنون توسعه داده شده اند مثل Xerces و Apache Software Foundation و پارسرهای از Oracle و IBM. شما برای اطلاعات بیشتر درباره این پارسرها می توانید از سایت <http://xml.apache.org> اطلاعات بگیرید. این دو ابزار به صورت open source بوده و به صورت گسترده ای استفاده میشوند.

SAX

اغلب پارسر ها در دو مد مختلف می توانند استفاده شوند و بر پایه نیازهای برنامه شما می باشد. مدل اول مدل رخداد گرا که آنرا با Simple API for XML(SAX) می نامیم. این پارسر داده منبع xml را خوانده و برنامه سرویس گیرنده را اگر در یک قسمت مشخص از سند با اشکال مواجه شود از آن مطلع می کند. برای مثال یک رخداد SAX این است که وقتی انتهای سند xml مشخص نباشد یا یک تگ بسته نشده باشد.

برای استفاده از SAX می بایست دستگر رخداد(event handler) را برای پارسر در زمانی که سند xml را پارس میکند تکمیل کنید. این دستگیر رخداد اغلب اوقات یک ماشین حالت(یک نمودار که فقط با گرفتن ورودی های خاص مراحل بعدی را شروع می کند) است که داده ها را جمع کرده و همچنین در شروع پارس کردن مجموع دادهای زیر مستندات(subdocument) را به صورت مستقل از دیگری دستگیر و کنترل می کند. استفاده از SAX در شکل زیر به نمایش در آمده که در آن از متدهای سریع پارس کردن و مختص برای مستندات بزرگی که در یک زمان به حافظه کپی نمی شوند می باشد.

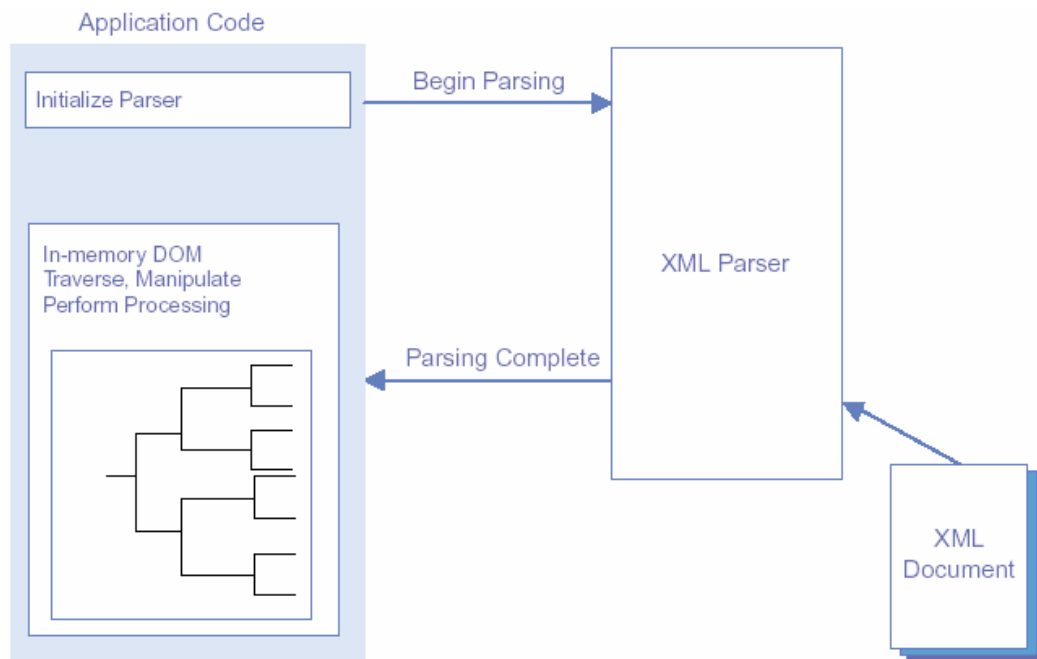


یک از اشکالهای SAX این است که قادر به جستجوی رو به جلو در هنگام پارس کردن اطلاعات نمی باشد. دستگیر SAX همانند یک ماشین حالت است که می تواند فقط بروی یک قسمت از سند که همانکون پارس شده کار کند. عیب دیگر نقص در پیش بینی رابطه بین گره های داخل سند است.

DOM

مدل دیگر پارس کردن مستندات xml به جای SAX مدل Document Object Model(DOM) می باشد. پارسر به طور کامل سورس سند xml را خوانده و یک ارائه شبیه به ساختار درختی را در حافظه می سازد. در هنگام استفاده DOM یک اشاره گر

به کل سند به نرم افزار استفاده کننده از آن برمي گرداند. برنامه کاربردي با اين اشاره گر سند xml را دستکاري ، چيدن دوباره گره ها ، اضافه و حذف کردن محتوا را انجام مي دهد. استفاده از DOM در شکل زیر نشان داده شده.



استفاده از DOM سادر تر از SAX مي باشد و منابع کمتری از آن مي خواهد. مي توان DOM را بصورت موثر با ساختارهاي داده xml كوچك در وضعيتي كه سرعت براي برنامه کاربري اهميت زيادي ندارد استفاده كرد. به دليل اين كه كل ساختار را در حافظه يکجا پارس مي کند سرعت کمتری دارد. در تصميم گيري براي اينکه از کدام روش براي پارس کردن استفاده کنید بايد به نياز هاي نرم افزار ، سرعت ، دستکاري داده ها و اندازه سند xml توجه داشته باشید.

تکنولوژی تبدیل XML

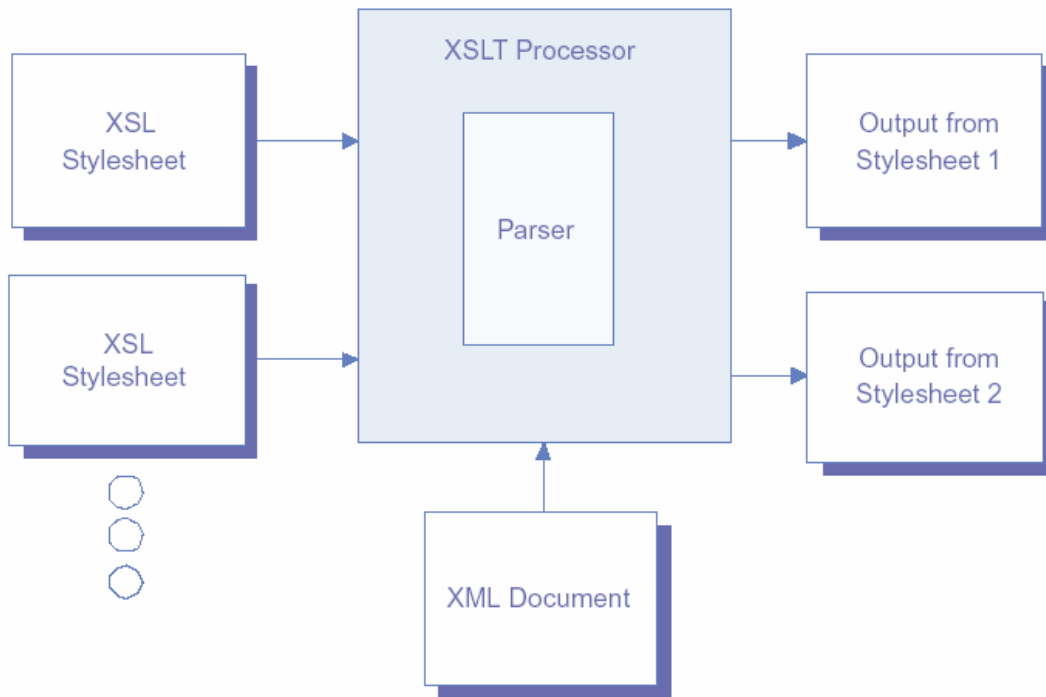
يکي از مزيتهاي کليدي xml در مقايسه با ساير فرمتهاي داده كه در معنای کلي قادر به برگرداندن يك مجموعه داده هاي xml از يك شکل به شکل ديگر مي باشد. تکنولوژی اين ترجمه را قابل انجام مي کند eXtensible Stylesheet Language for Transformations(XSLT) نام دارد.

XSLT

در حالت ساده xslt يك چارچوب (framework) براي ترجمه ساختار سند xml مي باشد. xslt يك سند xml ورودی را با يك xslt stylesheet براي ايجاد سند خروجي تركيب مي کند. Xslt stylesheet مجموعه دستورات تبديل براي برگرداندن يك سند xml به سند خروجي مورد نظر است.

شكل زیر فرآيند xslt را نشان ميدهد.

اجرای تبديل به کمک xslt نيازمند يك پردازشگر xslt است. معروف ترين موتور xslt براي جاوا Apache Software Foundation Xalan مي باشد و به صورت open source در دسترس است.



يك پردازنده xslt يك درخت سند xml را با اختصاص الگوهاي داخل سند خروجي به كمك XSLt stylesheet تطابق داده و آنها را ترجمه مي كند. براي مثال فرض كنيد كه شما نياز به ترجمه سند product catalog xml به يك فايل html براي تحويل دادن کاربرد آن محصول داريد. اين كار شامل گنجاندن داده هاي محصولات اختصاصي در سند xml داخل html است. در زير xslt كه اين وظيفه را انجام مي دهد ملاحظه مي كنيد.

```

<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:template match="/">
<html>
  <head><title>My Products</title></head>
  <body>
    <h1>Products Currently For Sale in the U.S.</h1>
    <xsl:for-each select="//product">
      <xsl:value-of select="@name"/> : $
      <xsl:value-of select="./price[@unit='USD']"/> USD
    </xsl:for-each>
  </body>
</html>
</xsl:template>
  
```

</xsl:stylesheet>

ویژه گی match در يك عبارت XPath به معني عنصر ریشه XML مي باشد. پس اين الگو براي اجرا در سند كامل است. در قسمت پائين كد، هر عنصر محصول در داخل سند منبع xml با مشخصاتش چاپ خواهد شد. ابتدا نام محصول سپس قيمت به همراه علامت دلار و در آخر رشته USD.

پردازنده xslt قادر به تغيير در اصطلاح ارزيابي مشخصات است و بيشتر به راه هاي پيش كامپايل براي ترجمه بر اشاره دارد. اين كار اجازه مي دهد كه موتور xslt روز به روز سريع تر شود. جزئيات بيشتر درباره xslt در آدرس <http://www.w3.org/Style/XSL> موجود است.

تبدیل دودوي برای XML

قابليت هاي xslt محدود به تبديلهاي متني نمي باشد. بعضي از مواقع لازم است كه داده هاي متني به شكل دودوي (binary) تبديل شوند. يك مثال معمولي تبديل داده هاي تجاري به فرمت PDF براي نمايش مي باشد. براي همين دليل XSLT 1.0 يك سري ويژه گيها براي تنظيم اشياء فرمت (formatting object) تعين کرده. اشياء فرمت دستوراتي هستند كه يك طرح بندي از ارائه اطلاعات مي دهند. اشياء فرمت در چاپ و طراحي چنדרسانه اي بيشتر مفيد خواهند بود. برخي كتابخانه هاي جاوا هم اكنون براي تبديل انواع معمولي در دسترس هستند.

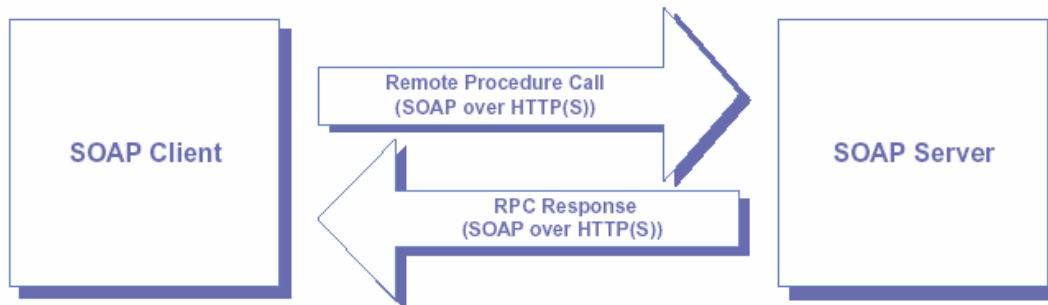
تكنولوژي هاي ارسال پيام

چندين تكنولوژي براي انتقال داده ها با ساختار xml بين برنامه ها و سيستم هاي enterprise هم اكنون در حال توسعه هستند. پتانسيل عظيم xml، آن را به عنوان پل در ميان فاصله بين فرمت داده ها و پرتكولهاي پيام رساني تبديل کرده. استفاده از xml شركتها را قادر مي سازد كه استانداردهاي رابط را براي سيستمها و سرويسها فراهم آورد و صاحبان صنايع را قادر به برقراري ارتباط با كمترين هزينه توسعه مي كند. در اين بخش يك شرح خلاصه از اين تكنولوژي ها براي شما بيان خواهيم كرد.

SOAP

بيشتر پيشرفتها در ارتباط به كمك xml، حوال اين تكنولوژي يعني Simple Object Access Protocol (SOAP) بوده. پرتكول Soap يك مشخصه پيام رساني است كه داده هاي رمزگذاري و قواعد بسته بندي را براي ارتباط بر پايه xml را شرح ميدهد. پرتكول soap مشخص مي كند كه چگونه يك پيام بايد ايجاد، بسته بندي و به مقصد ارسال شود. اين ارتباط شامل يك به هم بستن يا نگاشت به پرتكول HTTP مي باشد و به اين معني مي باشد كه پيامهاي soap قابل انتقال با سيستم وب جهاني كنوني است. بيشتر پايه soap بر اساس XML-RPC مي باشد اين قابليت شرح براي خصوصيت فراخواني روال راه دور كه به كمك xml اجرا مي شود است. پرتكول soap هر دو مدل انتقال پيام بصورت همگام و ناهمگام بين سرويس دهنده و سرويس گيرنده را پشتيباني مي كند. در پيام رساني به صورت همگام سرويس گيرنده يك درخواست را براي يك روال به همراه پارامترها در قالب xml به سرويس دهنده soap و به فرم پيام soap request ارسال مي كند. سرويس دهنده نيز داده هاي درخواست سرويس گيرنده را به فرم

soap response message برای آن ارسال می کند. این اعمال در شکل زیر نمایش داده شده.



پیام‌رسانی‌ها همگام‌طور کامل توسط soap پشتیبانی شده. این در وضعیتی مناسب می‌باشد که در آنجا قصد به روزرسانی اطلاعات را با ارسال و دریافت آنها در زمانی که تغییر اتفاق می‌افتد داریم. رخداد به روزرسانی اغلب نیاز به پاسخ سریع ندارد و یک پیام پاسخ به همان نقطه در آینده ارسال خواهد شد. این پاسخ ممکن است تصدیق برای دریافت پیام باشد و گزارشی از وضعیت پردازش در سمت گیرنده پیام باشد. علاوه بر به روزرسانی سرویسی گیرنده می‌تواند درخواست برای روال را داده و در اتصال بعدی که زمان آن مشخص نیست پاسخ را دریافت کند. در شکل زیر پیام‌رسانی به صورت ناهمگام نشان داده شده.



سرویس‌های وب

رابطه نزدیکی بین توسعه با soap و مفاهیم سرویس‌های وب وجود دارد. سرویس‌های وب یک عبارت با کاربردهای مختلف در استاندارد سازی برای معماری سرویس‌های تجاری توزیع در اینترنت است. سرویس‌های وب بر روی سرویس دهنده و گیرنده soap برای انتقال پیام‌های inter-enterprise تکیه کرده‌اند. دو موضوع پیام‌رسانی با xml و سرویس‌های وب در کنار هم در حد کافی پیچیده می‌باشند و در این بخش ما فقط پایه‌ای از سرویس‌های وب و تکنولوژی‌های وابسته را شرح خواهیم داد.

ادامه پیشرفت سرویس‌های وب تا به این جا رسیده که یک استاندارد برای ثبت (register) و پیدا کردن سرویس‌های وب با استفاده از یابنده سرویس‌های توزیع شده یا موتور جستجو تعریف شده. این یابنده‌های سرویس، از xml برای شرح دادن سرویس‌های وب و شرکت‌هایی که آنها را ارائه می‌دهند استفاده می‌کنند.

این سرویس هم اکنون با Universal Description, Discovery, and Integration (UDDI) نام گذاری شده و فروشنده گانی مثل میکروسافت و IBM نمونه های از آن را ایجاد کرده اند.

UDDI

یک کنسرسیوم از شرکتهای بزرگ با هم برای ایجاد یک مجموعه استاندارد حول محور ثبت و فرآیند کشف سرویسهای وب با هم همکاری کردند و نتیجه به آنجا رسید که UDDI به عنوان استاندارد پذیرفته شد. هدف UDDI ایجاد قابلیت ثبت online و جستجوی سرویس های وب به کمک یابنده های در دسترس است. همانند عمل Domain Name System (DNS) که آدرس url یک سرویس وب را گرفته و آدرس IP کامپیوتری را که این سرویس روی آن می باشد بر می گرداند UDDI عمل می کند. رجیستری سرویس (نام سرویس های که در آن سرویس دهنده وجود دارد) به صفحات سبز اشاره دارند و در یک الگوی xml تعریف شده اند.

صفحات سبز اتحادیه های (syndicate) در سراسر سایتهای عامل هستند. هر سایت عامل اطلاعات کلی درباره سرویس ها در سطوح مختلف می دهند. این اطلاعات مثل فرا داده (metadata) یا داده در مورد داده بوده و با نام tModel شنا ساي می شوند.

یکی از چالشها در ثبت کردن سرویس وب تصمیم گیری برای رده بندی آن می باشد. یک لیست راکد الفبا پیدا کردن سرویسها با نوعهای ویژه را ناممکن میکند. بنابراین UDDI اجازه می دهد که رده بندی سرویسها بر اساس نواحی جغرافیایی و کدهای استاندارد صنعتی مثل NAICS, UN/SPC صورت بگیرد. برای اطلاعات بیشتر راجع UDDI می توانید به آدرس <http://www.uddi.org> مراجعه کنید.

WSDL

ایجاد کننده گان UDDI درک کردند که نیاز به یک استاندارد برای شرح دادن سرویسهای وب در داخل رجیستری دارند و آنها Web Service Description Language (WSDL) را ایجاد کردند. استاندارد WSDL از یک زبان xml که برای شرح کلی سرویس وب می باشد استفاده می کند. اطلاعات داخل هر شرح دهنده شامل یک آدرس شبکه، پروتکل و مجموعه اعمال پشتیبانی شده است.

تکنولوژی ذخیره و بازیابی داده ها

ذخیره و بازیابی داده ها در فرمت xml به یکی از کاربردهای xml تبدیل شده. نیاز ما به ذخیره و بازیابی داده ها در فرمت xml در نتیجه ایجاد تعداد بیشماري از ویژگی های آن که نزدیک به هم می باشد. در این بخش با خلاصه ای از این ویژگی ها آشنا می شوید.

XPath

زبان XPath برای آدرس دهی ساختارهای xml بوده که در سایر استانداردهای گوناگون xml مثل xslt، XPointer، XQuery استفاده می شود. آن تعریف کننده نحو (syntax) برای ایجاد عبارات، آنهایی که در یک سند xml ارزیابی شده اند استفاده می شود. برای مثال علامت (/) یک عبارت ساده XPath است و همان طور که قبلاً گفتیم این عبارت گره ریشه در یک سند xml می باشد. عبارتهای XPath می توانند یک

مجموعه گره، مقدار منطقی، عدد یا رشته را نشان دهند. آنها از عنصر ریشه یا از محل خاص در آن سند شروع کرده و اکثر انواع عبارتهای XPath عبارتهای مسیر یک محل در یک مجموعه گره می باشند. برای product catalog مثال XPath زیر تمام گره های محصول در کاتالوگ را نشان می دهد.

/product

در داخل XPath یک مجموعه از توابع وجود دارد که شما را قادر می سازد که عبارات بسیار پیچیده را توسعه دهید.

XPointer

XPointer یک سری خصوصیات زبان خاص است که بر روی XPath طراحی شده است. عبارات XPointer فقط برای یک مجموعه عبارات نبوده و برای مشخص کردن یک محدوده یا وضعیت در داخل یک مجموعه گره ها که در یک شرط خاص صادق هستند. توابع XPointer دارای متدهای قدرتمند برای جستجو در داخل ساختارهای xml می باشند. برای مثال گره های زیر را در نظر بگیرید.

<desc>This chapter provides an overview of the J2EE echnologies.</desc>

<desc>This chapter provides an overview of the XML landscape.</desc>

<desc>This chapter is an introduction to distributed systems.</desc>

یک عبارت ساده XPointer که روی این گره ها عمل می کند به شکل زیر می باشد.

xpointer(string-range(//desc, 'overview'))

این عبارت تمام گره ها با نام desc که شامل رشته overview می باشند را بر می گرداند. عبارتهای XPointer در اشکال مختلف می توانند شکل بگیرند و سریع پیچیده شوند.

XInclude

مکانیزم XInclude برای زمینه کردن یک سند xml در داخل سایر سندهای xml می باشد. این به ما اجازه می دهد که وابستگی های پیچیده را بین سند های مختلف xml ایجاد کنیم. برای ایجاد از تگ <include> استفاده کرده و مکان مشخص را برای سند مورد نظر تعیین کنید. میتوانید ذکر کنید که آیا آن قسمت می تواند توسط پارسر، پارس شود یا خیر.

تگ <include> در هر کجا از سند xml می تواند استفاده شود و اشاره به یک سند کامل xml یا یک XPointer برای قسمتی از یک سند داشته باشد. استفاده از XInclude با XPointer امکان شامل کردن داده های خاص xml و گرفتن آنها از فایل های مختلف را فراهم می کند.

اضافه کردن خط زیر به یک سند xml یک مجموعه گره را از فایل خارجی afile.xml به سند مورد نظر در مکان نوشتن آن زمینه خواهد کرد.

<xi:include href="afile.xml#xpointer(XPath expression)" parse="xml" />

XLink

تکنولوژی XLink برا آسان کردن ارتباط بین منابع مختلف در سندهای xml مورد استفاده قرار می گیرد. دلیل ایجاد آن، ارتباط منابع xml همانند یک مکانیزم آنچه که ابر لینکها(hyperlinks) در html ارائه می کردند ضروی شمرده می شد. تکنولوژی XLink در نوع پیمایش را در خود دارد ساده و توسعه یافته. XLink های ساده

قواعدي شبیه به ابر لینکها در html دارند و XLink هاي توسعه یافته قابلیتهاي اضافي علاوه بر قابلیتهاي XLink ساده دارند. قابلیت XLink این امکان را به ما مي دهد که وابستگی هاي رابطهاي بین اشیاء را به وجود آوریم. در مثال زیر يك رابطه بین سفارش و مشتري ایجاد مي کنیم. متن سند xml زیر يك مشتري را نشان مي دهد:

```
<customer id="0059">
  <name>ABC Company</name>
  <employees>1000-1500</employees>
</customer>
```

سند زیر يك لینک به سند مشتري در خود دارد:

```
<orders>
  <order xlink:type="simple"
    href="customers.xml#//customers/customer/@id[.='0059']"
    title="Customer" show="new">
    <number>12345</number>
    <amount>$500</amount>
  </order>
</orders>
```

XBase

مکانیزم XBase یا XML Base برای مشخص کردن پایه برای شناساي کننده منابع یکسان (Uniform Resource Identifier URI) در سندهاي xml همانند زیر دنباله از ارجاعات که به چیزی اشاره مي کنند استفاده مي شود. استفاده از آن ساده بوده و به طور فراوان در دسترس است و به شما اجازه مي دهد Xlink ها را با طول قابل قبول استفاده کنید. در زیر يك URI پایه با استفاده از XBase شرح داده شده. هر تعداد ارجاع URI در داخل عنصر کاتالوگ مي تواند به راحتی با استفاده از <http://www.manning.com/books> به آن ارجاع کند و به عنوان پایه برای ارجاعات مي باشد.

```
<catalog xml:base=http://www.manning.com/books/>
  .....
  .....
</catalog>
```

زبانهاي پرس وجو

زمانی که داده هاي ذخیره شده در سند xml افزایش یافتند این نکته اهمیت خواهد داشت که چندین زبان پرس وجوي مختلف برای سندهاي xml توسعه داده شده اند. يك از موثرترین این زبانها (XQL) XML Query Language (XQL) مي باشد. XQL يك زبان برای پرس وجو از ساختار داده اي xml بوده و اشتراك گسترده اي با XPath

دارد. بعد از استفاده از پرس وجوهای XQL آن يك مجموعه از گره ها را از يك يا چند سند xml برمي گرداند.

ساير زبانهاي پرسوجو XML QL , Quilt مي باشند و W3C در حال كار بروي استاندارد كردن اين زبانها و تركيب ويژگي هاي هر کدام با هم است. نتيجه به دست آمده از اين استاندارد XQuery بود. زبان XQuery از احاطه نحوي شبیه به زبان SQL بوده و شامل کلمات کلیدی RETURN, WHERE, LET, FOR مي باشد. عبارت XQuery زیر تمام گره هاي محصول را از afile.xml برميگرداند.

```
document( "afile.xml" )//product
```

عبارت کمي پیچیده زیر گره warranty براي هر محصول را انتخاب مي کند.

```
FOR $product in //product
RETURN $product/warranty
```

براي گرفتن اطلاعات بیشتر در مورد آنچه که در اين بخش بحث شد مي توانيد از آدرس <http://www.w3c.org/TR/> کمک بگيريد.

تكنولوژي ذخيره داده ها

ذخيره داده هاي xml در سيستم فايلها (filesystem) هم اکنون بسيار مورد استفاده قرار ميگيرد. اما ذخيره xml به صورت متني و با فرمت پارس نشده قابليت استفاده را پائين آورده و غير موثر مي باشد. مسندات ايستا در هر زمان که آنها مورد نیاز هستند بايد پارس شوند و يك مکانيزم مياني براي ذخيره فايلهاي متني Persistent Document Object Model (PDOM) مي باشد. در W3C روش PDOM مشخصات W3C DOM را کاملتر کرده و تفاوت در اين است که سند پارس شده xml را در فرمت باينري در روي سيستم فايلها نگهداري مي کند و نیاز نيست هر بار آن را پارس کنيم.

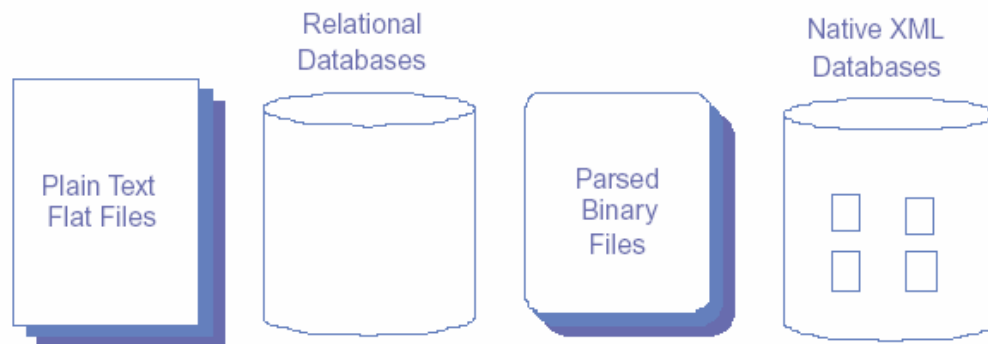
سندهاي PDOM ممکن است از DOM موجود يا از جريان ورودي xml ايجاد شوند و نیاز به در حافظه بودن کامل در هر زمان ندارند. اين مزيت در جاهاي که مستندات xml بزرگ مي باشند قابل استفاده است.

PDOM اعمال استاندارد مورد انتظار از يك کامپوننت ذخيره داده را در خود دارد مثل پرسوجوها با XQL ، حذف کردنها، فشردن سازي و کش کردن (caching). روش ديگر ذخيره در سيستم فايلهاي ايستا استفاده از Native-XML Database است. نرم افزار هاي مثل Tamino بطور ويژه براي xml طراحي شده اند. اين نرم افزار مستندات xml را در فرمت مخصوص به خود (native) نگهداري مي کند، اما بانکهاي اطلاعاتي رابطه اي سلسه مراتب سندهاي xml را در داخل جداول ذخيره مي کنند.

اين کار کاراي Tamino را وقتي که با مستندات xml کار ميکند را بالا مي برد. بنا بر اين ظهور فروشنده گان بانک هاي اطلاعاتي Native-XML، فروشنده گان بانکهاي اطلاعاتي تجاري مثل IBM , Oracle که قصد دست کشيدن از تجارت ذخيره داده ها را ندارند را به تکاپو وا داشت.

آنها با ايجاد قسمتهاي بروي محصولاتشان همکاري با داده اي xml را بالا برده و آنها را قادر به اجراي قابليت هاي پرسوجو کردند. اين مزيت براي بسياري از شرکتهای که محصولاتشان بروي بانکهاي اطلاعاتي رابطهاي بود بسيار مفيد ولقع گرديد.

در شکل زیر خلاصه ای از چهار گزینه برای ذخیره داده های xml نشان می دهد.



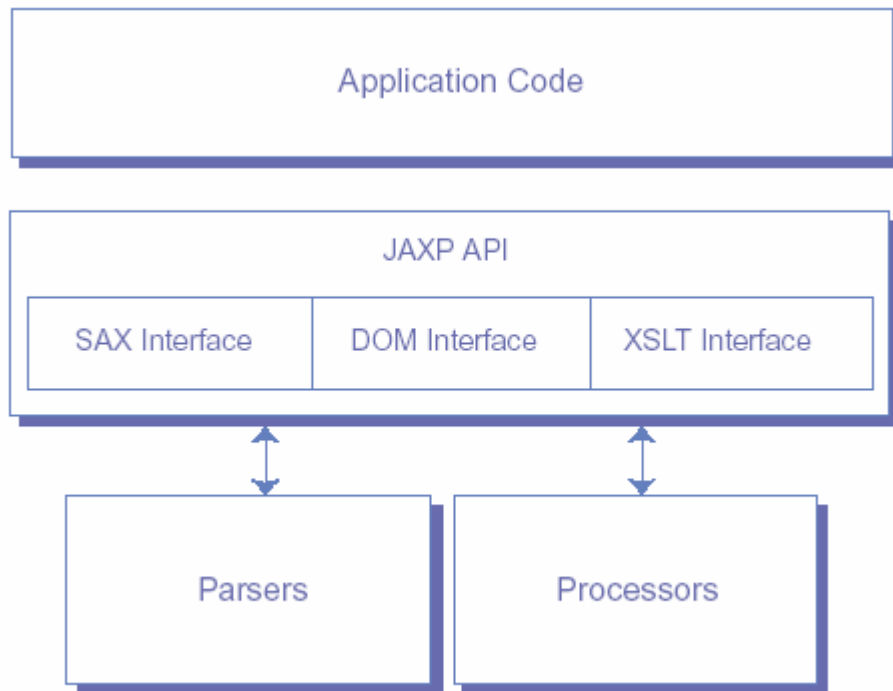
API های جاوا برای XML

کمیتة توسعه جاوا در زمینه تکنولوژیهای xml که در قسمت قبل گفته شد فعالیت زیادی انجام داده و اغلب جاوا اولین زبانی است که تکنولوژی های جدید xml را توسعه می دهد. هر چند که گروه های زیادی در زمانهای مختلف توابع مختلفی برای جاوا طراحی کرده اند و در برخی مواقع با هم ناسازگار می باشند. شرکت سان با کمیتة فرآیندهای جاوا (JCP) java community process استاندارد Java API for XML (JAX) را برای توسعه برنامه های کاربردی ایجاد کرد. در جدول زیر قسمتهای مختلف این مجموعه API را ملاحظه می کنید.

Java API for XML	JAX acronym	Functional description
Java API for XML parsing	JAXP	Provides implementation-neutral access to XML parsers and XSLT processors.
Java Document Object Model	JDOM	Provides a Java-centric, object-oriented implementation of the DOM framework.
Java API for XML binding	JAXB	Provides a persistent XML mapping for Java object storage as XML.
Long Term Java-Beans Persistence		Similar to JAXB, provides XML serialization for JavaBean components.
Java API for XML messaging	JAXM	Enables the use of SOAP messaging in Java applications, using resource factories in a manner similar to the Java Messaging Service (JMS).
JAX-RPC	JAX-RPC	An XML-RPC implementation API for Java. Similar to JAXM.
Java API for XML repositories	JAXR	Provides implementation-neutral access to XML repositories like ebXML and UDDI.

JAXP

Jaxp يك سري API عمومي كامل براي ايجاد و استفاده از SAX, DOM و XSLT API در جاوا است. برنامه شما مي بايد از jaxp استفاده كنيد و استفاده از API هاي كه توسط فروشنده گان خاص كه با سليقه خود ايجاد و پشتيباني شده اند خودداري كنيد. اين API ها روزه روز سزيتر و بهتر مي شوند. به منظور ارتقا با تعويض فايلهاي (Java Archive)JAR مي توتنيد API هاي جديد را استفاده كنيد. با اين قابليت به هدف اصلي توسعه برنامه هاي توزيع شده يعني تطابق دست خواهيد يافت. در شكل زير معماري jaxp را مشاهده مي كنيد.



همان طور كه در معماري مشاهده مي كنيد قابليت انعطاف با جدا كردن كدهاي برنامه کاربري از زير لايه XML API است. شما مي توانيد با زير لايه SAX يا DOM استراتژي نحوه پارس كردن سند xml خود را مشخص كنيد و همچنين براي تبديل xml از XSLT محصول فروشنده گان مختلف استفاده كنيد. Jaxp شامل چهار بسته مي باشد كه در جدول زير آنها مشخص شده.

Package	Description
<code>javax.xml.parsers</code>	Provides a common interface to DOM and SAX parsers.
<code>javax.xml.transform</code>	Provides a common interface to XSLT processors.
<code>org.xml.sax</code>	The generic SAX API for Java
<code>org.w3c.dom</code>	The generic DOM API for Java

دو بسته از اين API ها در `javax.xml` قرار دارد. بسته `javax.xml.parsers` شامل كلاسها و رابطهاي مورد نياز براي پارس كردن سند xml است. بسته `javax.xml.transform` تعريف كننده رابط به پردازنده XSLT است.

پیکره بندی JAXP

- برای استفاده از jaxp برای پارس کردن شما به JAXP XML Parser نیاز دارید. همچنین برای پردازش xslt به XSLT Engine نیاز دارید. در اولین دستیابی به کلاس پارسر jaxp در کدتان گامهای زیر را framework جاوا برای خود مقدار دهی خواهد کرد.
- آن ابتدا چک می کند که آیا خصیصه (property) سیستم یعنی `javax.xml.parsers.DocumentBuilderFactory` یا `javax.xml.parsers.SAXParserFactory` تنظیم شده می باشند یا نه. اگر نیاز به تبدیل xslt دارید خصیصه سیستمی `javax.xml.transform.TransformerFactory` را نیز چک می کند.
 - اگر این خصیصه های به طور صریح تنظیم نشده باشند framework به دنبال فایل `jaxp.properties` در دایرکتوری کتابخانه های JRE خواهد گشت. در زیر یک مثال از محتوای این فایل نشان داده شده.

<pre>javax.xml.parsers.DocumentBuilderFactory= org.apache.crimson.jaxp.DocumentBuilderFactoryImpl javax.xml.parsers.SAXParserFactory= org.apache.crimson.jaxp.SAXParserFactoryImpl javax.xml.transform.TransformerFactory= org.apache.xalan.processor.TransformerFactoryImpl</pre>	Sets DOM builder, SAX parser, and XSLT processor implementation classes
--	--

توجه داشته باشید که مثال بالا در چند خط نمایش داده شده و در فایلهای `jaxp.properties` هر دستور در یک خط بوده و بین علامت (=) نمی تواند فاصله وجود داشته باشد.

- اگر فایل `jaxp.properties` که نام کلاس دلخواه برای پارسر است پیدا نشود framework به صورت پیش فرز پارسر Crimson را فراخوانی خواهد کرد و از Xalan برای xslt استفاده خواهد کرد.

برای اینکه خصیصه های مربوط به پارسر و پردازنده را برای نگاشت به کلاس آنها تنظیم کنید یک راه دیگر ساده تر برای پیکره بندی گذاشتن فایل JAR پارسر و پردازنده دلخواه در مسیر کلاسهای برنامه است. اگر در مسیر کلاسهای برنامه دو نوع JAXP API وجود داشته باشد شما باید یکی از آنها را در فایل `jaxp.properties` گفته شده در بالا مشخص کنید.

هم اکنون jaxp به قسمتی از مشخصه های J2EE تبدیل گشته و به این معنی می باشد که شرکتهای پشتیبانی کننده J2EE باید از آن پشتیبانی کنند. این استفاده از jaxp را ساده تر کرده و استفاده مستقیم از XSLT, SAX, DOM فراهم می کند.

استفاده از JAXP با SAX

کلاسهای کلیدی jaxp برای استفاده با SAX در جدول زیر به نمایش در آمده. پیش از شرح استفاده از SAX ما می بایست از توضیحات جزئی در باره پارس کردن به فرم SAX برای لحظاتی گذر کنیم. برای استفاده از SAX بدون jaxp شما می باید همیشه یک یا چند دستگیر رخداد برای پارسر تعریف کنید.

JAXP class or interface	Description
<code>javax.xml.parsers.SAXParserFactory</code>	Locates a <code>SAXParserFactory</code> implementation class and instantiates it. The implementation class in turn provides <code>SAXParser</code> implementations for use by your application code.
<code>javax.xml.parsers.SAXParser</code>	Interface to the underlying SAX parser.
<code>javax.xml.parsers.SAXReader</code>	A class wrapped by the <code>SAXParser</code> that interacts with your SAX event handler(s). It can be obtained from the <code>SAXParser</code> and configured before parsing when necessary.
<code>org.xml.sax.helpers.DefaultHandler</code>	A utility class that implements all the SAX event handler interfaces. You can subclass this class to get easy access to all possible SAX events and then override the specific methods in which you have interest.

توجه کنید که یک دستگیر رخداد SAX یک کامپوننت می باشد که خود را برای فراخوانی های پارسر زمانی که یک رخداد اتفاق می افتد رجیستر می کند. SAX API چهار دستگیر رخداد اصلی را تعریف می کند که در رابط های `EntityResolver`, `DTDHandler`, `ContentHandler`, `ErrorHandler` از بسته `org.xml.sax` کیسوله سازی شده اند. رابط `ContentHandler` رابط اصلی است که اغلب برنامه ها باید آنرا کامل کنند. این رابط شامل متدهای رخداد `startDocument`, `startElement`, `endElement`, `endDocument` می باشد. برنامه شما باید این متدهای رخداد رابط ضروری را که لازم دارد کامل کند.

سایر دستگیرهای موجود در SAX برای کارهای جانبی در پارس کردن سند xml استفاده می شوند. رابط `EntityResolver` ما قادر می سازد که نگاشت ارجاعات به منابع خارجی مثل بانکهای اطلاعاتی یا URL ها داشته باشیم. رابط `ErrorHandler` برای دستگیر کردن خطاهای ویژه در پردازش از `SAXExceptions` استفاده می کند. رابط `DTDHandler` برای گرفتن اطلاعات درباره اعتبار سنجی سند از سندهای DTD استفاده می شود.

همچنین SAX یک کلاس برای راحتی توسعه دهنده گان به نام `org.xml.helpers.DefaultHandler` ارائه می دهد که به کمک آن برنامه شما به تمامی رخدادهای کامل شده پیش فرز SAX دسترسی پیدا میکند. حال که ما اطلاعاتی در باره نحوه کار SAX به دست آوردیم نوبت گذاشتن `jaxp` برای کار با آن می باشد. برای مثال اجازه دهید که سند xml گذشته یعنی `product catalog` را به کمک رخدادهای SAX و `jaxp` بخوانیم. برای کوتاه و مناسب کردن این مثال ما فقط کلاس دستگیر رخداد `endElement` را تعریف کرده ایم. هر زمان که یک عنصر محصول به طور کامل از سند xml با پارسر SAX خوانده شد ما یک پیغام که به آن اشاره دارد چاپ خواهیم کرد. کد این مثال در زیر نوشته شده.

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
public class ProductEventHandler
    extends DefaultHandler {
//Extends this class to only handle the endElement event
```

```
// other event handlers could go here
public void endElement( String namespaceURI,
                        String localName,
                        String qName,
                        Attributes atts )
    throws SAXException {
    // make sure it was a product node
    if (localName.equals("product"))
        System.out.println( A product was read from the catalog.);
}
}
```

حال که ما یک دستگر رخداد تعریف کردیم ، ما می توانیم یک پارسر کامل SAX را به کمک jaxp در کد برنامه شما اجرا کنیم و دستگیرها را برای اعمال کامل کنیم. متد دستگیر endElement زمانی که سند مثال ما حتی یک گره محصول در آن باشد فراخوانی خواهد شد. در زیر مثالی که از JAXP SAX استفاده می کند را مشاهده می کنید

```
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import java.io.File;
public class JAXPandSAX {
    public static void main(String[] args) {
        //Instantiates our event handler
        ProductEventHandler handler
            = new ProductEventHandler();
        try {
            SAXParserFactory factory
                = SAXParserFactory.newInstance();
            //Obtains a SAXParser via JAXP
            SAXParser parser
                = factory.newSAXParser();
            File ourExample
                = new File("product-catalog.xml");
            parser.parse( ourExample, handler);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

زمانی که کد مثال بر روی سند product catalog اجرا می شود شما خروجی زیر را خواهید دید.

Product read from the catalog.

زمانی که ما یک محصول را در داخل سند مشخص کرده باشیم این دستور فقط یک بار چاپ خواهد شد. اگر چند محصول در داخل سند باشد این دستور به تعداد آن محصولات چاپ خواهد شد.

استفاده از JAXP با DOM
استفاده از jaxp با DOM با استفاده آن در SAX کمی کمتر پیچیده می باشد. این به این دلیل می باشد که ما نیاز به به توسعه دستگیر رخدادهای و پاس کردن آنها به پارسر نداریم. در استفاده از DOM آن به طور کامل سند xml را به داخل حافظه به صورت درخت واره خواهد خواند. این به شما این اجازه را می دهد که در هر زمان سند را بطور کامل دستکاری و تغییر دهید. همچنین همانند SAX نیاز به برنامه نویسی ماشین حالتها در کد خود ندارید (در SAX چون حجم سند میتواندست بزرگ باشد و بطور کامل در حافظه قرار نداشت حالتها را مختلفی برای برنامه اتفاق می افتاد).
البته این ساده گی و راحتی منابع و هزینه بیشتری برای سیستمی که برنامه روی آن در حال اجرا می باشد مصرف خواهد کرد و سرعت کمی کاهش پیدا می کند. کلاسهای اصلی JAXP DOM در جدول زیر بطور خلاصه نشان داده شده.

JAXP class or interface	Description
<code>javax.xml.parsers.DocumentBuilderFactory</code>	Locates a <code>DocumentBuilderFactory</code> implementation class and instantiates it. The implementation class in turn provides <code>DocumentBuilder</code> implementations.
<code>javax.xml.parsers.DocumentBuilder</code>	Interface to the underlying DOM builder.

زمانی که سند product catalog طول کمی دارد مشکلی با DOM برای خواندن از آن نداریم. کد خواندن از این سند برای خواندن به کمک DOM در زیر نوشته شده و می توانید ببینید که گامهای کلی از گرفتن یک پارسر از JAXP و فراخوانی آنها همانند روش در SAX می باشد. فرق اساسی در دستگیرهای SAX میباشد.
توجه کنید که پارسر یک اشاره گر به DOM بعد از پارس کردن بر می گرداند. برای استفاده از DOM API از بسته `org.w3c.dom` استفاده می کنیم. شما می توانید سند به فرم DOM را با کمک کدها در حافظه پیمایش کنید و محصولات در کاتالوگ را مشاهده کنید.

```
import javax.xml.parsers.DocumentBuilderFactory;
//Imports the JAXP DOM classes
```

```

import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import java.io.File;
public class JAXPandDOM {
    public static void main(String[] args) {
        try {
            DocumentBuilderFactory factory
                = DocumentBuilderFactory.newInstance();
            //Obtains a DOMBuilder via JAXP
            DocumentBuilder builder
                = factory.newDocumentBuilder();
            File ourExample
                = new File("product-catalog.xml");
            //Parses the XML and builds a DOM tree
            Document document
                = builder.parse( ourExample );
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```

استفاده از JAXP با XSLT

JAXP از XSLT بطور مستقل از پیاده سازی (از سازنده گان مختلف) به همان روشی که با پارسرها برخورد می کرد پشتیبانی می کند. رابط JAXP به XSLT در بسته `javax.xml.transform` موجود است. کلاسها و رابطهای اصلی در جدول زیر مشخص شده. به علاوه این رابطهای سطح بالا JAXP شامل سه زیربسته برای پشتیبانی از استفاده `SAX`, `DOM`, `I/O Stream` به همراه XSLT است. این بسته ها در جدول بعدی به نمایش درآمده اند.

JAXP class or interface	Description
<code>javax.xml.transform.TransformerFactory</code>	Locates a <code>TransformerFactory</code> implementation class and instantiates it.
<code>javax.xml.transform.Transformer</code>	Interface to the underlying XSLT processor.
<code>javax.xml.transform.Source</code>	An interface representing an XML data source to be transformed by the Transformer.
<code>javax.xml.transform.Result</code>	An interface to the output of the Transformer after XSLT processing.

در قسمتهای قبلی ما روی فرآیند تبدیل صحبت کردیم و گفتیم که چگونه می توان `product catalog` را به یک سند `html` با کمک XSLT ترجمه کرد. حال به شرح چگونه گی فراخوانی فرآیند XSLT با کدهای جاوا به کمک JAXP خواهیم پرداخت. به منظور ساده گی و وضوح از کلاس کمکی `I/O Stream` در داخل بسته

javax.xml.transform.stream برای شی‌های Source و Result استفاده خواهیم کرد.

Package name	Description
javax.xml.transform.dom	Contains classes and interfaces for using XSLT with DOM input sources and results.
javax.xml.transform.sax	Contains classes and interfaces for using XSLT with SAX input sources and results.
javax.xml.transform.stream	Contains classes and interfaces for using XSLT with I/O input and output stream sources and results.

کد جاوا که کار تبدیل یک سند xml به html را دارد در زیر نوشته شده. برای کامپایل کردن آن شما باید فایل JAXP jar را در مسیر کلاس‌هایتان داشته باشید. همچنین فایل xml و xsl گفته شده در قسمت‌های قبل به نام‌های product-catalog.xml و product-catalog-to-html.xsl که stylesheet مربوط به ساختار تبدیل است باید را داشته باشید.

شما این فایل‌ها را با ویراستار دلخواه می‌توانید تایپ کرده یا از سایت <http://www.manning.com/gabrick> دانلود کنید. قبل از تست کردن این مثال نیاز به جای دادن یک XSLT engine JAXP-compliant همانند Xalan در مسیر کلاس‌هایتان دارید.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
//Imports the JAXP XSLT API
import java.io.File;
public class JAXPandXSLT {
    public static void main(String[] args) {
        File sourceFile
            = new File("product-catalog.xml");
        // Loads the XML and XSL files
        File xsltFile
            = new File("product-catalog-to-html.xsl");
        Source xmlSource = new StreamSource(sourceFile);
        //Creates I/O Stream sources and results
        Source xsltSource = new StreamSource(xsltFile);
        Result result = new StreamResult(System.out);
        //Returns an instance of TransformerFactory
        TransformerFactory factory
            = TransformerFactory.newInstance();
        try {
            //BFactory returns new Transformer
            Transformer transformer
```

```

    = factory.newTransformer(xsltSource);
//C Performs transformation
    transformer.transform(xmlSource, result);
    } catch (TransformerConfigurationException tce) {
        System.out.println("No JAXP-compliant XSLT processor
found.");
    } catch (TransformerException te) {
        System.out.println("Error while transforming document:");
        te.printStackTrace();
    }
}
}
}

```

شي TransformerFactory مشخصات ویژه تبدیل را کامل می کند. دقت داشته باشید که قواعد تبدیل در فایل XSLT stylesheet به کارخانه (مبدلی که کار واقعی تبدیل را انجام میدهد) برای ایجاد شی تبدیل کننده پاس شده اند. متد Transform کار تبدیل XSLT را انجام می دهد و نتایج به به شی Result جریان داده می شوند. در این مثال جریانی که نتایج خروجی را دریافت می کند خروجی استاندارد یعنی System.out است. در یک نگاه استفاده JAXP از XSLT برای تبدیل زیاد پیچیده نیست. این درست است که این تبدیل ساده می باشد، اما بسیاری از ویژگی های فرآیند XSLT با رابطهای Transformer و TransformerFactory وجود دارند که قابل پیکره بندی می باشند. همچنین شما می توانید دستگیرهای خطا برای خطاهای که ممکن است در حین تبدیل اتفاق بافتند ایجاد و ثبت کنید. برای اطلاعات بیشتر لیست مستندات کامل JAXP را نگاه کنید.

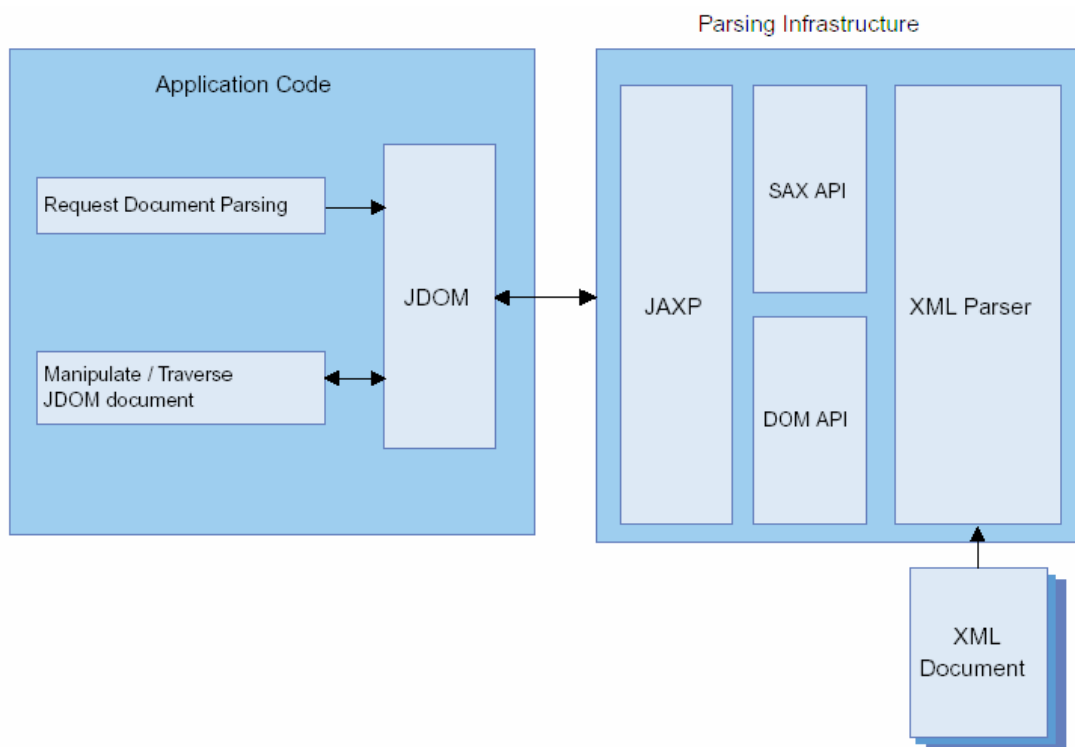
کمی احتیاط

استفاده از XSLT با JAXP بدون چالش نخواهد بود. یک مانع بزرگ برای استفاده گسترده از XSLT کارای در زمان اجرا است. اجرای یک تبدیل XSLT روی یک سند زمان بر بوده و منابع بسیاری را می گیرد. برخی پردازنده های XSLT همانند Xalan به شما اجازه می دهند که قواعد تبدیل داخل stylesheet را برای کارای بیشتری کامپایل کرده.

JDOM

اولین چیز که درباره اعضاء خانواده Java API for XML (JAX) فهمیده می شود حروف مخفف برای بسته های آن است. شما هم اکنون می توانید کد برای برنامه های کاربردی بنویسید که مستقل از پارسر می باشند. در اینجا دیگر API ها وجود دارند که این کار ها را بیشتر ساده می کنند. آنها را با Java Document Object Model (JDOM) نام گذاری کرده اند و هم اکنون توسط کمیته فنی جاوا به صورت رسمی پذیرفته شده.

این سری از API ها برای کار با ساختار های xml به صورت جاوا محور ایجاد شدند. بطور ویژه برای جاوا طراحی شده اند یک مدل شی قابل استفاده آسان می باشد. برای مثال JDOM از کلاسهای Java Collection مثل `java.util.list` برای مجموعه گره ها در کار با سندهای xml استفاده می کند. علاوه بر این کلاسهای JDOM ارتباط قوی با هم دارند و کامل می باشند در حالی که کلاسهای DOM به صورت انتزاعی می باشند. این قابلیت استفاده از آن را ساده کرده و وابستگی ها و تنظیمات مخصوص فروشنده گان را همانند JAXP را حذف می کند. استفاده از اکثر ورژنهای JDOM همانند استفاده از JAXP API می باشند. این به این معنی است که JDOM تغییر زیادی در معماری JAXP انجام نداده و فقط روی آن ایجاد شده و زمانی که یک شی xml ایجاد می کنند از API های قابل دسترس JAXP استفاده می کنند. در شکل زیر معماری JDOM نشان داده شده.



در جدول زیر کلاسهای مرکزی JDOM به نمایش درآمده. همان طور که شما می بینید آنها کاملاً با معنا نام گذاری شده ند. یک سند JDOM می توانند در حافظه ایجاد شود یا از یک جریان و URL ایجاد شود.

Class name	Description
org.jdom.Document	The primary interface to a JDOM document.
org.jdom.Element	An object representation of an XML node.
org.jdom.Attribute	An object representation of an XML node's attribute.
org.jdom.ProcessingInstruction	JDOM contains objects to represent special XML content, including application-specific processing instructions.
org.jdom.input.SAXBuilder	A JDOM builder that uses SAX.
org.jdom.input.DOMBuilder	A JDOM builder that uses DOM.
org.jdom.transform.Source	A JAXP XSLT Source for JDOM Documents. The JDOM is passed to the Transformer as a JAXP SAXSource.
org.jdom.transform.Result	A JAXP XSLT Result for JDOM Documents. Builds a JDOM from a JAXP SAXResult.

برای شرح سریع نحوه کار JDOM اجازه دهید که سند product catalog از حافظه خود پاک کرده و سپس ما آنرا از نوع بسازیم و به داخل یک فایل بنویسیم. برای این کار ابتدا ما به ساده گی درخت JDOM Element و یک JDOM Document از آن ایجاد می کنیم. کد برای این کار در قسمت زیر مشخص شده و زمانی که شما این کد را کامپایل و اجرا می کند بعد از آن می بایست یک ورژن خوش شکل (well-formatted) از سند xml که بعد از در فهرست جاری شما ایجاد شده و موجود باشد.

```
import org.jdom.*;
import org.jdom.output.XMLOutputter;
import java.io.FileOutputStream;
public class JDOMCatalogBuilder {
    public static void main(String[] args) {
        // construct the JDOM elements
        Element rootElement = new Element("product-catalog");
        Element productElement = new Element("product");
        //Creates element attributes
        productElement.addAttribute("sku", "123456");
        productElement.addAttribute("name", "The Product");
        Element en_US_descr = new Element("description");
        en_US_descr.addAttribute("locale", "en_US");
        // Adds text to the element
        en_US_descr.addContent("An excellent product.");
        Element es_MX_descr = new Element("description");
        es_MX_descr.addAttribute("locale", "es_MX");
        es_MX_descr.addContent("Un producto excelente.");
    }
}
```

```

Element en_US_price = new Element("price");
en_US_price.addAttribute("locale", "en_US");
en_US_price.addAttribute("unit", "USD");
en_US_price.addContent("99.95");
Element es_MX_price = new Element("price");
es_MX_price.addAttribute("locale", "es_MX");
es_MX_price.addAttribute("unit", "MXP");
es_MX_price.addContent("9999.95");
// arrange elements into a DOM tree
//Builds the DOM by adding one element as content to another
productElement.addContent(en_US_descr);
productElement.addContent(es_MX_descr);
productElement.addContent(en_US_price);
productElement.addContent(es_MX_price);
rootElement.addContent(productElement);
//Wraps root element and processing instructions
Document document = new Document(rootElement);
// output the DOM to "product-catalog.xml" file
//Indents element two spaces and uses newlines

XMLOutputter out = new XMLOutputter(" ", true);
try {
    FileOutputStream fos = new FileOutputStream("product-
catalog.xml");
    //Writes the JDOM representation to a file
    out.output(document, fos);
} catch (Exception e) {
    System.out.println("Exception while outputting JDOM:");
    e.printStackTrace();
}
}
}
}

```

در قسمتهای بعدی به طور گسترده از JDOM استفاده خواهیم کرد. JDOM با رابطهای JAXP کار کرده و آن را پشتیبانی می کند. برای اطلاعات بیشتر درباره JDOM می توانید به آدرس <http://www.jdom.org> مراجعه کنید.

JAXB

به منظور نگاشت دو طرفه بین اشیاء داده جاوا و ساختارهای xml از JAXB که مخفف کلمه های Java API for XML Binding می توانیم در برنامه های خود استفاده کنیم. هدف از این کار ایجاد ایجاد ماندگاری و سازگاری اشیاء جاوا با xml برای راحت شدن

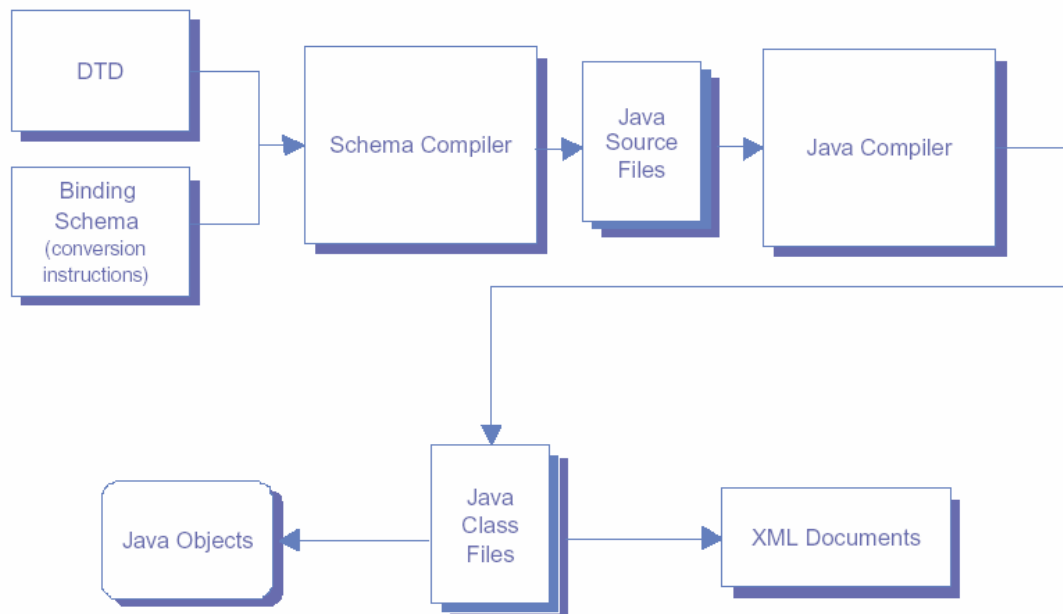
کار توسعه است. بدون JAXB فرآیند ذخیره و بازیابی (serializing و deserializing به ترتیب) اشیاء جاوا با xml نیاز به ایجاد و نگهداری کد های سنگین و زیادی برای خواندن، پارس کردن و خروجی گرفتن از آنها داریم.

می توان فرآیند سریال سازی (serialization) نوشتن نتیجه خروجی یک شی نرم افزاری در حال اجرا به یک جریان خروجی تعریف کرد. این جریانهای خروجی می توانند یک فایل یا سوکت های داده TCP باشد.

با JAXB شما قادر خواهید بود با مستندات xml، که اگر آنها اشیاء جاوا باشند کار کنید.

فرآیند توسعه در JAXP نیاز مند ایجاد یک DTD و binding schema در یک سند xml برای تعیین نگاشت بین شی جاوا و شمای xml آن میباشد. شما باید یک binding schema، DTD را به یک کامپایلر schema برای ایجاد کد های منبع جاوا ارسال کنید. کلاسهای نتیجه به وجود آمده از زمان کامپایل جزئیات فرآیند تبدیل xml به java و برعکس را دستگیر (اجرا) خواهند کرد. این به این معنی است شما به طور مستقیم نیاز به اجرای پارسرهای SAX یا DOM در کد برنامه خود نداشته و شکل زیر فرآیند JAXB را روشنتر نشان می دهد.

در زمانی که برای اولین بار JAXB ایجاد شد آن نشان داد که می توان کارایی بر روی پارسرهای SAX و DOM را بهبود داد. این به دلیل می باشد که کلاسهای آن پیش کامپایل شده و سبکتر می باشند. این از نشانه های آینده مثبت JAXB است و معلوم میکند ارتباط عمومی با کارایی وقتی که از xml استفاده می شود دارد.



در یک ارزیابی در استفاده از JAXB این است در این جا انعطاف پذیری در سیستم کمتر می شود و این زمانی که هر تغییر در xml و ساختار اشیاء نیازمند به دوباره کامپایل کردن JAXB کلاسها دارد. این عیب برای ایجاد سیستمهای که از JAXB در برنامه های خود استفاده می کند اشکال ایجاد خواهد کرد. هر تغییر در ساختار JAXB نیازمند ایجاد دوباره JAXB binding و تست دوباره آنها را به همراه دارد. اعلامیه های در مورد JAXB اعلام شده در باره اینکه در ورژن جدید این API پیش از استفاده آنها در برنامه هایتان شما می باید آنها را کاوش کنید. به طور مثال

فرآیند مشخص می کند که کدام ساختار داده xml ایجاد شده اند و از جدولهای رابطه که ساده هستند و منابع زیادی دارند استفاده می کنند. نتایج مورد انتظار و به دست آمده از این ارتقاء در اینجا امکان گفتن ندارند و درمورد آنها در یک زمان دیگر بحث کرد. برای اطلاعات بیشتر می توانید به آدرس <http://java.sun.com/xml/jaxb/> مراجعه کنید.

مقید (binding) کردن اشیاء جاوا به XML

برای دیدن JAXB در عمل ما به مثال product catalog برمی گردیم. در گذشته ما یک DTD برای آن توسعه داده ایم و آن را ملاحظه کرده اید. ایجاد الگوی مقید کردن (binding schema) کمی پیچیده تر از ایجاد DTD است و ما کار را با ایجاد فایل جدید binding schema به نام product-catalog.xjs شروع می کنیم. الگوهای مقید کردن در ورژنهای اولیه JAXB همیشه عناصر ریشه زیر را دارند.

```
<xml-java-binding-schema version="1.0-ea">
```

این عنصر تعیین می کند که این سند یک الگوی مقید کردن می باشد. حال ما عناصر پایه و درونی در سند product catalog را تعریف می کنیم.

```
<element name="description" type="class">
  <attribute name="locale"/>
  <content property="description"/>
</element>
```

همچنین

```
<element name="price" type="class">
  <attribute name="locale"/>
  <attribute name="unit"/>
  <content property="price"/>
</element>
```

صفت type از عنصر گره به این اشاره دارد که عناصر از نوع description و price در سند product catalog باید همانند اشیاء جدا در جاوا آنها برخورد شود. این کار برای این ضروری است که هر دو description و price صفات خاص خود را همانند آنچه که در سند دارند داشته باشند. عنصر content در هر دو تعریف بالا به کامپایلر JAXB می فهماند که صفات برای این کلاسها را با نام ویژه شان ایجاد کند. محتوای ایجاد شده کلاس Description با متدهای getDescription() و setDescription() می توانند مورد دسترسی قرار بگیرند. همچنین برای کلاس Price متدهای getPrice() و setPrice() ایجاد می شوند. برای این که یک شرح خلاصه از این عناصر داشته باشیم ما می توانیم به آنها در تعریفات عنصر product مراجعه کنیم.

```
<element name="product" type="class">
  <content>
    <element-ref name="description"/>
```

```
<element-ref name="price"/>
</content>
</element>
```

عنصر product به کلاس جاوا با نام Product نگاشت خواهد شد و شامل دو متغیر نمونه که یکی متغیر نمونه Description و دیگری Price خواهد بود. توجه کنید که میتوانید از کلمه element-ref به جای element در در تعریف گره های description, price, استفاده کنید. استفاده از این ساختار می تواند برای ساخت اشیاء با ساختار پیچیده و از به دور انداختن اطلاعات در سند binding جلوگیری کند. عنصر آخری که به عنصر ریشه مقید می شود در تعریفات زیر نشان داده شده.

```
<element name="product-catalog" type="class" root="true">
  <content>
    <element-ref name="product"/>
  </content>
</element>
```

توجه کنید که صفت root=true در این تعریف مقید سازی، product-catalog را به عنوان ریشه عنصر xml تعیین میکند. از این تعریف کامپایلر JAXB برای ایجاد کلاس به نام ProductCatalog شامل لیس نمونه محصولات استفاده میکند. الگوی کامل مقید سازی برای مثال ما در زیر نشان داده شده.

```
<xml-java-binding-schema version="1.0-ea">
  <element name="description" type="class">
    <attribute name="locale"/>
    <content property="description"/>
  </element>
  <element name="price" type="class">
    <attribute name="locale"/>
    <attribute name="unit"/>
    <content property="price"/>
  </element>
  <element name="product" type="class">
    <content>
      <element-ref name="description"/>
      <element-ref name="price"/>
    </content>
  </element>
  <element name="product-catalog" type="class" root="true">
    <content>
      <element-ref name="product"/>
    </content>
  </element>
</xml-java-binding-schema>
```

حال ما يك DTD و يك الگوي مقيد سازي داريم و آماده براي ايجادكدهاي منبع JAXB هستيم. از اين اطمینان پیدا کنید که در مسیر کلاس هایتان (فایلهاي برنامه) فایل JAXB jar موجود بوده و دستور زیر را اجرا کنید.

```
# java com.sun.tools.xjc.Main product-catalog.dtd product-catalog.xjs
```

اگر همه چیز درست باشد، فایلهاي زیر را در فهرست جاری شما ایجاد خواهد شد.

Description.java

Price.java

Product.java

ProductCatalog.java

حل شما مي توانيد اين کلاسها را کامپایل کرده و در برنامه هایتان استفاده کنید.

استفاده از اشیاء JAXB

استفاده از کلاسهای کامپایل شده JAXB در برنامه هاي کاربردي شما آسان مي باشد. براي خواندن اشیاء از داخل فایلهاي xml شما به ساده گي با اشیاء JAXB آنها را از سند آن مي خوانيد. اگر شما با استفاده از java.io.ObjectInputStream آشنا باشيد، اين کار بسيار ساده خواهد بود. در زیر کد نمونه اي براي خواندن سند product catalog با JAXB نشان داده شده.

```
ProductCatalog catalog = null;
File productCatalogFile = new File("product-catalog.xml");
try {
    FileInputStream fis
        = new FileInputStream(productCatalogFile);
    catalog = ProductCatalog.unmarshal(fis);
} catch (Exception e) {
    // handle
} finally {
    fis.close();
}
```

براي فرآیند عکس آن يعني ذخيره کردن نمونه شي ProductCatalog به شکل سند xml بايد کدهاي زیر را اجرا کنید.

```
try {
    FileOutputStream fos
        = new FileOutputStream(productCatalogFile);
    catalog.marshall(fos);
} catch (Exception e2) {
    // handle
} finally {
```

```
fos.close();
}
```

متد هاي unmarshal از شي ProductCatalog وظيفه تبديل عناصر داخل سند xml به کلاسهاي از نوع خود ProductCatalog را انجام مي دهد و همچنين متد marshal از کلاس تعريف شده از نوع ProductCatalog قادر به تبديل کلاس که از روي آن فراخواي شده به عناصر xml مي باشد. دقت داشته باشيد که ورودی هر دو متد از نوع جريان ورودی و خروجی فایل می باشد.

در طی پردازش برنامه شما می توانید از اشیاء JAXB استفاده کنید، همان طور که با سایر اشیاء کار میکنید مثل متغیرهای نمونه آنها. در بسیاری از حالات شما ممکن است نیاز داشته باشید به گرفتن زیر مجموعه داخل يك عنصر نمونه گرفته شده به کمک تکرار و پیدا کردن دادهای که نیاز دارید. برای مثال گرفتن شرح انگلیسی برای يك محصول نمونه، شما می بایست از کدهای زیر استفاده کنید.

```
String description = null;
List descriptions = product.getDescription();
ListIterator it = descriptions.listIterator();
while (it.hasNext()) {
    Description d = (Description) it.next();
    if (d.getLocale().equals(en_US)) {
        description = d.getDescription();
        break;
    }
}
```

این نوع تکرارها زمانی که داده ای xml با API ها پردازش می شوند لازم می باشند و فقط مخصوص JAXB نمی باشند. آنها يك قسمت ضروی در پیمایش ساختارهای درختی داده همانند xml می باشد. ما شما را دعوت می کنیم که قابلیت های کامل JAXP را در آدرس URL گفته شده در بخش قبلی، دیدن کرده و این مفید خواهد بود.

۲ طولانی کردن ماندگاری JavaBeans

يك از نام گذاری های نادرست JavaXML API، نام گذاری Long Term JavaBeans Persistence بر روی API های مخصوص نگاشت xml به کامپوننت های جاوا می باشد. قابلیت های آن شبیه به قابلیت های JAXB بوده اما شیوه کار آن به این صورت است که به جای کار با مقید سازی الگو با کامپوننت های جاوا برای نگاشت به xml کار میکند. زمانی که کامپوننت های جاوا (bean) باید متدهای get, set برای دستیابی به هر صفت (property) ایجاد کنند، به کمک آن قادر به توسعه کامپوننت های سریال شده (گرفته شده از جريان ورودی xml) خواهیم بود که از سند xml بدون مقید سازی الگو گرفته شده اند. همچنین ما را قادر به انجام عکس این عمل یعنی تبديل کامپوننت های JavaBean به فرمت استاندارد xml میکند.

این API ها هم اکنون قسمتی از Java 2 Standard Edition (J2SE) شده و نیاز به دانلو کردن کلاس های خارجی و اضافه کردن آنها در مسیر کلاس های برنامه ندارید. رابط اصلی این API ها در جدول زیر به طور خلاصه نشان داده شده. این کلاس ها سبکی شبه به java.io.ObjectInputStream و java.io.ObjectOutputStream دارند و به جای استفاده در فرمت باینری برای xml استفاده میشوند.

Class name	Description
java.beans.XMLEncoder	Serializes a JavaBean as XML to an output stream.
java.beans.XMLDecoder	Reads in a JavaBean as XML from an input stream.

نوشتن JavaBean به داخل XML

مثال زیر یک کامپوننت ساده JavaBean را با یک property به ما نشان میدهد.

```
public class SimpleJavaBean {
    private String name;
    public SimpleJavaBean(String name) {
        setName(name);
    }
    // accessor
    public String getName() { return name; }
    // modifier
    public void setName(String name) { this.name = name; }
}
```

همان طور که مشاهده میکنید این bean استاندارد javaBean برای دسترسی و تغییر property ها را با `set`, `get` ارائه کرده. با استفاده از کدهای کوچک زیر میتوانیم این bean را در داخل یک سند xml با نام `simple.xml` ذخیره کنیم.

```
import java.beans.XMLEncoder;
import java.io.*;
...
XMLEncoder e
    = new XMLEncoder(new BufferedOutputStream(
        new FileOutputStream("simple.xml")));
e.writeObject(new SimpleJavaBean("Simpleton"));
e.close();
```

کد موجود در بالا یک XMLEncoder روی `java.io.BufferedOutputStream` ایجاد کرده و آنرا به داخل فایل `simple.xml` می ریزد. سپس ما نمونه ای از شی `SimpleJavaBean` را به متد `writeObject` از `encoder` پاس میکنیم و جریان را می بندیم. محتوای فایل خروجی به صورت زیر خواهد بود.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.0" class="java.beans.XMLDecoder">
  <object class="SimpleJavaBean">
```

```

<void property="name">
  <string>Simpleton</string>
</void>
</object>
</java>

```

اینجا ما جزئیات نحوی xml را بیان نمیکنیم، زیرا شما نیاز به آموختن آنها در استفاده از این API ها ندارید.

بازیابی JavaBean از XML خواندن يك JavaBean ذخیره شده در xml به همان ساده گي ذخیره کردن آن میباشد. ما با استفاده از مثال SimpleJavaBean دو باره آن bean را خواهیم ساخت. کد هاي زیر این کار را انجام مي دهند.

```

XMLDecoder d
= new XMLDecoder( new BufferedInputStream(
    new FileInputStream("simple.xml")));
SimpleJavaBean result = (SimpleJavaBean) d.readObject();
d.close();

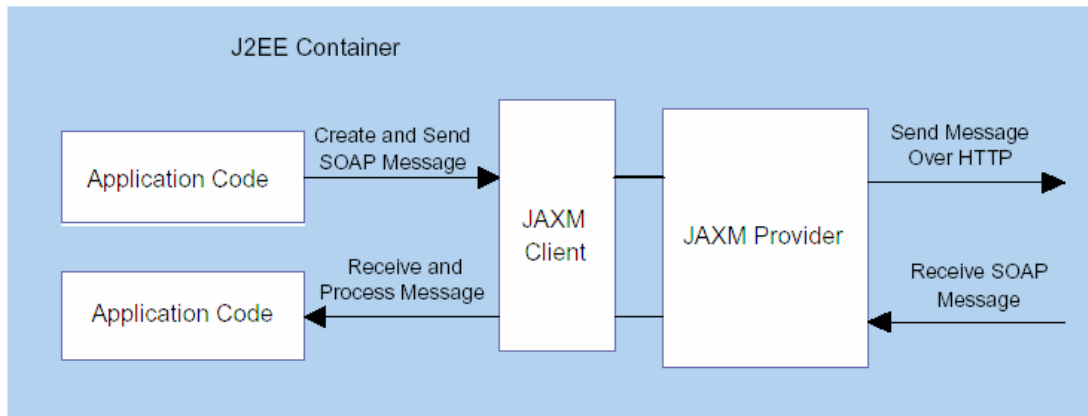
```

Decoder چگونه گي از نوع ساختن هر نوع bean ذخیره شده را با کامپوننت XMLDecoder را مي داند. این API مي تواند يك راه سریع و کم زحمت براي صادر کردن bean ها به xml براي استفاده دیگر ابزار ها و برنامه هاي کاربردي دیگر باشد. به خاطر بیاورید که شما قادر هستید که سندهاي xml شامل bean را به کمک XSLT به راحتی به سایر محیط ها و framework هاي توسعه وارد کنید!

JAXM ۵

نام گذاری JAXM براي Java API for XML Messaging که يك سري API هاي enterprise مخصوص جاوا براي استاندارد دستيابي به متدها و يك مکانيزم انتقال براي پیامهاي SOAP است. همچنین از SOAP با هر دو روش پیام رسانی همگام و ناهمگام پشتیبانی میکند.

مشخصات JAXM سرویس هاي گوناگوني را تعريف کرده که باید توسط تکمیل کننده (شرکت ایجاد کننده آن سرویس) آن JAXM ارائه شود. استفاده از هر کدام براي توسعه دهنده گان پوششي بر روي پیچیده گي هاي سیستمهاي پیام رسانی است. دستيابي کامل به این سرویسها داخل معماری JAXM در شکل زیر نشان داده شده.



دو کامپوننت اصلی در معماری JAXM Client و Provider هستند. Client قسمتی از J2EE Web یا ظرف (container) EJB می باشد که قابلیت دسترسی به سرویس های JAXM را در داخل برنامه شما فراهم می کند. همچنین Provider ممکن است از روشهای مختلفی ایجاد شده باشد و مسئول برای فرستادن و دریافت پیامهای SOAP است. به کمک این سازماندهی شما قادر به ارسال و دریافت پیامهای SOAP انحصاراً با JAXM API خواهید بود.

دو بسته در داخل JAXM API همان طور که در جدول زیر نشان داده شده وجود دارد. کامپوننتهای شما برای دستیابی به سرویسهای JAXM از رابطهای Connection و ConnectionFactory استفاده کرده و به همان روش شما می بایست دستگیر یک پیام در معماری (JMS) Java Message Service را بدست آورید. بعد از گرفتن یک Connection شما میتوانید از آن برای ایجاد ساختار پیام SOAP و فرستادن آن به میزبان راه دور از طریق HTTP استفاده کنید.

همچنین در JAXM یک Servlet جاوا موجود است که به کمک آن شما می توانید زمانی که نیاز به دستگیر کردن پیامهای SOAP داخل شونده دارید از آن استفاده کنید.

Package name	Description
<code>javax.xml.messaging</code>	Contains the <code>ConnectionFactory</code> and <code>Connection</code> interfaces and supporting objects.
<code>javax.xml.soap</code>	Contains the interface to the SOAP protocol objects, including <code>SOAPEnvelope</code> , <code>SOAPHeader</code> , and <code>SOAPBody</code>

ایجاد و استفاده از پیامهای SOAP کمی پیچیده می باشد و از JAXM برای ایجاد و دسترسی به سرویس های وب در J2EE استفاده می شود. اطلاعات بیشتر را می توانید از <http://java.sun.com/xml/jaxm> دریافت کنید.

۶ JAX-RPC

JAX-RPC برای اجرای فراخوانی روال راه دور (Remote Procedure Call) توسط xml در جاوا است. با JAX-RPC اکثر مکانیزمهای XML-RPC که پایه ای برای SOAP هستند را می توانیم استفاده کنیم. بطور مثال شما می توانید متدهای یک کامپوننت bean را که در ظرف EJB در حال اجرا می باشد را به سرویس گیرنده های تحت جاوا یا سایر برای اجرا اشکار کنید. بسته JAX-RPC یک از بسته های Java

XML Pack شده و اطلاعات به روز در این مورد در <http://java.sun.com/xml/jaxrpc> موجود است.

این نکته قابل توجه است که SOAP با سرعت در حال رایجتر شدن نسبت به XML-RPC برای ایجاد سرویسهای وب است. هم اکنون JAXM، پروتکل SOAP را کاملتر کرده و بلوغ بیشتری برای تکمیل برنامه ها پیدا کرده و آینده JAX-RPC API تا حد زیادی نا معلوم است.

JAXR v

يك کامپوننت حیاطي برای موفق تر کردن سرویسهای وب باید قادر به دستیابی و انتشار اطلاعات درباره سرویس های موجود در رجیستر های در دسترس سراسری باشد. این رجیستری ها در واقع سرویس دهنده های مخصوص پیدا کردن سایر سرویس های وب هستند. هم اکنون چندین استاندارد محاسباتی در زمینه رجیستر های سرویسهای وب وجود دارد. معمول ترین این رجیستری UDDI ebXML, می باشند.

تفاوتهای اندکی بین این رجیستری ها از انواع مختلف وجود دارد و تلاشها بر این است که تنها يك سری API برای همه این رجیستری ها استفاده شود. نتیجه این طرح يك سری API با نام (JAXR) Java API for XML Registries بود. در JAXR يك لایه انتزاعی برای شناسایی سیستم های رجیستری ایجاد گردیده، که آنرا قادر به استاندارد سازی دستیابی به اطلاعات سرویسهای وب از جاوا می سازد. انتظار میرود که JAXR شامل API های باشد که پرسوچوهای پیچیده رجیستری را را اجرا کرده و نتایج را برای ارسال و به روز رسانی داده های خود به يك سیستم رجیستری مشخص ارسال کند. مزیت اصلی آن این است که شما توانایی دستیابی به محتوای رجیستری های ناهمگون را بدون داشتن فرمت خاص آن رجیستری ویژه دارید و از فرمت استاندارد جاوا استفاده می کنید.

فقط در JNDI این امکان وجود دارد که به طور پویا می توانید منابع را کشف کنید. JAXR قادر است که بطور پویا اطلاعات رجیستری ها بر پایه xml را کشف کند.

قابلیتهای JAXR هم اکنون به طور عمومی در دست مطالعه و بررسی ایست و هنوز استاندارد کاملی برای رجیستری ها به وجود نیامده و ورژن های اولیه JAXR در آینده تغییرات عمده ای خواهد داشت.

فصل هفتم

EJB

قبل از پرداختن به EJB لازم است کمی در مورد CORBA(Common Object Request Broker Architecture) برای شما توضیح دهیم.

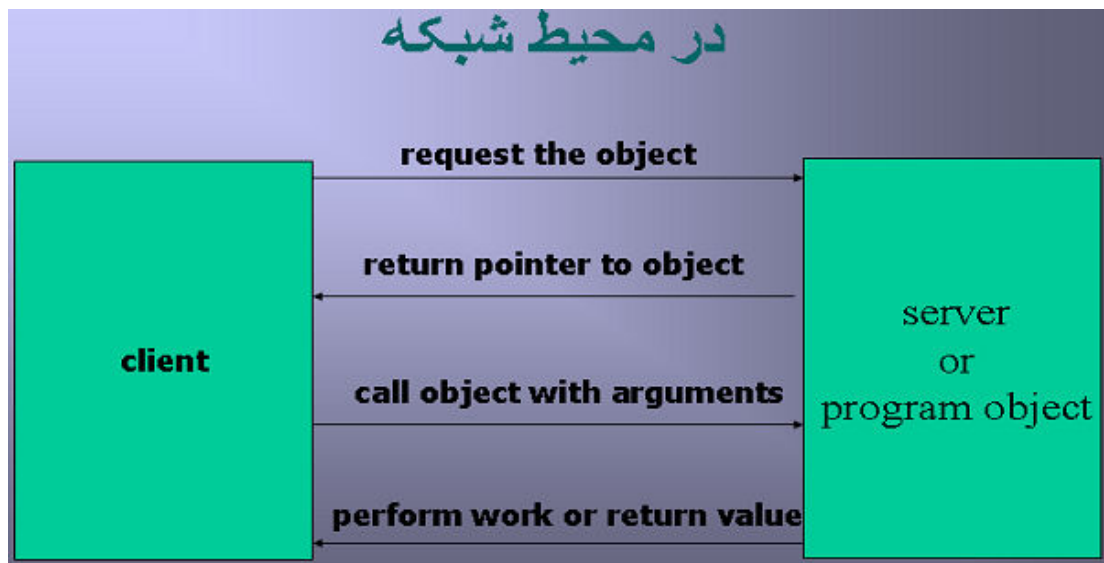
CORBA

امروزه زمانه برنامه های تک کاربره و desktop به پایان رسیده است و برنامه های جدید باید دارای قابلیت هایی در سطح Enterprise باشند که در آنها هیچ محدودیتی روی تعداد کاربران و مکان آنها نداریم. این گونه برنامه ها نیاز به زیر ساخت هایی دارند که مانند زیر ساخت هایی برای اشیا موجود در شبکه و شناسایی آنها ، امنیت دسترسی به آنها و تراکنش ها و پیام رسانی متصل و غیر متصل و همچنین کار با رویدادها و پروتکل های کنترل مدیریت Authentication و کنترل دسترسی Rule ها و موارد دیگر می باشد.

CORBA در واقع این زیر ساخت می باشد که مستقل از هر گونه سخت افزار و سیستم عامل و زبان برنامه نویسی بطور وسیعی پیشرفت نموده است. شبیه این زیر ساخت به نام COM+ که محصول شرکت Microsoft و محیطی از قبل آماده شده می باشد که به راحتی بر روی Windows platform بکار می رود. Microsoft هزینه و زمان زیادی را برای رسیدن به این قابلیت صرف نموده است که در ابتدا از ویندوز NT 3.1 با **Microsoft Transaction Server** و در NT 4 با **DCOM(Distributed COM)** و در ویندوز ۲۰۰۰ با COM+ ارایه داده است.

ORB (Object Request Broking)

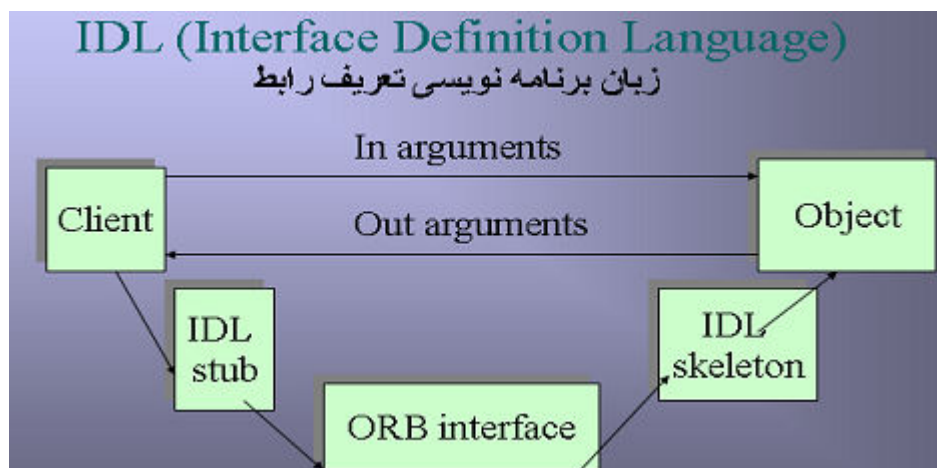
ORB واسط بین client و سرویس دهنده که وظیفه پیدا کردن متد و دادن اشاره گر آن به client و سپس ارسال فراخوانی متد به همراه پارامتر ها از طرف client به سرویس دهنده.



ORB مکانیزمی است برای انتقال Client Request به شی مقصد با رابط که در برنامه های توزیع شده شرح و مختصات فراخوانی متد را از دید Client مخفی می کند بطوری که به نظر می رسد تابع محلی را فراخوانی می کند ORB مسئول انتقال پارامترها و پیدا کردن اشیا است.



انتقال بین تعریفات IDL CORBA و زبان برنامه نویسی نهایی که بطور اتوماتیک توسط IDL COMPILER انجام می شود. در Java به وسیله کامپایلر زمان اجرا JIT(Just in Time) و در COM+ که Stubs به عنوان یک PROXY عمل می کند و عمل تبدیل آدرس شبکه به اشاره گر را انجام می دهد.



Marshalling / Unmarshalling

در دریافت یا انتقال آرگومان ها بین دو کامپوننت تبدیل آرگومان ها توسط Xml بطوری که بعدا بین اشیا شناسایی شوند را می گویند.

Argument -> Marshall -> Unmarshall -> Argument

نحوه پیدا کردن اشیاء در شبکه:

با دو روش

برپایه نام:

پیدا کردن اشیاء بر پایه نام آنها که عملکرد آن شبیه به روشی که در URL استفاده می شود.

بطور مثال در ویندوز NT ارجاع به یک فایل چنین است.

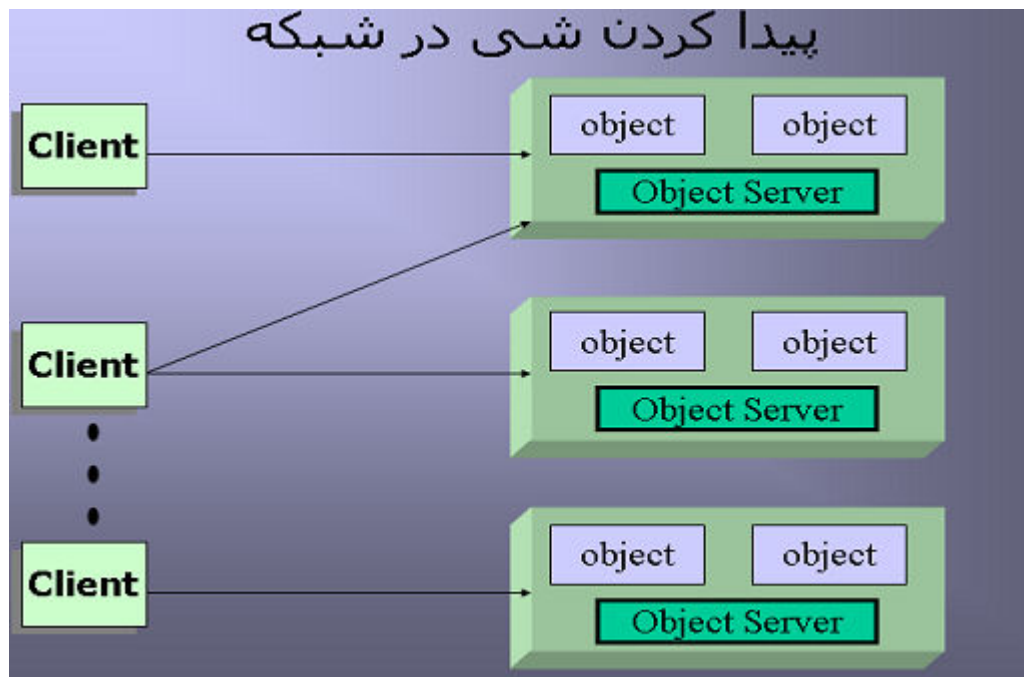
نام فایل \ نام پوشه اشتراکی \ نام سرور \

برپایه آدرس:

بر پایه مشخصات ویژه شی به همراه Socket

Socket = IP : Port Number

پس از اینکه شی پیدا شد و توسط ORB اشاره گری برگردانده می شد که به این عمل Bind کردن یا Binding گویند.



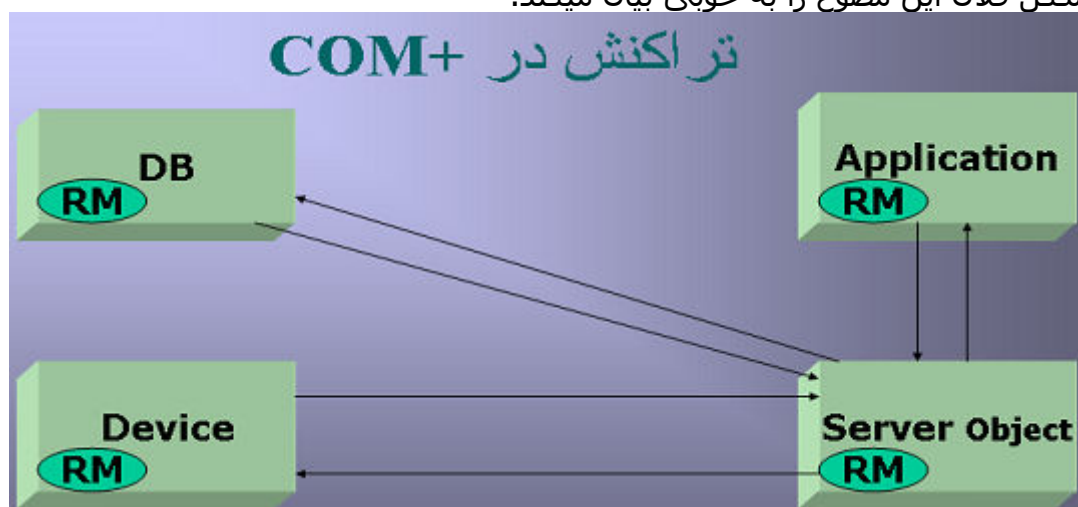
:Thread

اگر ما چند client که از اشیاء یکسان استفاده داشته باشیم به جای اجرای چند Process از اشیاء همه آنها را در داخل یک Process سیستم عامل گذاشته که قسمت کد و حافظه آنها یکی می باشد و برای هرکدام از client ها یک نخ (thread) اجرا می کنیم به این ترتیب در منابع صرفه جوی میشود. البته این اعمال توسط خود Framework انجام میشود. یک مثال در ویندوز مثل برنامه IE (internet explorer) در زمانیکه ما چند صفحه را باز کرده ایم که قسمت کد و حافظه برنامه IE یکی بوده و برای هرکدام از صفحات یک نخ اجرا میشود.

پشتیبانی تراکنش:

یکی از مهمترین نیازهای بعضی از اشیاء در منطق تجاری شرکت در تراکنش است بطور مثال فرض کنید که می خواهیم پولی را از حسابی به فردی پشت دستگاه بانک یا حسابی دیگر انتقال دهیم. در این بین ممکن است یک سری اشیاء یا دستگاه (مثل خود پرداز ATM) و بانکهای اطلاعاتی در تراکنش شرکت کنند. Resource Manager(RM) در واقع کنترل کننده نحوه شرکت آن موجودیت در تراکنش است که موفقیت یا عدم موفقیت آنرا برای کنترل کننده تراکنش بیان می کند تراکنش عملی می باشد که برای اتمام آن می بایست تمام کسانی که در آن شرکت داشته اند موافق باشند.

اگر حتی یکی از آنها مخالف باشند عمل Roll Back یا برگشت به عقب انجام می شود. شکل فلان این موضوع را به خوبی بیان میکند.



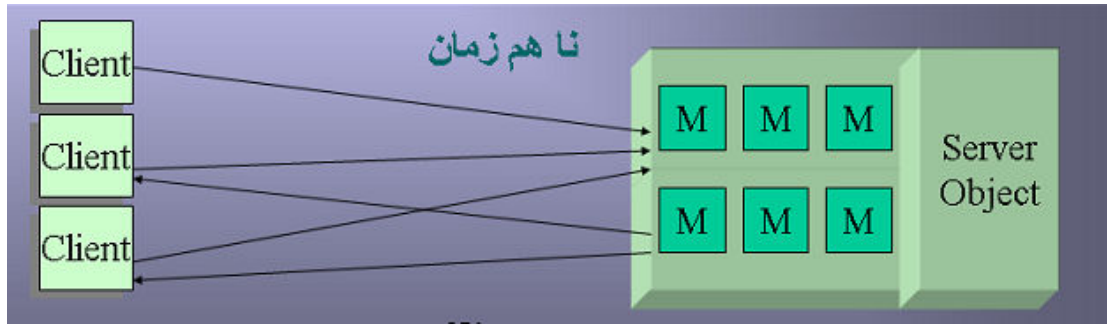
صف پیغامها (Message Queue)

در اتصال همزمان client مستقیماً به سرویس دهنده وصل شده و به کمک ORB پاسخ را دریافت میکند بطور مثال در اینترنت از یک سایت دینامیک وقتی صفحه ای را دریافت میکنیم .



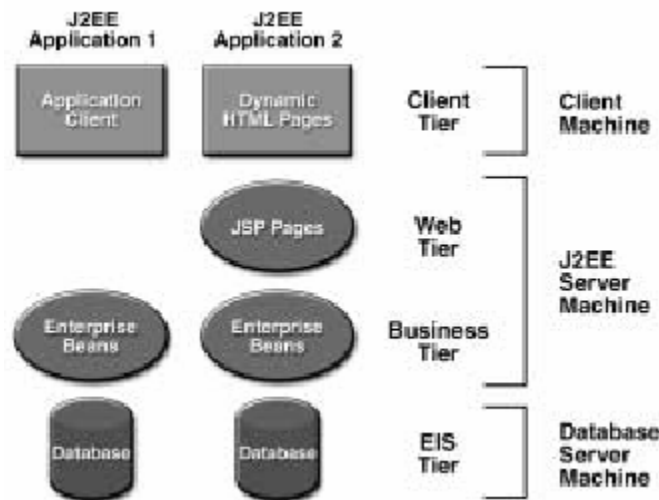
در اتصال نا همزمان Offline Client بطور همزمان و مستقیم با سرویس دهنده کار نمی کند بلکه با گذاشتن پیغام خود در صف پیغامها (درخواست مربوط به شی مورد نظر و پارامترهای آن) در زمان مناسب متصل شده و پاسخ را میگیرد. این روش در زمانی استفاده میشود که تعداد client ها زیاد است و سرویس دهنده توانایی پاسخ همزمان به همه را ندارد یا در جاهای که اتصال دائم سرویس دهنده و گیرنده وجود ندارد و همچنین زمان اتصال مشخص نباشد .

M=Message



برنامه های توزیع شده چند طرفه (Multitiered):

منطق برنامه تقسیم شده به اجزا و توابع که هر کدام روی ماشین های جدا کار های مختلف را انجام داده که به هر کدام J2EE Tier میگویند (شکل ۱)



شکل ۱

Client Tier: بر روی سرویس گیرنده قرار دارد و از سرور سرویس وب یا سرویس نرم افزار دریافت می کند

Web Tier: کامپوننت JSP که روی J2EE Server اجرا میشود

Business Tier: کامپوننت Bean که وظیفه کنترل منطق تجاری را دارد.

EIS: Enterprise Information System Tier: نرم افزار نگهداری اطلاعات که یک سرویس دهنده پایگاه داده رابطه ای میباشد و با Enterprise Resource Planning (ERP) ساختار آن مشخص میشود. بطور کلی سیستمهای Multitier دارای سه طرف هستند که Web, Business Tier یکسان فرض میشود.

J2EE Components

کامپوننتهای J2EE دارای توابعی هستند که خود را در سرور J2EE اسمبل (رجیستر) میکنند و دارای کلاسها و فایلها هستند که با کامپوننتهای دیگر ارتباط برقرار میکنند. J2EE دارای انواع کامپوننتهای زیر است

Client Application and Applets : تکنولوژی که در سمت سرویس گیرنده برای ارتباط با سرور

Java Server Page and Servlet : کامپوننت های مخصوص سرویس دهنده وب.
Enterprise Java Beans (EJB) : کامپوننت Bean که وظیفه کنترل منطق تجاری و Application Server .

این کامپوننتها به زبان جاوا نوشته و کامپایل شدهاند و تفاوت بین کامپوننتهای J2EE و کلاسهای استاندارد جاوا در این است که کامپوننتهای J2EE باید در سرور J2EE اسمبل (رجیستر) شده و نصب (deploy) شود و مدیریت آن به عهده J2EE Runtime Manager است.

:Applets

اپلت یک برنامه سمت سرویس گیرنده کوچک به زبان جاوا است و ماشین مجازی جاوا آن را روی مرورگر اینترنت اجرا میکند و باید Java Plug in نصب باشد. کامپوننتهای وب جاوا راه حل خوبی برای طراحی نرم افزار بصورت مایکروسافت و واضح میباشد زیرا راه ساده ای برای تقسیم برنامه به دو سمت سرویس دهنده و گیرنده و هر دو به زبان جاوا طراحی می شود

:Application Clients

Application Clients های J2EE در روی ماشین سرویس گیرنده اجرا شده و با اتصال به سرویس دهنده J2EE Enterprise Bean با پرتکل های ارتباطی یا استفاده از XML و همچنین با ایجاد رابط کاربر GUI با Swing or Abstract window Toolkit (AWT) یا با خط فرمان همانند Ms Dos متصل میشود.

روش دیگر با http و ارتباط با Servlet اجرا شده روی jsp سرور.

JavaBeans Component Architecture

معماری کامپوننت JavaBeans برای مدیریت جریان داده بین برنامه Client یا Applet به یک پایگاه داده یا کلاسهای اجرای روی سرویس دهنده.

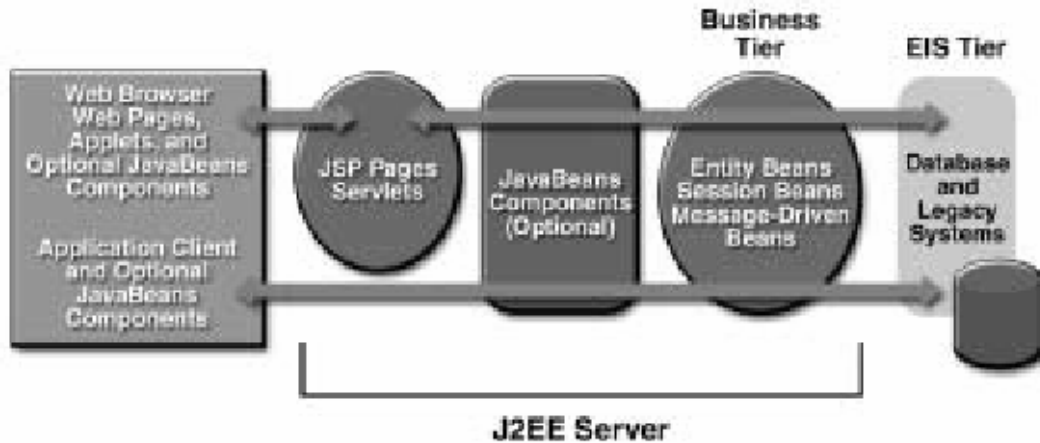
JavaBeans Component دارای متغیر های نمونه و متدهای Get, Set برای دستیابی به دادههای متغیر ها است (همانند Property ها در میکروسافت .NET). و منطق تجاری برنامه شما را در کامپوننت کپسوله سازی میکند. سرویس گیرنده با اتصال و فراخوانی متد های داخل کلاس های کامپوننت آنها را اگر مجوز داشته باشد اجرا می کند و نتیجه به آن فرستاده میشود.

کامپوننتها در داخل ظرفهای به نام Container که در قسمتهای بعد توضیح داده خواهد شد قرار میگیرند که وظیفه نگهداری و ایجاد و مدیریت کامپوننتها را دارد.

:Business Components

کدهای تجاری آنها در حوزه های مختلف تجارت از قبیل بانکداری و مالی و... که در داخل Bean (کامپوننت J2EE) در سمت Business tier اجرا می شوند (در شکل ۲).

در شکل ۲ نحوه دریافت داده های Client و پردازش آن و فرستادن به Enterprise Information System Tier: EIS و سپس فرستادن پاسخ به Client.



شکل ۲

انواع Bean

۱ Session Bean: زمانی که Client به سرور J2EE متصل می‌شود برای آن اتصال فقط فعال می‌شود (بصورت گذرا) و بعد از قطع اتصال (Disconnect) آن اشیاء داده‌های آنها از بین می‌روند.

۲ Entity Bean: اشیاء داده‌ای که برعکس Session داده‌های آنها پایدار است و بعد از قطع اتصال داده‌های آنها در جداول پایگاه داده ذخیره می‌شود.

۳ Message Driven Bean: ترکیبی از مشخصات Session Bean و Java Message Service که با آنها کامپوننت از JMS بصورت ناهمگام پیغام دریافت می‌کند. بیشتر بحث ما روی Session, Entity Bean می‌باشد.

J2EE Containers

به طور عادی نوشتن کد سرویس گیرنده Thin Client مشکل می‌باشد بخاطر اینکه بسیاری از کدها را برای مدیریت حالت‌های مختلف و کنترل تراکنش، چند نخ (multithreading) و غیره بسیار پیچیده می‌باشد. معماری بر پایه کامپوننت و مستقل از ماشین J2EE این امکان را به ما می‌دهد تا به راحتی برنامه نوشته‌شده را در محیط تجاری برنامه در کامپوننت‌های با قابلیت استفاده مجدد سازمان دهی شده و Container در زیر لایه کامپوننت‌ها برای آنها سرویس مورد نیاز را برای مدیریت منابع، اشیاء، تراکنش‌ها و چند نخ را ارائه می‌دهد.

:Container Services

Containers رابط بین کامپوننت‌ها و توابع ویژه در زیربنای سطح پائین مورد نیاز هستند پس یک Bean وب یا Enterprise باید به داخل یک Container اسمبل شود. اسمبل کردن تنظیم مشخصات ویژه Container برای سرور J2EE و این تنظیمات شامل:

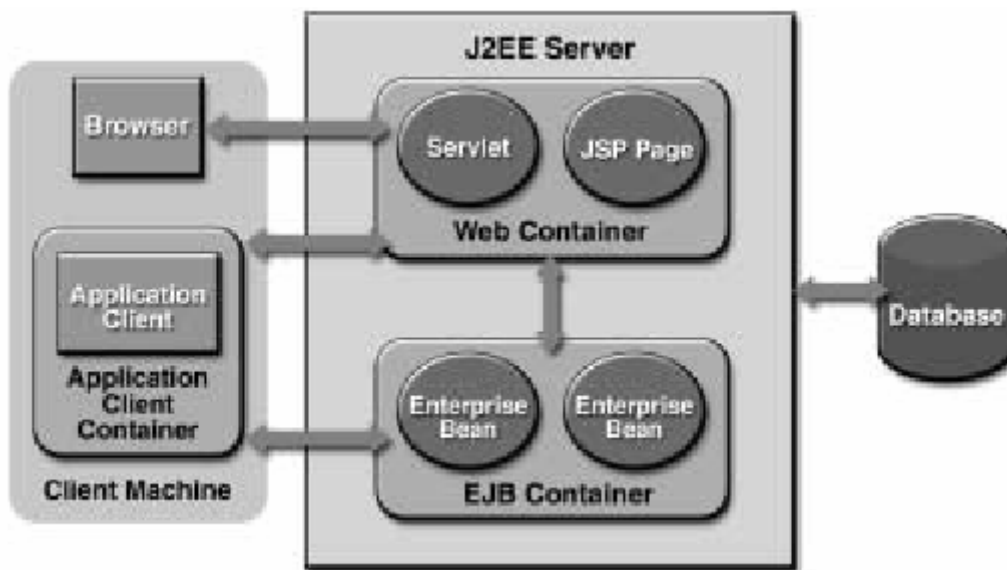
مشخص کردن مدل امنیتی بر پایه دستیابی با شناسایی شدن (authorized) مشخص کردن مدل تراکنش که روابط ویژه بین متدها در تراکنش (یک مثال مثل انتقال پول از یک حساب به حساب گیرنده که توابع گیرنده و فرستنده و پایگاه داده هر دو داخل تراکنش قرار می‌گیرند و در نهایت تراکنش زمانی موفق می‌شود که همه راضی باشند برای اطلاعات بیشتر می‌توانید به کتاب‌های پایگاه داده مراجعه کنید.) مشخص کردن JNDI (Java Naming and Directory Interface service) رابط ارتباطی برای سرویس فهرست راهنما که در شبکه‌های سازمانی موجود است (بطور مثال آدرس E mail شما در اینترنت به همراه @ یکی از آن نوع سرویس‌ها است).

مشخص کردن ارتباط راه دور که برای ارتباط سرویس گیرنده و EJB (Enterprise Java Bean) (بطور مثال RMI (Remote Method Invoktion) و RPC (Remote Procedure Call) که RPC بیشتر در COM+ استفاده می‌شود.

معماری J2EE سرویس پیکره بندی نیز ارائه میکند بطور مثال سرویس گیرنده به پایگاه داده متصل شده با تنظیمات امنیتی و ارتباطی که J2EE با API پایگاه داده خود انجام می دهد.

انواع Container:

همان طور که در شکل ۳ مشخص میباشد در سرویس دهنده J2EE سه نوع Container وب ، Client و EJB موجود است در نوع وب وظیفه مدیریت اجرای صفحات وب جاوا و کامپوننتهای Servlet رابه عهده دارد و در نوع EJB مدیریت اجرای کامپوننتهای Bean . در نوع Client نیز مدیریت اجرای نرم افزار و Applet در روی Client (می توان گفت که Applet کلاسهای شبیه به ActiveX در تکنولوژی میکروسافت می باشد).



شکل ۳

بسته بندی Packaging:

میتوان کامپوننتهای J2EE را در داخل بسته های نرم افزاری جدا گذاشت و در جاهای دیگر از آنها استفاده کرد(با deploy کردن) بسته ها شامل فایل های وابسته نیز مثل کلاسهای مفید gif, html میباشد. بسته ممکن است از یک یا چند Bean وب و Enterprise بوده به همراه شرح دهنده (descriptor) که یک سند XML که تنظیمات نصب (deploy) کردن کامپوننت در داخل آن نوشته شده می باشد. این شرح میتواند شامل مشخصات امنیتی یا مدل و متد های تراکنش و غیره که به راحتی قابل تغییر بوده و لازم به تغییر کد منبع نیست چون بصورت بیانی میباشد.

بسته ها با پسوند EAR (Enterprise Archive) بسته های برای کامپوننتهای Enterprise هستند که فایل های EAR یک استاندارد از بایگانی فایل جاوا JAR (Java Archive) می باشند
بسته ها با پسوند WAR (Web Archive) یک شرح دهنده نصب برای فایل کامپوننتهای وب و منابع وابسته می باشد . از نوع JAR در بسته های سمت Client استفاده میشود .

در مدل EAR میتوانید چند عدد از کامپوننتهای داخل EAR را بطور جدا با تنظیمات متفاوت نصب کنید.

نقشهای توسعه (Development Roles):

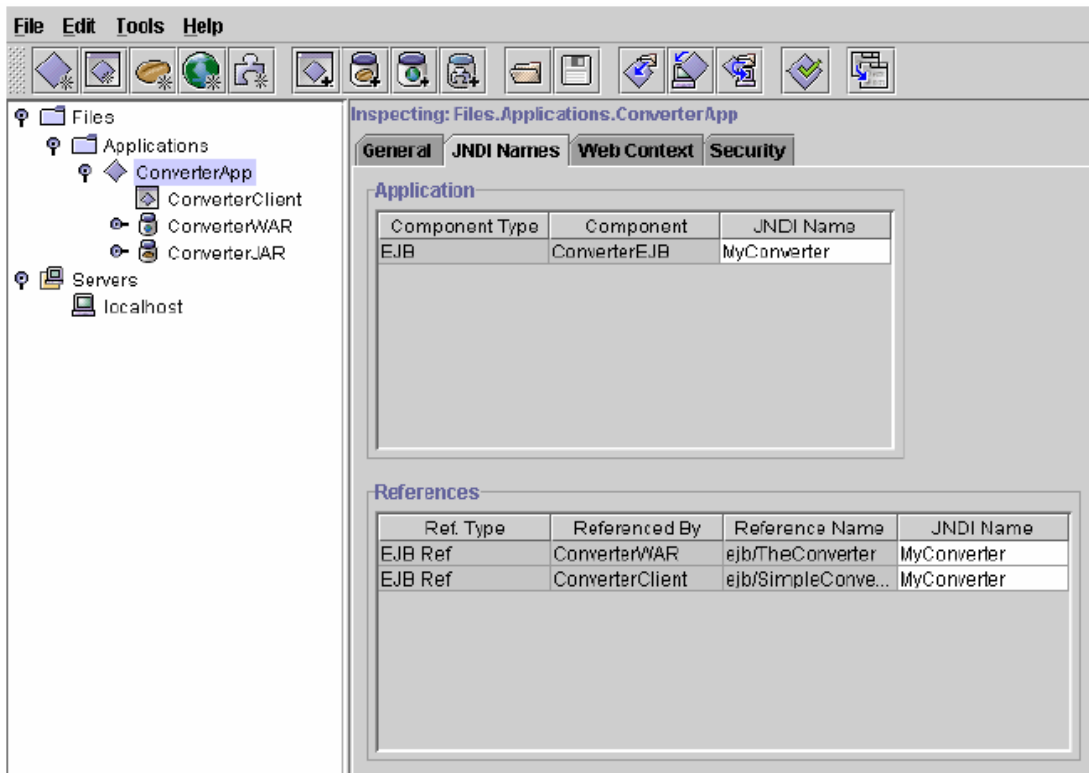
تقسیم فرایند نصب و ارتقاء (deploy) به نقشها مثل افراد و شرکتها برای اجرای قسمتهای جدا در سازمانهای بزرگ. خرید و نصب ابزارها و برنامه ها و بعد اسمبل کردن برنامه یا کامپوننتهای J2EE که این مراحل می تواند توسط تیمهای جدا انجام شود مثلاً تیمی برای خرید یا توسعه EAR آماده و تیمی برای تبدیل فایلهای EAR, WAR, JAR به یک برنامه کاربردی J2EE.

Enterprise Bean Developer: فرد یا سازمان که این کارها را انجام میدهد
نوشتن و کامپایل کد منبع، نوشتن شرح دهنده (deploy) و بسته بندی فایلهای class. و شرح دهنده به فایل EAR JAR.

Web Component Developer: فرد یا سازمان که کارهای نوشتن و کامپایل کد منبع Servlet، نوشتن فایل های JSP, Html، نوشتن شرح دهنده (deploy) و بسته بندی فایلهای class, JSP, Html و شرح دهنده به فایل WAR.

Application Deployment Tool: یک ابزار برای deploy کردن به نام deploytool که امکانات اسمبل کردن و تغییر برنامه های J2EE را دارد و دارای دو ورژن خط فرمان و گرافیکی میباشد که شکل نوع گرافیکی را آن را در زیر مشاهده می کنید (شکل ۴). در مدل گرافیکی Wizard ها ی مختلف برای بسته بندی کردن و نصب دارد.

در نوع خط فرمان انواع Script ها را برای مدیریت سرویس دهنده J2EE می توان نوشت مثل ایجاد کلید برای کامپوننت، بسته بندی کامپوننت، اجرای کامپوننت در سرور J2EE یا در کلاینت، فعال و یا غیر فعال کردن سرور J2EE و سایر اعمال.



Enterprise JavaBeans

برای آموختن طرز کار EJB با یک مثال ساده شروع میکنیم شی `PersonServer` یک شی توزع شده تجاری با رابط `Person` که دارای دو متد `getName()` و `getAge()` البته در برنامه های واقعی این متد ها بیشتر میباشد

```
public interface Person {

    public int getAge() throws Throwable;

    public String getName() throws Throwable;
}
```

حال به تعریف کلاس `PersonServer` که از رابط `Person` استفاده میکند می پردازیم .

```
public class PersonServer implements Person {
    int age;
    String name;
    public PersonServer(String name, int age){
        this.age = age;
        this.name = name;
    }
    public int getAge(){
        return age;
    }
    public String getName(){
        return name;
    }
}
```

Now we need some way to make the `PersonServer` available to a remote client. That's the job of the `Person_Skeleton` and `Person_Stub`. The `Person` interface describes the concept of a person independent of implementation. Both the `PersonServer` and the `Person_Stub` implement the `Person` interface because they are both expected to support the concept of a person. The `PersonServer` implements the interface to provide the actual business logic and state; the `Person_Stub` implements the interface so that it can look like a `Person` business object on the client and relay requests back to the skeleton, which in turn sends them to the object itself. Here's what the stub looks like:

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
```

حالا ما باید کاری کنیم تا PersonServer برای سرویس گیرنده قابل دسترس باشد این کار با Person_Skeleton و Person_Stub انجام میشود . PersonServer و Person_stub رابط Person را کامل میکنند و سرویس گیرنده به کمک Person_Stub برای PersonServer تقاضا می فرستد

```
import java.io.ObjectInputStream
import java.io.ObjectOutputStream;
import java.net.Socket;

public class Person_Stub implements Person {
    Socket socket;
    public Person_Stub() throws Throwable {
        socket = new Socket("localhost",9000);
    }
    public int getAge() throws Throwable {
        // When this method is invoked, stream the method name to the
        // skeleton.
        ObjectOutputStream outStream =
            new ObjectOutputStream(socket.getOutputStream());
        outStream.writeObject("age");
        outStream.flush();
        ObjectInputStream inStream =
            new ObjectInputStream(socket.getInputStream());
        return inStream.readInt();
    }
    public String getName() throws Throwable {
        // When this method is invoked, stream the method name to the
        // skeleton.
        ObjectOutputStream outStream =
            new ObjectOutputStream(socket.getOutputStream());
        outStream.writeObject("name");
        outStream.flush();
        ObjectInputStream inStream =
            new ObjectInputStream(socket.getInputStream());
        return (String)inStream.readObject();
    }
}
```

به کمک جریان ورودی و خروجی و Socket این امکان را فراهم میکنیم تا از پورت ۹۰۰۰ از Localhost توابع getName() , getage() را فراخوانی یا آنها را مقدار دهی کنیم

وقتی که یک متد روی Person_Stub فراخوانی میشود یک token از نوع رشته ای ایجاد شده و به skeleton جریان داده میشود . این توکن (نام تابعی که فراخوانی شده) روی شی تجاری مورد نظر جستجو و اجرا میشود و نتیجه توسط skeleton به سرویس گیرنده ارسال میشود حال به کدهای skeleton توجه کنید.

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
```

```

import java.net.Socket;
import java.net.ServerSocket;

public class Person_Skeleton extends Thread {
    PersonServer myServer;
    public Person_Skeleton(PersonServer server){
        // Get a reference to the business object that this skeleton wraps.
        this.myServer = server;
    }
    public void run(){
        try {
            // Create a server socket on port 9000.
            ServerSocket serverSocket = new ServerSocket(9000);
            // Wait for and obtain a socket connection from stub.
            Socket socket = serverSocket.accept();
            while (socket != null){
                // Create an input stream to receive requests from stub.
                ObjectInputStream inStream =
                    new ObjectInputStream(socket.getInputStream());
                // Read next method request from stub. Block until request is
                // sent.
                String method = (String)inStream.readObject();
                // Evaluate the type of method requested.
                if (method.equals("age")){
                    // Invoke business method on server object.
                    int age = myServer.getAge();
                    // Create an output stream to send return values back to
                    // stub.
                    ObjectOutputStream outStream =
                        new ObjectOutputStream(socket.getOutputStream());
                    // Send results back to stub.
                    outStream.writeInt(age);
                    outStream.flush();
                } else if(method.equals("name")){
                    // Invoke business method on server object.
                    String name = myServer.getName();
                    // Create an output stream to send return values back to
                    // the stub.
                    ObjectOutputStream outStream =
                        new ObjectOutputStream(socket.getOutputStream());
                    // Send results back to stub.
                    outStream.writeObject(name);
                    outStream.flush();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}
} catch(Throwable t) {t.printStackTrace();System.exit(0); }
}
public static void main(String args []){
    // Obtain a unique instance Person.
    PersonServer person = new PersonServer("Richard", 36);
    Person_Skeleton skel = new Person_Skeleton(person);
    skel.start();
}
}
}

```

Person_Skeleton درخواست stub را به شی تجاری مورد نظر یعنی PersonServer میفرستد اساساً Person_Skeleton تمام وقت خود را برای گرفتن درخت و است سروس گیرنده صرف میکند . Person_Skeleton نتیجه گرفته از شی را به صورت جریان داده به stub میفرستد .

حالا نوبت ایجاد کد های ساده لازم برای سرویس گیرنده است.

```

public class PersonClient {
    public static void main(String [] args){
        try {
            Person person = new Person_Stub();
            int age = person.getAge();
            String name = person.getName();
            System.out.println(name+" is "+age+" years old");
        } catch(Throwable t) {t.printStackTrace();}
    }
}
}

```

برنامه client چگونه گی استفاده از stub را نشان می دهد و با کمک Proxy و تبدیل آدرس فیزیکی به اشاره گردد شبکه این کار عملی میشود .

RMI (Remote Method Invocation)

Rmi درواقع کل این فرآیند میباشد که معادل ORB در CORBA و RPC در COM+ میباشد RMI پایه سیستمهای اشیاء توزیع شده J2EE میباشد

در پرتکولهای اشیاءتوزیع شده مثل Java RMI , DCOM , CORBA یک سری ساختار و ابزارها برای ایجاد وکنترل skeleton , stub وجود دارد همچنین آنها میتوانند اعمال پاس کردن پارامترها handle کردن خطاها وسایر سرویسها مثل تراکنش و امنیت را در زمینه خود ارائه دهند.

مدل کامپوننتها (Component Models)

Enterprise JavaBeans دارای مدل مشخص و ویژه خود در سمت سرور می باشد که از یک سری کلاسها و رابطهای پکیج Java.ejb استفاده میکند. توسعه دهنده گان میتوانند برای ایجاد و اسمبل و deploy کردن کامپوننتها ارزش و ویژگی های EJB استفاده کنند .

JavaBeans یک مدل کامپوننت برای بسته بندی اشیاء میباشد اما مثل EJB مدل کامپوننت Server side نیست و ویژگیهای کنترل اشیاء صف پیغامها و سایر موارد گفته شده را ندارد.

یک کامپوننت برای کاربرد خاص توسعه داده می شود و آن یک برنامه نیست. بطور مثال در داخل یک دکمه یا صفحه گسترده در داخل یک نرم افزار با GUI (رابط گرافیکی کاربر) برطبق قواعد JavaBeans استفاده میشود اما در مدل EJB کامپوننت(شیء تجاری مشتری) باید به داخل یک EJB Server وارد شده و مدیریت آن به عهده سرور و نه برنامه کاربردی می باشد .
سایر مدل های کامپوننت جاوا شامل JSP , Servlet (در ASP در .NET) و Applets (در ActiveX در .NET) می باشند.

مانیتور تراکنش کامپوننتها (CTM Component Transaction Monitors)

CTM همزمان با پلاتفرم اشیاء توزیع شده و ORB رشد کرد و از مانیتورهای TP (Transaction processing) استفاده می کند. TP اول در سیستم عاملها برای برای مانیتورینگ برنامه های تجاری استفاده شد سپس برای مدیریت محیط اشیاء تجاری استفاده شد

در این روش client درخواست خود را به TP میدهد و TP آن روال را به همراه پارامترهایش برای سرویس گیرنده اجرا میکند. تفاوت اساسی بین RPC , RMI در این است که RPC از برنامه های بر پایه روال استفاده میکند و Procedural base است ولی RMI از سیستم اشیاء توزیع شده استفاده میکند و شیء گرا می باشد.

CTMs در واقع پیوند ORB و TP میباشد که تراکنش و مدیریت منابع (RM) را نیز پشتیبانی می نماید البته پیچیده تر هم شده .
استراتژی مدیریت منابع شامل سرکشی منابع ، فعال سازی و جابهجای ها میباشد و توسعه دهنده گان بدون آن مجبور هستند که خود کدهای کنترلی بنویسند.

این را میتوان با مثال شرح داد همان طوری که ما در پایگاه داده رابطه ای محتوای جداول ، کلید اصلی ، شاخصها و روالهای ذخیره شده را ایجاد میکنیم و شما

الگوریتمهای شاخص سازی و مدیریت Cursor SQL را توسعه ندهید بلکه از DBMS پایگاه داده که ساخت شرکتهاى متخصص این کار هستند استفاده میکنید . در این جا هم CTMS همان قابلیتهاى مورد نیاز را میدهد . کامپوننتهاي سمت سرور بر پایه مشخصات ویژه خود هستند و رابطه بین کامپوننتهاي سمت سرور و CTMS شبیه رابطه بین CD و CD-ROM میباشد. همچنین رابطه بین CTMS و مدل کامپوننت مثل رابطه ریل راه آهن و ترن قطار میباشد بطوري که راه آهن محیط ترن را مدیریت میکند مثل همزمانی یا کنترل ترافیک و توازن بار.

پیغامدهی غیرهمزمان (Asynchronous Messaging)

به نرم افزار ها این امکان رامیدهد که اطلاعات را با پیغام تبادل کنند. منظور ما از پیغام یک بسته داده های تجاری در شبکه که همراه سرآیند های آدرس دهی است که البته فرستنده و گیرنده یا گیرنده ها باید مشخص باشند و از Message oriented MOM (middleware) استفاده می کند . MOM تضمین میکند که پیغامها بین نرم افزارها توزیع و جا به جا میشوند و MOM تحمل بار(fault tolerance)، ایجاد توازن درخواست (load balancing) و پایداری پیغامها را برای پیغامهای نرم افزار Enterprise پشتیبانی می کند . به همین منظور دارای API های مخصوص به خود که میتواند در برنامه خود استفاده کنید

Java Message Service

Java Message Service یک MOM محصول شرکت SUN با نام اختصاری JMS که API های آن را در J2EE استفاده شده و میتواند برای تبادل پیغام با سایر محصولات مثل Microsoft Message Queue از آنها بهره ببرید.

Message Driven Beans

در EJB 2.0 یک نوع کامپوننت جدید ابداع شد که Message driven Beans نامیده شد که همان کامپوننتهای استاندارد JMS هستند. آنها می توانند پیغامهای JMS را ارسال یا دریافت نمایند و با کامپوننتهای RMI تعامل داشته باشند همچنین در تراکنشهای سمت سرور شرکت کنند. از این کامپوننتها باید با توجه به منطق و نیاز تجاری استفاده کرد که قطعاً تأثیر مطلوبی در برقرای ارتباط با سایر سیستم شبیه به خود دارد.

Microsoft .NET Frame work

میکروسافت اولین تولید کننده CTM بود که با نام Microsoft Transaction Server یا به اختصار MTS نامیده شد که در سال 2000 نام آن به COM+ تغییر کرد. COM+ بر پایه COM (Component Object Model) بوده و MTS در سال ۱۹۹۶ تولید شد و توسعه دهنده گان کامپوننتهای COM را بدون مشخصات سطح سیستمی مینوشتند.

COM+ به طور خودکار از همه کنترلرهای لازم در CTMs پشتیبانی میکرد و امروزه جزئی از .NET Framework شده که در آن اشیاء COM تبدیل به اشیاء .NET شده که در آنها به CLR (Common Language Runtime) برای اجرا شدن نیاز دارند شبیه Java Virtual Machine.

هر چند که .NET Framework بسیار مزایای ارائه داده این استاندارد فقط بر روی پلات فرم میکروسافت اجرا شده و در سازمان های بزرگ امروزی که انواع مختلف سیستم عاملها و پرتکولهای شبکه وجود دارد قابلیت اجرا نداشته. دلیل دیگر رقابت سایر شرکتها با میکروسافت در زمینه فناوری اطلاعات که محصولات آنها ارزانتر میباشد.

EJB and CORBA

زمانی میکروسافت در حال توسعه CTMs خود بود سایر شرکتها بر روی یک استاندارد آزاد (مستقل از سیستم) کار میکردند که CORBA نام گرفت.

CORBA چون توسط شرکتهای افراد مختلف و نه مهندسان پرهزینه (مثل مهندسان ماکروسافت) طراحی شده دارای قابلیتهای بالا با سطح مشخصات برای انواع محیطها میباشد و اکنون توسط OMG (Object Management Group) کسانی که CORBA را توسعه دادند استاندارد سازی میشود. CORBA برای برقرار ی ارتباط بین سیستمهای ناهمگون Object SOAP (Simple Access Protocol) را ابداع کرده که قسمتی از CORBA IIOP (Internet Inter Operability Protocol) میباشد.

در سال ۱۹۹۷ شرکت سان استاندارد Enterprise JavaBeans بر اساس CORBA و زبان Java توسعه داد يك سري API پایه براي اينكار طراحي كرد كه بانام EJB 1.0 بيرون آمد.

Enterprise JavaBeans

انواع Beans :

Session, Entity, Message Driven که در آنها Session, Entity بر پایه RMI بوده و Message Driven در EJB 2.0 براي کار کردن راحتتر با JMS طراحي گرديد.

۱ Session Bean : زمانی که Client به سرور J2EE متصل میشود برای آن اتصال فقط فعال میشود (بصورت گذرا) وبعد از قطع اتصال (Disconnect) آن اشیاء داده های آنها از بین میروند.

۲ Entity Bean : اشیاء داده ای که برعکس Session داده های آنها پایدار است وبعد از قطع اتصال داده های آنها در جداول پایگاه داده ذخیره می شود

۳ Message Driven Bean : ترکیبی از مشخصات Session Bean و Java Message Service که با آنها کامپوننت از JMS بصورت ناهمگام پیغام دریافت میکند .

بیشتر بحث ما روی Session , Entity Bean میباشد.

يك راه فهم bean بررسیي نحوه كامل کردن كد آنها میباشد . براي Session , Entity شما باید يك رابط تعریف کرده به همراه يك كلاس Bean و كلید اصلي . بطور اساسي دو نوع رابط کامپوننت وجود دارد Local , Remote .

:Remote Interface

براي دسترسی به يك متد در داخل EJB container از بیرون از آن ظرف . رابط راه دور باید javax.ejb.EJBObject را با extend درك خود توسعه دهد همچنین Java.rmi.Remote را با Import داخل برنامه خود كند .

:Local Interface

براي دسترسی به يك متد در داخل EJB container از داخل آن ظرف توسط يك Bean دیگر در داخل همان فضای آدرس . این قابلیت در EJB 2.0 ارائه شد که ارتباط bean ها را برقرار می کرد بدون هزینه بالا سري براي اشیاء توزیع شده . رابط راه دور باید javax.ejb.EJBLocalObject را با extend درك خود توسعه دهد .

: Local Home Interface

تعریف کننده توابع چرخه زندگی Bean هاي که توسط سایر bean هاي دیگر در داخل همان فضای آدرس آن EJB قابل استفاده اند . این قابلیت در EJB 2.0 ارائه شد که ارتباط bean ها را برقرار می کرد بدون هزینه بالا سري براي اشیاء توزیع شده . رابط راه دور باید javax.ejb.EJBHomeObject را با extend درك خود توسعه دهد .

Bean Class

کلاسهای Session , Entity beans کامپوننتهای تجاری و متدهای چرخه اشیاء را کامل میکنند.
برای session bean باید javax.ejb.SessionBean استفاده کرد همچنین برای entity bean باید javax.ejb.EntityBean استفاده کرد .
هر دو Session , Entity beans در کد خود javax.ejb.EJBHomeObject را با extend توسعه میدهند.

Message driven bean در EJB 2.0 از رابط کامپوننت همانند Session , Entity beans استفاده نمیکند چون از بیرون EJB container قابل دسترس نیستند و فقط شامل يك متد میباشد ()onMessage که به وسیله خود container فراخوانی میشود یعنی در زمانی که يك پیغام برای آن bean می رسد .

Message driven bean رابطهای javax.ejb.MessageDrivenBean و javax.ejb.MessageListener را استفاده میکند. رابط MessageListener برای برقراری ارتباط bean و سرویس JMS میباشد .
Message driven bean مانند Session , Entity beans در کد خود javax.ejb.EJBHomeObject را با extend توسعه میدهد.

کلی اصلی (Primary Key)

يك کلاس بسیار ساده میباشد که يك اشاره گر به پایگاه داده میدهد و فقط entity bean به آن نیاز دارد و از java.io.Serializable استفاده میکند.

در EJB نیاز به رابط محلی ندارید چون شما میدانید که در توسعه Enterprise Bean فقط نیاز به ارتباط با سرویس گیرنده راه دور دارید پس لازم نیست در رابط محلی RemoteException تعین و throw کنید. سرویس گیرنده بطور مستقیم با کلاس bean تعامل ندارد و با stub به طرف آن EJB متصل میشود. تعامل بین Enterprise Bean و سرویس دهنده آن که به کمک container یا ظرف مدیریت شده و مسئول ارائه رابط بین bean و سرور میباشد .
container وظیفه ایجاد يك نمونه از bean و اطمینان از ذخیره شدن آن در سرور دارد و ابزار های برای آن توسط شرکتهای مختلف تولید شده . این ابزارها يك نگاشت بین entity bean و رکورد ذخیره شده در پایگاه داده ایجاد میکند .

حال بررسی يك مثال عملی

وقتی در مورد رابط Remote صحبت میکنیم در Cabin EJB ما میخواهیم ترکیب کنیم نام تجاری عادی را با کلمه Remote برای مثال نام رابط را CabinRemote interface میگذاریم و همچنین برای رابطهای محلی و خانه گی نامهای CabinLocal interface و CabinHome interface انتخاب میکنیم.
در نهایت نام کامپوننت نیز CabinBean می شود.

:Remote Interface

حال کد رابط يك entity bean يا stateful Enterprise bean را با هم کامل میکنیم .
ما فرض میکنیم که CabinEJB يك کابین کشتي در سفر دریاي مورد نظر ماست وبا رابط راه دور که که
داراي چهار متد با قابلیت کنترل استسنا با extend کردن javax.ejb.EJBObject در زیر مشاهده میکنید.

```
import java.rmi.RemoteException;

public interface CabinRemote extends javax.ejb.EJBObject {

    public String getName() throws RemoteException;

    public void setName(String str) throws RemoteException;

    public int getDeckLevel() throws RemoteException;

    public void setDeckLevel(int level) throws RemoteException;

}
```

متد اول ودوم برای کارکردن با نام کابین و متد دوم برای سطح آن کابین نسبت به دکل کشتي . البته شما
میتوانید متدهاي دیگری نیز اضافه کنید.

:Remote home interface

وظیفه Remote home interface تعیین متد هاي چرخه زندگی که توسط client از session , entity
bean برای اشاره به enterprise bean استفاده میکند .
نام آنرا CabinHomeRemote گذاشته ایم.

```
import java.rmi.RemoteException;

import javax.ejb.CreateException;

import javax.ejb.FinderException;

public interface CabinHomeRemote extends javax.ejb.EJBHome {

    public Cabin create(Integer id)

        throws CreateException, RemoteException;

    public Cabin findByPrimaryKey(Integer pk)

        throws FinderException, RemoteException;
```

```
}
```

تابع `create()` وظیفه مقدار دهی یک نمونه از این شی `bean` دارد. اگر نرم افزار شما به متد `create()` دیگری نیاز دارد آنرا با یک آرگمان ورودی جدا بسازید. به علاوه متد `findByPrimaryKey()` که برای جستجو با کلید است شما آزاد هستید سایر متدهای مورد نیاز را تعریف کنید. برای مثال `findByship()` که تمام کابین ها در یک کشتی را برمیگرداند. چنین متدهای جستجو و نظیر آنها فقط رد `entity bean` استفاده میشود و در `session`, `message driven` استفاده نمی شوند.

:Bean Class EJB 2.0

دراین جا یک `entity bean` واقعی را داریم البته در قسمتهای بعد آنرا کامل خواهیم کرد و می خواهیم نشان دهیم که چگونه این قسمتها با هم ترکیب میشود.

```
import javax.ejb.EntityContext;
public abstract class CabinBean implements javax.ejb.EntityBean {
    // EJB 1.0: return void
    public CabinPK ejbCreate(Integer id){
        setId(id);

        return null;
    }
    public void ejbPostCreate(int id){
        // do nothing
    }
    public abstract String getName();
    public abstract void setName(String str);
    public abstract int getDeckLevel();
    public abstract void setDeckLevel(int level);
    public abstract Integer getId( );
    public abstract void setId(Integer id);
    public void setEntityContext(EntityContext ctx){
        // not implemented
    }
    public void unsetEntityContext(){
        // not implemented
    }
    public void ejbActivate(){
        // not implemented
    }
    public void ejbPassivate(){
        // not implemented
    }
}
```

```

public void ejbLoad(){
    // not implemented
}
public void ejbStore(){
    // not implemented
}
public void ejbRemove(){
    // not implemented
}
}

```

CabinBean به صورت انتزاعي داراي متدهاي براي براي دستيابي و به روز رسانی EJB در حالت پايدار ميباشد . نکته قابل توجه اين جاست که در کلاس ما فيلدهاي براي نگهداري اطلاعات نداريم و اين به دليل است که ما در حال کار با entity bean مدیریت شده توسط ظرف EJB 2.0 آن هستيم . در اين روش در bean فقط تعريفهاي دستيابي متدها موجود است البته در دژشد هاي ديگر مثل session ,... چنين چيزي نخواهيد دید .

:Bean Class EJB 1.0

کدهاي زير براي همان کلاس بالا امادر EJB 1.0 که فيليد ها درآن تعريف شده ميباشد.

```

import javax.ejb.EntityContext;
public class CabinBean implements javax.ejb.EntityBean {
    public Integer id;
    public String name;
    public int deckLevel;
    // EJB 1.0: return void
    public Integer ejbCreate(Integer id){
        setId(id);
        return null;
    }
    public void ejbPostCreate(Integer id){
        // do nothing
    }
    public String getName(){
        return name;
    }
    public void setName(String str){
        name = str;
    }
    public int getDeckLevel(){
        return deckLevel;
    }
}

```

```

public void setDeckLevel(int level){
    deckLevel = level;
}
public Integer getId( ){
    return id;
}
public void setId(Integer id){
    this.id = id;
}
public void setEntityContext(EntityContext ctx){
    // not implemented
}
public void unsetEntityContext(){
    // not implemented
}
public void ejbActivate(){
    // not implemented
}
public void ejbPassivate(){
    // not implemented
}

```

مقایسه کدهای EJB 1.0 , 2.0 :

متدهای set , get برای نام و سطح دکل با رابط Remote این کلاس یعنی CabinRemote سازگار میباشند .
این متدهای set , get فقط برای نرم افزار سرویس گیرنده و سایر متدها فقط برای ظرف EJB و کلاس Bean قابل دید هستند.

متدهای ejbPostCreate() , ejbCreate() برای مقدار دهی کلاس bean وقتی که يك رکورد کابین به پایگاه داده اضافه میشود می باشند .
هفت متد آخر کلاس CabinBean ورژن EJB 2.0 در رابط javax.ejb.EntityBean تعریف شده میباشند و متدهای برای مدیریت حالتها می باشند. وقتی که bean اجرا میشود ظرف متد مربوط به آن رخداد را اجرا میکند . برای مثال ejbRemove() به نکته اشاره میکند که يك entity bean داده هایش باید از پایگاه داده حذف شود.

`ejbLoad()` , `ejbStore()` در حالي اجرا مي شود كه مي خواهيم داده در پايگاه داده خوانده يا نوشته شود. هنگام فعال و غير فعال شدن فرايند مربوط به `CabiBean` (گرفتن يا آزاد سازي منابع) متدهاي `ejbPassivate()` , `ejbActivate()` اجرا مي شوند. `setEntityContext()` برقراري كردن ارتباط به `EJB container` كه اجازه گرفتن اطلاعات درباره خود `bean` را درظرف ميدهد . `unsetEntityContext()` توسط ظرف براي گرفتن فضاي حافظه اشغال شده توسط `bean` درهنگام خروج از ظرف مي باشد .

Primary Key

يك اشاره گر كمك كننده به پيدا كردن داده ها كه شرحي از يك ركورد يا موجوديت داخل پايگاه داده است وبا متد `findByPrimaryKey()` از رابط `home` محل يك موجوديت پيدا ميشود. در `Cabin EJB` داده بانوع `java.lang.Integer` براي كليد اصلي استفاده ميشود

تفاوت session beans با entity bean

`CabinBean` مثال ما يك `entity bean` مي باشد اما يك `session bean` خيلي با آن متمايز نيست. در آن بايد `SessionBean` به جاي `EntityBean` دركد `extend` شودو متد `ejbCreate()` آنرا مقدار دهني مي كند اما `ejbPostCreate()` ندارد .

`session bean` داراي متدهاي `ejbLoad()` , `ejbStore()` نيست چون اين نوع `bean` ها نامانا هستند . `session bean` داراي متد `setSessionContext()` مي باشد اما `unsetSessionContext()` ندارد . متد `ejbRemove()` در `session bean` براي ثبت اتمام ارتباط و مثل کاربرد آن در `entity bean` نيست . در `session bean` كليد اصلي نيز وجود ندارد چون قرارنيست به كمك آن به پايگاه داده ارجاع كنيم ودرحالت كلي اطلاعات `session bean` ناماندگار است .

تفاوت message driven beans با entity bean

اين نوع `bean` هيچ گونه رابط كامپوننت براي `home` , `local` , `remote` ندارد و فقط بايد چند متد خاص را فراخواني كرد . `ejbCreate()` هنگامي يك نمونه از كلاس `bean` را ايجاد ميكنيم و `ejbRemove()` زماني كه `bean` ميخواهد از سرور خاج شود فراخواني ميكنيم

متد `onMessage()` از طرف ظرف `bean` زماني كه يك پيغام براي آن ميرسد فراخواني ميشود و متد `setMessageDrivenBeanContext()` برقراري كردن ارتباط به `EJB container` كه اجازه گرفتن اطلاعات درباره خود `bean` را درظرف ميدهد . `message driven bean` متدهاي `ejbPassivate()` / `ejbActivate()` يا `ejbLoad()` / `ejbStore()` پشتيباني نمي كند چون نيازي به آنها ندارد .

همانند session bean کلاس message driven bean در خود Primary Key ندارد.

شرح دهنده های نصب و ارتقاء (Deployment Descriptors)

در اجرای bean ها يك سري اطلاعات در مورد مدیریت آن bean در زمان اجرا نیاز داریم که در کلاسها و رابط های گفته شده معین نکرده ایم بطور مثال نحوه تعامل با سرویسهای امنیتی ، تراکنش ، نام و سایر سرویس ها . برای این کار از deployment descriptors استفاده می شود .

مزیت اصلی deployment descriptors در این است که بدون تغییر در کد میتوان حالت نرم افزار را در زمان اجرا تغییر داد. و کاربرد آن شبیه فایل های property در Visual Basic است که نحوه Customize کردن نرم افزار بدون تغییر آن است.

اکثر IDE های Java از deployment descriptors پشتیبانی میکنند و به طور خود کار آن را تولید میکنند.

بسته بندی Packaging:

میتوان کامپوننتهای J2EE را در داخل بسته های نرم افزاری جدا گذاشت و در جاهای دیگر از آنها استفاده کرد(با deploy کردن) بسته ها شامل فایل های وابسته نیز مثل کلاسهای مگید و gif, html میباشد.

بسته ممکن است از یک یا چند Bean وب و Enterprise بوده به همراه شرح دهنده (descriptor) که یک سند XML که تنظیمات نصب (deploy) کردن کامپوننت در داخل آن نوشته شده می باشد.

بسته ها با پسوند EAR (Enterprise Archive) بسته های برای کامپوننتهای Enterprise هستند که فایل های EAR یک استاندارد از بایگانی فایل جاوا JAR (Java Archive) می باشند

بسته ها با پسوند WAR (Web Archive) یک شرح دهنده نصب برای فایل کامپوننت های وب و منابع وابسته می باشد . از نوع JAR در بسته های سمت Client استفاده میشود .

در مدل EAR میتوانید چند عدد از کامپوننتهای داخل EAR را بطور جدا با تنظیمات متفاوت نصب کنید.

در زیر deployment descriptors مربوط به کلاس CabinEJB رامشاهده میکنید.

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
EnterpriseJavaBeans 2.0//EN" "http://java.sun.com/j2ee/dtds/ejb-
jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>CabinEJB</ejb-name>
      <home>CabinHomeRemote</home>
      <remote>CabinRemote</remote>
      <local-home>CabinHomeLocal</local-home>
      <local>CabinLocal</local>
      <ejb-class>java.lang.Integer</prim-key-class>
```

```

<persistence-type>Container</persistence-type>
<reentrant>False</reentrant>
</entity>
</enterprise-beans>
</ejb-jar>

```

deployment descriptors از XML يك فايل متني كه ساختار آن بر اساس DTD(Document type Definition) استاندارد EJB DTD شرکت سان میباشد.

EJB Home

زمانی که يك enterprise bean در داخل ظرف نصب میشود متد های داخل Home کمک میکنند به Container به مدیریت EJB bean در چرخه زندگی آن و رابطه نزدیکی با ظرف دارد به طوری که مسئول تخصیص ، ایجاد و حذف EJB bean است .

برای مثال وقتی متد create() روی رابط Home فراخوانی میشود EJB home يك شی EJB object ایجاد کرده و اشاره گر را برمیگرداند ویا وقتی ejbCreate() در entity bean اجرا میشود يك reference به client میدهد. اکنون client میتواند با کمک stub آن شی را فراخوانی کند.

استفاده از Enterprise Bean

حال به نحوه استفاده سرویس گیرنده از يك enterprise bean می پردازیم . يك کابین شی یا موجودیتی است که شرح آن در داخل پایگاه داده ذخیره میشود و برای درک بهتر سایر entity bean ها مثل کشتی ، بلیت ، مشتری و کارمند را در نظر بگیرید .

گرفتن اطلاعات از يك entity bean

تصور کنید که سرویس گیرنده به کمک رابط گرافیکی خود باید اطلاعات يك سفر دریایی را به کاربر نمایش دهد و این اطلاعات شامل نام گشت یا سفر ، نام کشتی و لیست کابینهای آن میباشد . با استفاده از شماره گشت که فیلد متنی است میتوانیم از آن در entity bean به عنوان Primary Key استفاده کرده و داده ها را بازیابی کنیم. در زیر کدها یمشابه را ملاحظه میکنید.

```

CruiseHomeRemote cruiseHome = ... use JNDI to get the home
// Get the cruise id from a text field.

```

```

String cruiseID = textFields1.getText();
// Create an EJB primary key from the cruise id.

```

```

Integer pk = new java.lang.Integer.parseInt(cruiseID);
// Use the primary key to find the cruise.

CruiseRemote cruise = cruiseHome.findByPrimaryKey(pk);
// Set text field 2 to show the cruise name.

textField2.setText(cruise.getName());
// Get a remote reference to the ship that will be used
// for the cruise from the cruise bean.

ShipRemote ship = cruise.getShip();
// Set text field 3 to show the ship's name
.
textField3.setText(ship.getName());
// Get a list of all the cabins on the ship as remote references
// to the cabin beans.

Collection cabins = ship.getCabins();

Iterator cabinItr = cabins.iterator();
// Iterate through the enumeration, adding the name of each cabin
// to a list box.

while( cabinItr.hasNext())

    CabinRemote cabin = (CabinRemote)cabinItr.next();

    listBox1.addItem(cabin.getName());

}

```

در ابتدا با گرفتن يك reference يا اشاره گر به EJB home از entity bean با نام گشت درياي (cruise) که از JNDI (Java Naming and Directory Interface) استفاده میکند. JNDI يك سري API پر قدرت براي پيدا کردن منابع و اشیاء روي شبکه مي باشد و کمي پیچیده میباشد به همین دلیل در قسمتهای بعدی توضیح خواهیم داد. زمانی که شماره گشت cruiseID از کاربر گرفته میشود از آن در ایجاد کلید اصلي entity bean استفاده میشود. با CruiseRemote و pk آن گشت پیدا شده و نام گشت را به کاربر نمایش میدهد.

در گشت میتوان اشاره گر به کشتی آن را گرفت و در شی با نوع ShipRemote گذاشت سپس به کمک آن نام کشتی را بازیابی کرد. همچنین نام کابین های کشتی به کمک نام کشتی بازیابی و در حلقه while نشان داده. اینها این واقعیت را می رسانند که مدل داده entity bean يك سیستم با رابط پایدار به داده های پایگاه داده ارائه میدهد مثلاً relationship یا ارتباط منطقی بین گشت درياي و کشتی یا کشتی و کابین های آن که به راحتی با ship.getCabins() این عمل انجام میشود.

مدل سازي جريان كار با session bean

Entity bean مناسب براي مفاهيم و داده هاي تجاري بوده در يك نام ميتوانند خلاصه شوند اما آنها براي ارائه يك فرآيند با وظيفه مناسب نبوده . يك ship bean متدهاي براي كار باشي كشتي دارد اما نمي تواند ضمينه تعريف كند كدام اعمال انجام شده يا نشده

فصل هشتم

J2ME

تکنولوژی (Java 2 Platform Micro Edition (J2ME زیر ساخت جاوا برای اجرای برنامه های مخصوص جاوا روی دستگاه های کوچک همانند تلفن های سلولی (موبایل)، کامپیوترهای کوچک همراه (PDA) Personal Digital Assistant و سایر سیستم های جاسازی شده است. زیر ساخت J2ME شامل یک پیکره بندی، یک نمایه (profile) و یک بسته اختیاری است.

در بسته موبایل NetBeans IDE شما می توانید بسیاری از جنبه های فرآیند توسعه در بسته موبایل Mobile Information Device Profile (MIDP) را به ساده گی انجام دهید. این بسته شامل یک طراح ویژوال MIDP برای محتوا و روند صفحه برنامه، همچنین پشتیبانی از ویرایشگر مخصوص آن دستگاه و حتی فرآیند توسعه برنامه برای یک سرویس دهنده راه دور است.

بمنظور استفاده از قابلیت های موبایل جاوا شما باید بسته مخصوص را نصب کنید. اگر از NetBeans IDE 4.1 استفاده می کنید شما باید این بسته را دانلود کرده و از نصب کننده add-on آن استفاده کنید. در ورژن 5 این IDE این بسته را می توان هنگام نصب بطور خودکار در داخل IDE قرار داد. قابلیت های J2ME هم اکنون قسمتی از J2SE API شده. در زیر تعدادی از اصلاحات در باره برنامه نویسی موبایل را شرح داده ایم.

CLDC : این کلمه مخفف Connected Limited Device Configuration است و در J2ME بمنظور پیکره بندی انواع موبایل های مختلف استفاده می شود. این پیکره بندی شامل یک محیط زمان اجرا و API های بسته مخصوص پردازنده ها با سرعت و حافظه محدود دستگاه های موبایل است. همان طور که می دانید دستگاه های موبایل توسط شرکت های مختلف تولید می شوند و در داخل آنها هر شرکت IC پردازنده خاص که خود آنها را ساخته اند به کار می برند و فقط در مدارات میانی و ارتباطی از IC های استاندارد بازار استفاده می کنند. پس در اینجا نیاز داریم که برای انواع موبایل های مختلف یک استاندارد توسعه داشته باشیم. این موضوع همانند کامپیوترها با زیر ساختها و برنامه های مختلف است، و ماشین مجازی جاوا این مشکل را حل کرده. پس هر دستگاه موبایل CLDC مخصوص خود را دارد.

MIDP: نمایه اطلاعات دستگاه موبایل (Mobile Information Device Profile) یک مجموعه API به منظور ارائه قابلیت های سطح بالا مورد نیاز برای برنامه های موبایل است. این قابلیت ها می تواند شامل اجزاء قابل نمایش (مثل screen) و ارتباطات شبکه است.

MIDlet: یک کلاس می باشد که مورد نیاز تمام برنامه های MIDP است. در عمل MIDlet یک رابط بین برنامه و دستگاه موبایلی که برنامه در روی آن در حال اجرا است می باشد. می توان گفت که MIDlet شبیه main class در پروژه های J2SE است.

Preverification: زمانی که يك برنامه با CLDC اجرا مي شود تمام کلاسهاي کامپایل شده بايد پالایش شوند. این عمل يك فرآیند براي اضافه کردن تفسیر اضافي به کماک ماشین مجازي CLDC به يك فایل بایت کد کلاس است. فرآیند Preverification این اطمینان را به وجود مي آورد که محتوای کلاس فقط کدهای اجرا شوند در آن CLDC JVM باشند.

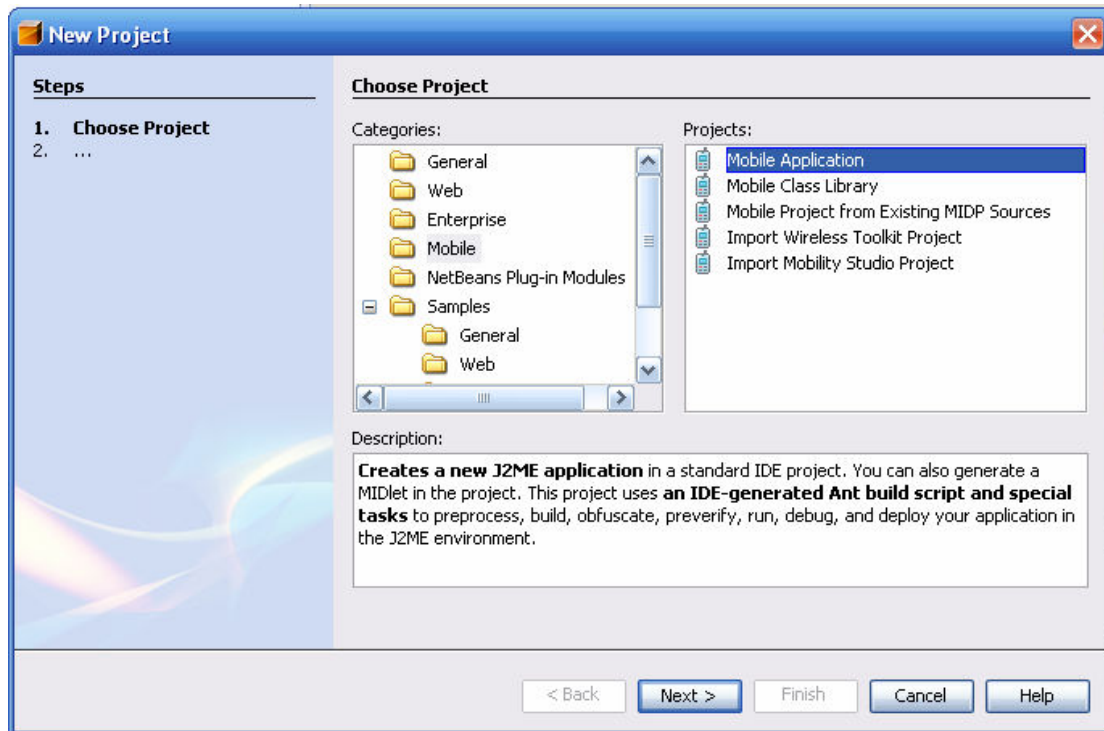
Device Fragmentation: این عبارت براي گوناگوني زیرساختهاي مختلف موبایلها که از داشتن يك مشخه برنامه نویسي اجرا شونده بر روی تمامی موبایلها جلو گيري مي کند است. این گوناگوني ها مي تواند بصورت فیزیکی مثل اندازه و رنگ صفحه نمایش یا اندازه حافظه باشد، و یا بصورت نرم افزاري مثل API هاي در دسترس و ورژنهای CLDC/MIDP باشد.

Preprocessor: پردازنده در بسته موبایل NetBeans به شکل يك وظیفه (task) است که فایلهاي کامپایل شده J2ME را اجرا مي کند.

Obfuscation: فرآیندي که فایلهاي کلاس را براي مهندسي معکوس (جلوي گيري از استفاده غير قانوني از کدهای برنامه شما) مشکل مي کند. این به طور معمول با جایگزینی نامهاي بسته ها، کلاسها، متدها و فیلدها با شناسه هاي عديدي است. این کار یکی از جنبه هاي فرآیند تولید نرم افزار موبایل مي باشد و نتیجه این کار کاهش اندازه برنامه است.

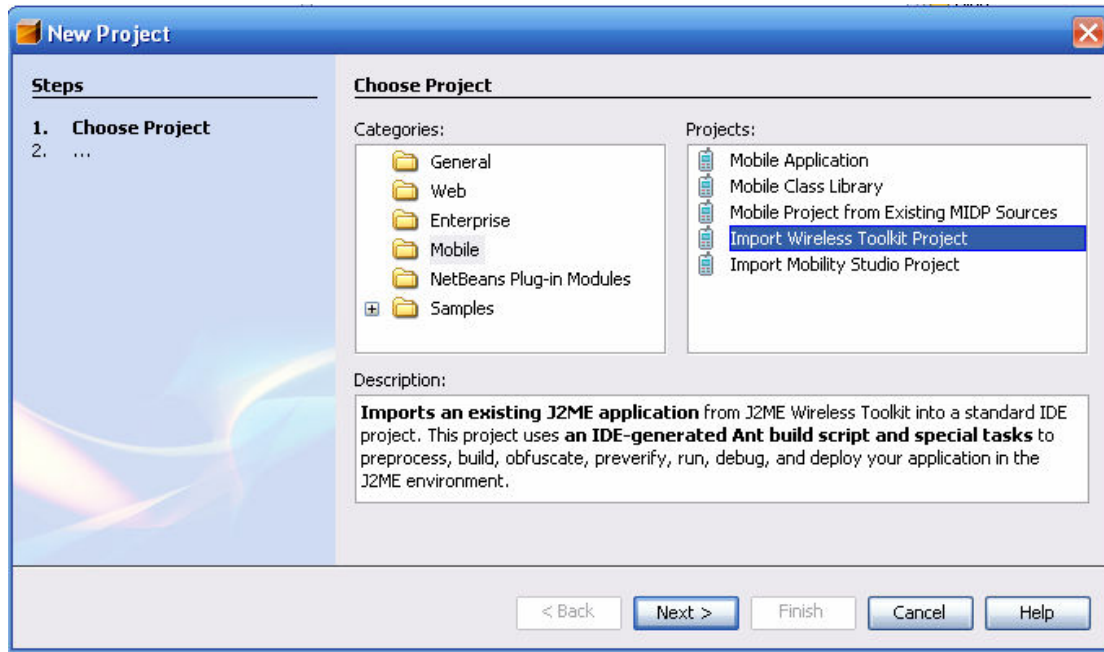
در زمانی که يك پروژه براي موبایل ایجاد مي کنید IDE يك مجموعه از کارها را انجام مي دهد. این کارها شامل انتخاب يك شبیه ساز پروژه، مسیر فایل کلاسهاي برنامه و ایجاد دستورات پيش فرز براي اجرا مي باشد. براي ایجاد يك پروژه موبایل گامهاي زیر را انجام دهید:

- از منوي فایل گزینه New Project را انتخاب کنید.
- در درخت اقلام فهرست Mobile را انتخاب کنید.
- در زیر درخت يك از دو گزینه Mobile Class Library یا Mobile Application را انتخاب کنید. گزینه Mobile Application يك الگوي (template) از MIDlet بطور خودکار را براي شما ایجاد مي کند. گزینه دوم هيچ کلاسي براي شما ایجاد نمي کند و شما خود بايد کلاسها را ایجاد کنید. مثالهاي MIDP و Bluetooth (پرتکول ارتباط دستگاهاي جانبي کوچک) از Visual Designer استفاده مي کنند.



- دکمه Next را کلیک کنید.
- به طور اختیاری قسمتهای زیر را می توانید برای نام و محل پروژه در ویزارد وارد کنید. در قسمت Project Name نام پروژه در رابط کاربر IDE را وارد کنید و در قسمت Project Location محل پروژه را در سیستم خود وارد کنید.
- به طور اختیاری می توانید جعبه checkbox ایجاد Hello MIDlet را غیر فعال کنید. این عمل از ایجاد مثال جلوگیری می کند.
- در این قسمت می توانید شبیه ساز و پیکره بندی مخصوص موبایلی که برنامه را برای آن توسعه می دهید، را انتخاب کنید. این کار سبب می شود در هنگام اجرای برنامه، برنامه شما بر روی شبیه ساز و دستگاه موبایل مورد نظر اجرا شود. توجه داشته باشید که این تنظیمات را بعد از ایجاد پروژه نیز می توانید تغییر دهید.
- در پایان گزینه Finish را کلیک کنید.

این قابلیت وجود دارد که به پروژه خود پروژه های J2ME Wireless Toolkit، پروژه های موبایل NetBeans و سایر منابع MIDP را وارد (import) کنید. برای کار بعد انتخاب New Project از منوی فایل در زیر درخت اقلام فهرست Mobile گزینه Import Mobility Studio Project یا Import Wireless Toolkit Project را انتخاب کنید. سپس محل فایل های این پروژه ها را مشخص کنید. این قسمت در شکل زیر نشان داده شده.



بعد مشخص کردن پروژه بر روی Next کلیک کنید. بقیه مراحل همانند ایجاد پروژه جدید گفته شده در قسمت قبل است.

پیکره بندی پروژه

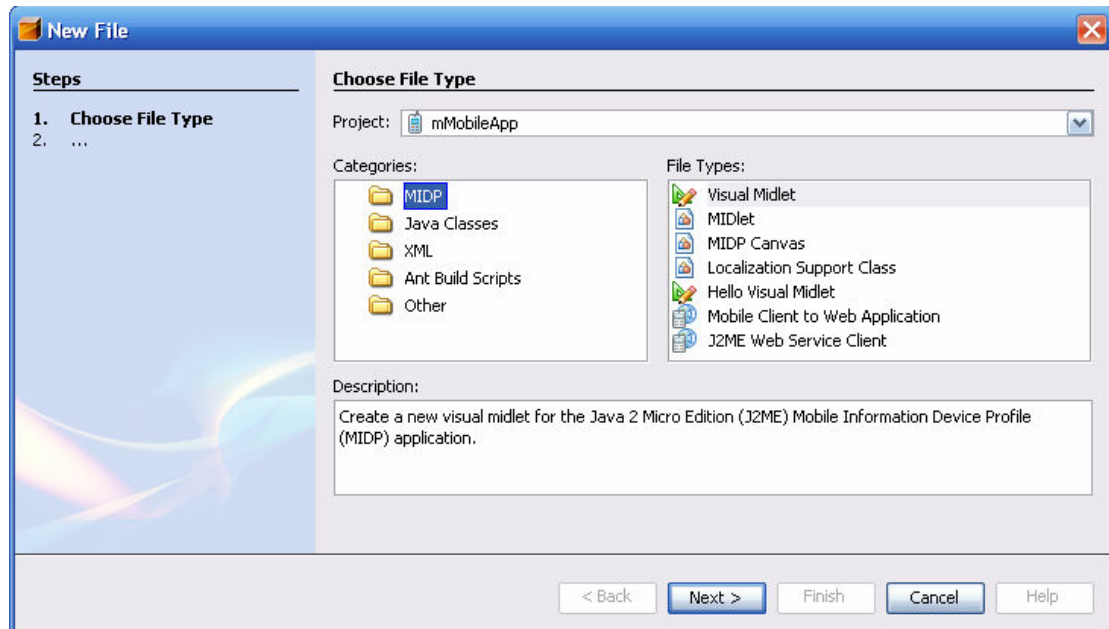
شما قادر به امتحان ساختار فیزیکی پروژه خود با استفاده از پنجره Files می باشید. زمانی که یک پروژه موبایل را ایجاد می کنید، بطور خودکار IDE فایلها و فهرستهای لازم را که برای همه پروژه ها ایجاد می کند را به جز فهرست test می سازد. اگر تستها ایجاد شده باشند بسته و کلاس آنها در فهرست test قرار خواهند داشت. فهرست build و dist زمانی که پروژه را ایجاد می کنید ساخته خواهند شد. در داخل فهرست build فهرستهای زیر موجود است:

- compiled: این فهرست شامل تمام کلاسهای کامپایل شده است.
- preprocessed: این فهرست ورژنهای از قبل پردازش شده فایلهای منبع را در خود دارد. اگر شما از پیکره بندی استفاده می کنید این فایلها از منبع های اصلی متمایز هستند.
- obfuscated: این فهرست ورژنهای obfuscated از فایلهای کلاس شما نگهداری می کند.
- preverified: نگهداری کننده ورژنهای preverified از کلاس فایلها شما است.

بسته بندی فایلهای کلاس در فایلهای JAR توزیع می شوند. این فایلها سرعت فرایند توسعه را افزایش داده و با اطلاعاتی که به شما می دهند اشکالات فرایند توسعه که ممکن است در داخل برنامه شما ایجاد شوند را در انزوا قرار می دهند. فرایند توسعه در NetBeans از استاندارد Ant برای مدیریت کلاسهای ایجاد شده موقت استفاده می کند. معنای این کار این می باشد که فایلهای منبع با بیش از چند برچسب زمانی (تغییرات در زمانهای مختلف) از کلاسها فایلها در فهرست build زمانی که دستور Build اجرا می شود بطور خودکار دوباره ایجاد خواهند شد. همچنین در زمان حذف کردن یک فایل منبع از پروژه بطور خودکار فایلهای کلاس وابسته نیز از فهرست build حذف خواهند شد. در فهرست dist از پروژه فایلهای JAR, JAD نگهداری می شوند.

استفاده از الگوهای فایل موبایل

فرآیند ایجاد فایلها در پروژه های موبایل شبیه بهایجاد فایلها در پروژه های عمومی جاوا است. الگوی ایجاد کلاسهای MIDP داخل فهرست MIDP از الگوها در ویزارد New File می باشد، این فهرست در شکل زیر نمایش داده شده. برای باز کردن ویزارد New File از منوی File گزینه New File را انتخاب کنید.



MIDP Canvas

بمنظور ایجاد MIDP Canvas مراحل زیر را انجام دهید.

- در لیست File Types گزینه MIDP Canvas را انتخاب کنید و Next را کلیک کنید.
- بطور اختیاری می توانید در قسمت MIDP Class Name نام کلاس جدید را وارد کنید. همچنین در قسمت Package بسته ای که کلاس جدید در آن قرار دارد را وارد کنید.
- دکمه Finish را کلیک کنید.

MIDlet

ویزارد ایجاد فایل MIDlet کمی متفاوت می باشد و این تفاوت در گرفتن اطلاعاتی که در فایل JAD برنامه استفاده می شود است.

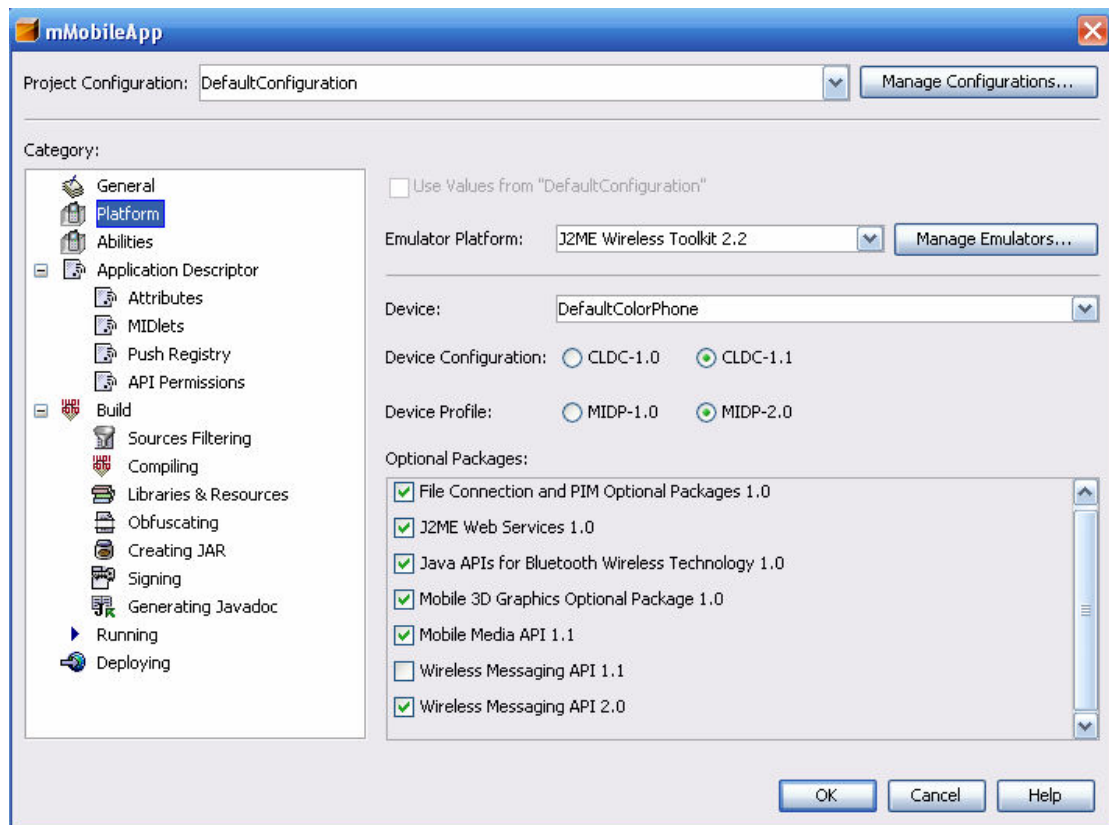
- در لیست File Types گزینه MIDlet را انتخاب کنید و Next را کلیک کنید.
- بطور اختیاری در قسمت می توانید فیلدهای نام و محل در ویزارد را تغییر دهید. فیلد MIDlet Name مقداری می باشد که در زمان شروع نمایش برنامه نشان داده خواهد شد. این نام جدا از نام کلاس خواهد بود. فیلد MIDP Class Name برای نام کلاس جدید MIDlet استفاده می شود. برای وارد کردن محل icon کلاس MIDlet از فیلد MIDlet Icon استفاده کنید. فیلد Package بسته ای که کلاس جدید MIDlet در آن قرار دارد را مشخص می کند.
- در نهایت روی دکمه Finish کلیک کنید.

پیکره بندی پروژه

شبيه به اغلب پروژه ها تنظيمات كلاسهاي كه در كامپايل پروژه استفاده مي شود در Project Properties مي باشد. دو پانل Libraries&Resources و Platform به طور ويژه براي كنترل كلاسها هستند. به علاوه با اين تنظيمات كامپايل كد نهاي كنترل مي شود.

تغيير شبيه ساز پروژه انتخاب زيرساخت شبيه ساز براي تعيين كتابخانه هاي در هنگام كامپايل پروژه استفاده مي شوند مي باشد. به صورت پيش فرز پيكره بندي پروژه از آخرين ورژن J2ME Wireless Toolkit استفاده مي شود. شما مي توانيد در پانل Platform اين تنظيمات را تغيير دهيد. گامهاي زير را براي اين كار دنبال كنيد.

- در پنجره Project بر روي پروژه راست كليك کرده و Properties را انتخاب كنيد.
- بر روي گره Platform كليك كنيد. اين پنجره در شكل زير به نمايش در آمده.



- به طور اختياري هر کدام از فيلدهاي زير را كه كتابخانه شبيه ساز از آن استفاده مي كند مي توانيد تغيير دهيد.
- Emulator Platform: زير ساخت نصب شده براي شبيه ساز را تعيين مي كند. مقدار هاي اين كنترل در فيلدهاي زير قابل تغيير مي باشد.
- Device: توسط اين گزينه مي توانيد شكل دستگاه موبايل براي اجراي برنامه را انتخاب كنيد.
- Device Configuration: پيكره بندي دستگاه كه بروي كلاسهاي برنامه تاثير گذار است را مشخص مي كند.

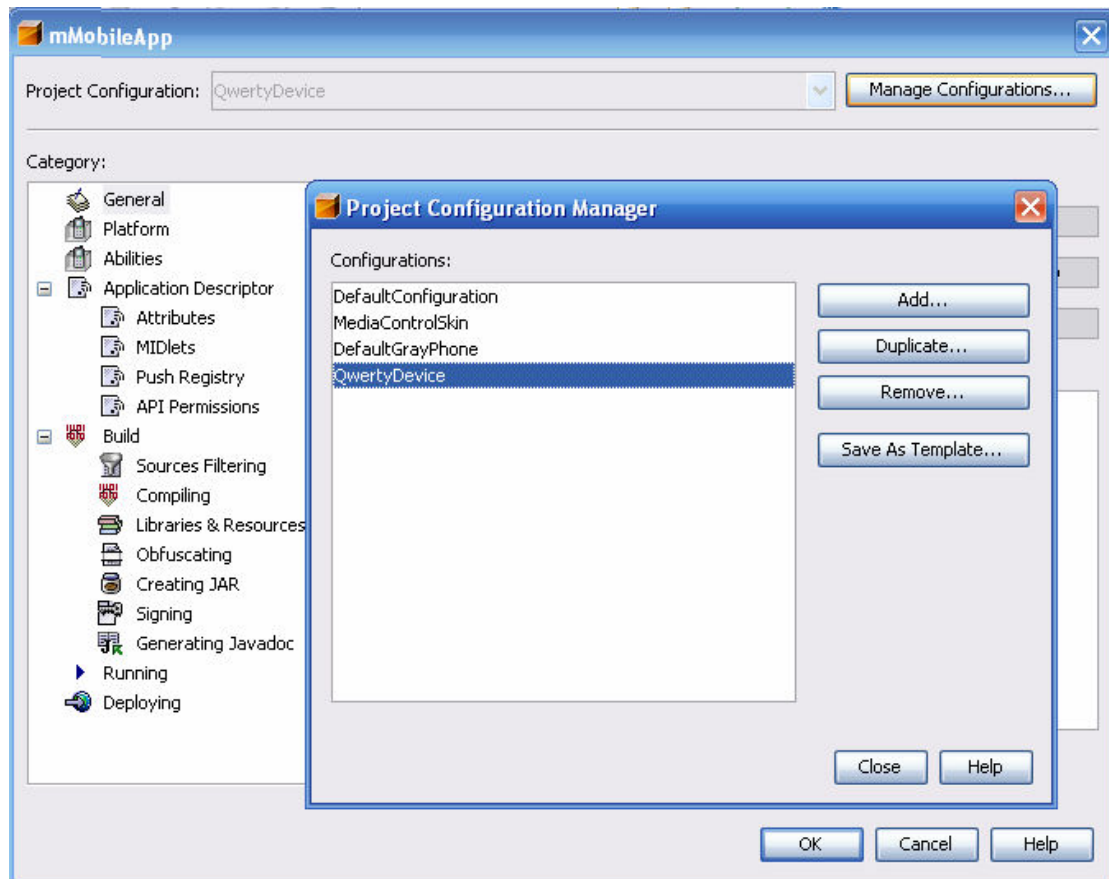
Device Profile: نمایه دستگاه موبایل (Mobile Information Device Profile) که يك مجموعه API به منظور ارائه قابلیت‌های سطح بالا مورد نیاز برای برنامه های موبایل است را مشخص می کند. این قابلیت‌ها می تواند شامل اجزاء قابل نمایش (مثل screen) و ارتباطات شبکه است. **Optional APIs:** در قسمت می توانید بسته های اختیاری که می توانید به برنامه خود اضافه کنید را انتخاب کنید.

پیکره بندی پروژه برای دستگاه های مختلف

یکی از جنبه های توسعه برنامه های موبایل قابلیت اجرای آنها بر روی انواع دستگاه ها است و این به يك چالش تبدیل شده است. دستگاه های مختلف دارای تفاوت‌های فیزیکی مثل اندازه صفحه و اندازه حافظه و غیره دارند. پس نیاز به پیکره بندی های مختلف دارند. برای پیکره بندی از برگه General در Properties پروژه استفاده می شود. توجه داشته باشید که تفاوت اصلی برگه Project Properties در پروژه های موبایل و پروژه های عمومی جاوا در جعبه ترکیبی با برچسب Project Configuration در قسمت بالا است. این جعبه ترکیبی شامل يك عنصر برای هر پیکره بندی در پروژه شما است. در حالت کلی شما باید يك پیکره بندی برای هر فایل JAR توزیع شده ایجاد کنید. برای مثال اگر شما قصد دارید يك برنامه با سه نوع اندازه صفحه و استفاده از دو مجموعه از API ایجاد کنید باید شما شش پیکره بندی را ایجاد کنید.

ایجاد پیکره بندی

- بعد از باز کردن برگه Properties پروژه بر روی General کنید.
- سپس بر روی کمه Manage Configuration کلیک کنید. بعد از این کار جعبه محاوره ای پیکره بندی باز خواهد شد. این برگه را در شکل زیر مشاهده می کنید.



- برای افزودن پیکره بندی بر روی دکمه Add کلیک کنید.
- در قسمت Name نام پیکره بندی جدید را وارد کنید و بر روی OK کلیک کنید.
- برای مثال نام NokiaSeries40Bluetooth یا HighResNoSizeLimit هر توزیع از پیکره بندی را مشخص می کند. توجه داشته این نام دلخواه است.

ایجاد پیکره بندی دلخواه

دو قاعده کلی در پیکره بندی داخل برگه Project Properties وجود دارد. قاعده اول استفاده از پیکره بندی های پیش فرز و دوم وارد کردن پیکره بندی که از تنظیمات پیش فرز گرفته شده توسط کاربر. برای استفاده از تنظیمات کاربر باید checkbox با برچسب "Use Value from 'Default configuration'" در برگه Platform باید غیر فعال (unchecked) شده و برنامه از پیکره بندی داخل combo box بالا استفاده می کند. در این مورد گامهای زیر یک مثال می باشد.

- در برگه Properties پروژه پانل Platform را انتخاب کنید
- مطمئن شوید که در جعبه پائین افتادنی (drop down) مربوط به Project Configuration گزینه Default Configuration فعال شده باشد. در جعبه پائین افتادنی Emulator Platform گزینه J2ME Wireless Toolkit و در جعبه پائین افتادنی Device گزینه DefaultGreyPhone را انتخاب کنید.
- در جعبه پائین افتادنی Project Configuration نام پیکره بندی ایجاد شده در قسمت را انتخاب کنید.
- اگر پانل Platform فعال باشد خواهید دید که تمام گزینه ها غیر فعال خواهند شد. گزینه "Use Value from 'Default configuration'" را غیر فعال کنید. با این

کار تمام گزینه ها در دسترس خواهند بود، حال شما يك پیکره بندي مخصوص خود را مي توانيد تنظيم كنيد.

استفاده از MIDP Visual Designer

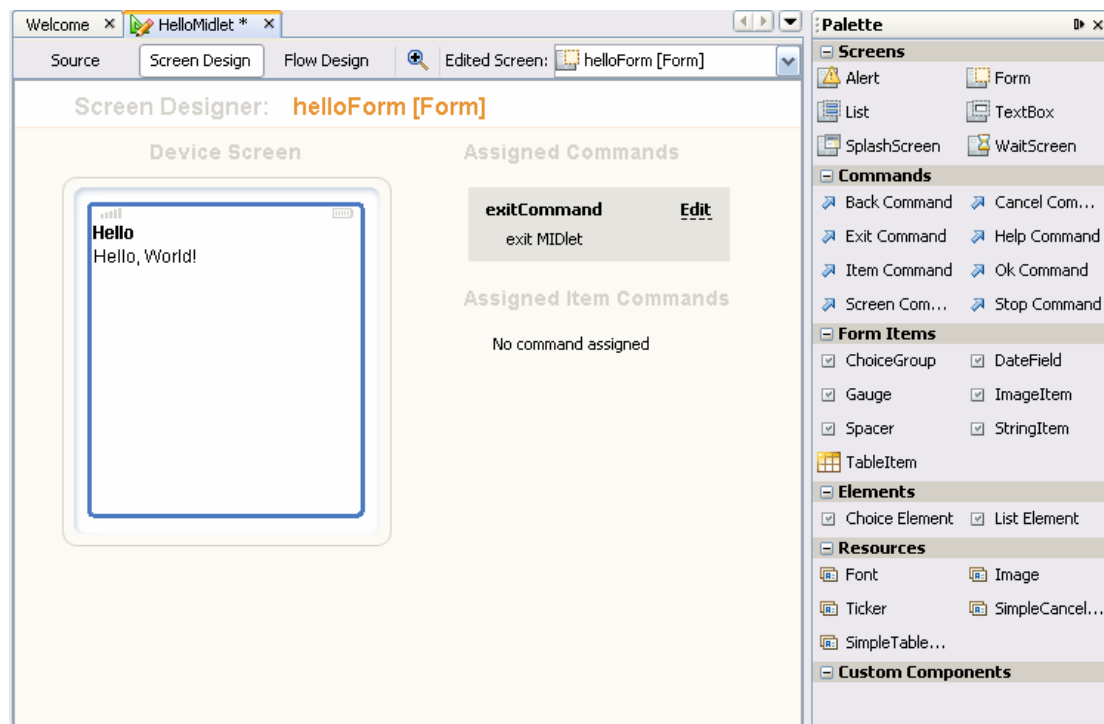
به كمك طرح ويژوال MIDP مي توانيد به طور بصري MIDlet ايجاد كنيد. اين يك محيط طراحي با تمام قابليت‌ها مي باشد كه به شما اين امكان را مي دهد كه به طور سريع برنامه ها را ايجاد کرده و محتوای صفحات را تغيير دهيد. تغيير محتوای صفحات به كمك يك GUI بصري از استاندارد MIDP 1.0 و MIDP 2.0 استفاده ميکند.

ايجاد Visual Design جديد

براي اين کار گام‌هاي زیر را دنبال كنيد.

- در پنجره پروژه بر روي بسته اي كه قصد داريد شامل Visual Design باشد كليك راست کرده و گزینه New | Visual Midlet را انتخاب كنيد. اگر اين قابليت فعال نيست مي توانيد از New | New File (Ctrl-N) از منوي اصلي استفاده كنيد.
- در جعبه ليست اقلام گره MIDP را انتخاب كنيد و سپس Visual MIDlet يا HelloMIDlet.java در ليست فايلها انتخاب كنيد. دكمه Next را كليك كنيد.
- حال ورژن MIDlet بر اساس دستگاهي كه براي آن برنامه را ايجاد مشخص كنيد. سپس Next را كليك كنيد.
- فايل‌هاي مربوط به الگو (template) از قبيل نام و پیکره بندي را وارد کرده و آنها را ذخيره كنيد سپس دكمه Finish را فشار دهيد.

بعد از مراحل بالا Visual Design در حالت Flow Design باز خواهدشد. اگر HelloMIDlet انتخاب شده باشد، اين MIDlet هم اکنون با يك جعبه متن در صفحه پيغام Hello World! را به شكل زیر نشان خواهد داد.



کد این برنامه به شکل زیر است.

```
package hello;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloMidlet extends MIDlet implements CommandListener {

    public HelloMidlet() {

        private Form helloForm;
        private StringItem helloStringItem;
        private Command exitCommand;

        private void initialize()

        {
            getDisplay().setCurrent(get_helloForm;())

            /**/   Called by the system to indicate that a command has been invoked on
            a particular displayable.
            @ *    param command the Command that was invoked
            @ *    param displayable the Displayable on which the command was
            invoked
            /*
            public void commandAction(Command command, Displayable displayable (
            }
            //    Insert global pre-action code here
            if (displayable == helloForm) {
                if (command == exitCommand) {
            //        Insert pre-action code here
                exitMIDlet();
            //        Insert post-action code here
            {
            {
            //    Insert global post-action code here
            {

            /**/
            *    This method should return an instance of the display.
            /*
            public Display getDisplay() {
                return Display.getDisplay(this);
            {
```

```

**/
*   This method should exit the midlet.
/*
    public void exitMIDlet                } ()
        getDisplay().setCurrent(null;(
        destroyApp(true;(
        notifyDestroyed();
        {

**/   This method returns instance for helloForm component and should be
called instead of accessing helloForm field directly.
@ *   return Instance for helloForm component
/*
    public Form get_helloForm} ()
        if (helloForm == null} (
//      Insert pre-init code here
        helloForm = new Form(null, new Item[] {get_helloStringItem;{()
        helloForm.addCommand(get_exitCommand;{()
        helloForm.setCommandListener(this;(
//      Insert post-init code here
{
    return helloForm;
{

**/   This method returns instance for helloStringItem component and
should be called instead of accessing helloStringItem field directly.
@ *   return Instance for helloStringItem component
/*
    public StringItem get_helloStringItem} ()
        if (helloStringItem == null} (
//      Insert pre-init code here
        helloStringItem = new StringItem("Hello", "Hello, World;(!
//      Insert post-init code here
{
    return helloStringItem;
{

**/   This method returns instance for exitCommand component and should
be called instead of accessing exitCommand field directly.
@ *   return Instance for exitCommand component
/*
    public Command get_exitCommand} ()
        if (exitCommand == null} (
//      Insert pre-init code here
        exitCommand = new Command("Exit", Command.EXIT, 1;(
//      Insert post-init code here
{

```

```

        return exitCommand;
    }

    public void startApp() {
        initialize();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}

```

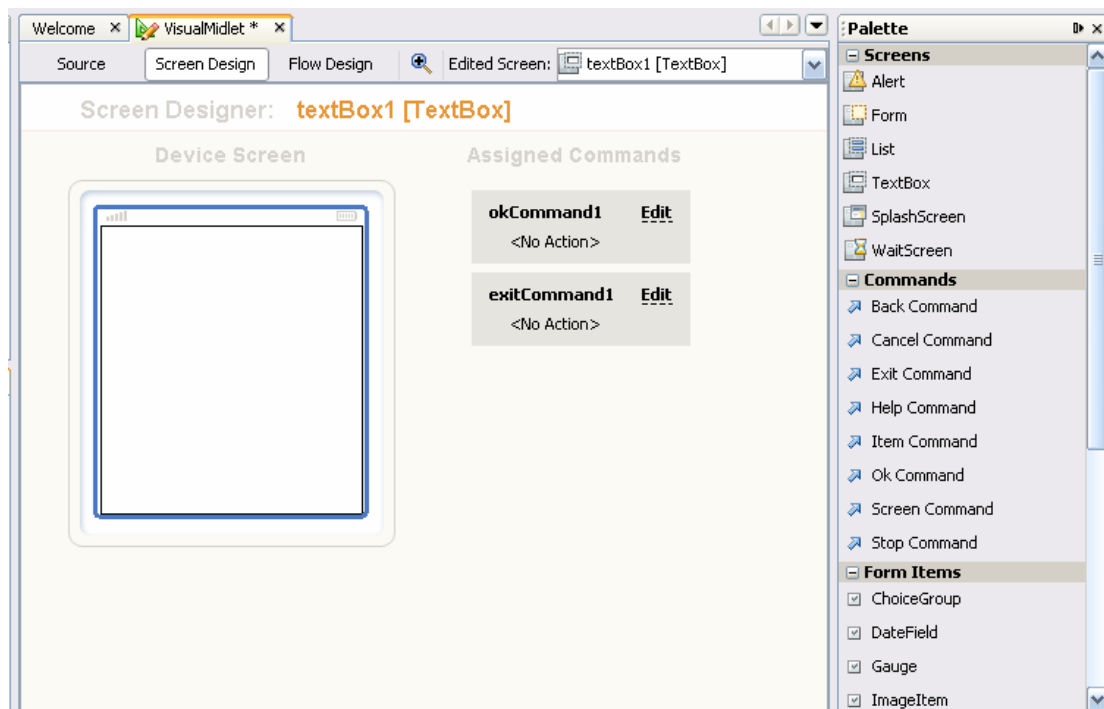
اضافه کردن اشیاء به جعبه کامپوننتها صفحات تعین شده کاربر و اقلام آنها می توانند به جعبه کامپوننتها برای استفاده اضافه شوند. البته ویراستار صفحه اجازه نمی دهد که تمام مشخصات کامپوننتهای که کاربر اضافه کرده را ویرایش کنیم، بلکه Flow Designer از آنها استفاده می کند. گامهای زیر را برای اضافه کردن دنبال کنید.

- مطمئن شوید که کلاسی که از آن قصد استفاده دارید در مسیر کلاسهای برنامه موجود است.
- در منوی Tools از زیر منوی Palette Manager گزینه Mobility Editor را انتخاب این کار جعبه مدیریت کامپوننتها را به شکل زیر باز خواهد کرد.
- بر روی دکمه Add کلیک کنید.
- پروژه ای را که شامل شی شما برای اضافه کردن است را انتخاب کنید.
- با استفاده از لیست کلاسهای پیدا شده MIDP در پروژه خود تمام اشیاء لازم برای اضافه شدن را انتخاب کنید.
- به صورت اختیاری می توانید از جعبه ترکیبی Add to Category برای انتخاب کردن جعبه اقلام استفاده کنید.
- سپس بر روی Finish و Close کلیک کنید.

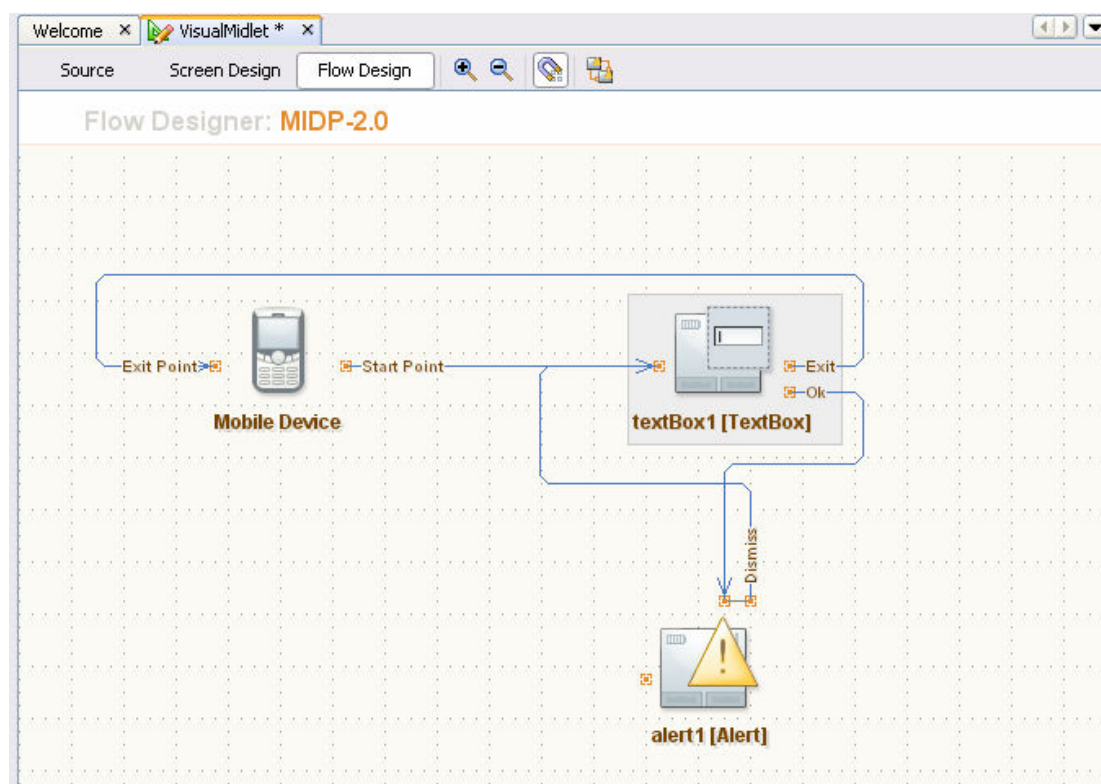
حال کامپوننتهای شما قابل دسترس در Visual Designer خواهند بود.

- ایجاد یک برنامه کوچک به کمک Visual Designer بطور مثال گامهای زیر نحوه ایجاد استخوان بندی یک برنامه با دو صفحه که می تواند برای ارسال یک SMS استفاده شود را بیان می کند.
- با کلیک راست بروی بسته پروژه و انتخاب Visual Midlet یک سند جدید Visual Design ایجاد کنید و بر روی دکمه Flow Design در میله ابزار designer کلیک کنید.
 - بر روی TextBox از جعبه کامپوننتهای Screen کلیک کرده و آن را به Flow Design بکشید (drag کنید).
 - Start Point برچسب دار شده در روی صفحه دستگاه موبایل را به جعبه متن تازه اضافه شده کلیک و درگ کنید. حال صفحه اول شما اضافه می شود. اگر شما هم اکنون برنامه را اجرا کنید برنامه با یک فیلد متنی بدون دستور اختصاصی برای آن آغاز خواهد شد.

- از قسمت Commands در جعبه کامپوننتها OK command را كليك کرده و به صفحه TextBox درگ کنید. حال به همان روش يك Exit command را نیز اضافه کنید.
- يك Alert را از جعبه کامپوننتهاي Screen كليك کرده و آن را به Flow Designer درگ کنید.
- حال صفحه Alert جدید را دابل كليك کنید. این کار آنرا در Screen Designer باز خواهد کرد.
- صفحه Device جاي كه <Enter Text> را خواهید دید كليك کنید. حال این جعبه متن قابل ویرایش خواهد بود. پیغام "SMS Sent" را وارد کنید و Ctrl-Enter را براي ذخیره کردن تغییرات فشار دهید.
- از combo box در بالاي Screen Designer براي رفتن به صفحه TextBox استفاده کنید. این صفحه در شکل زیر به نمایش درآمده.



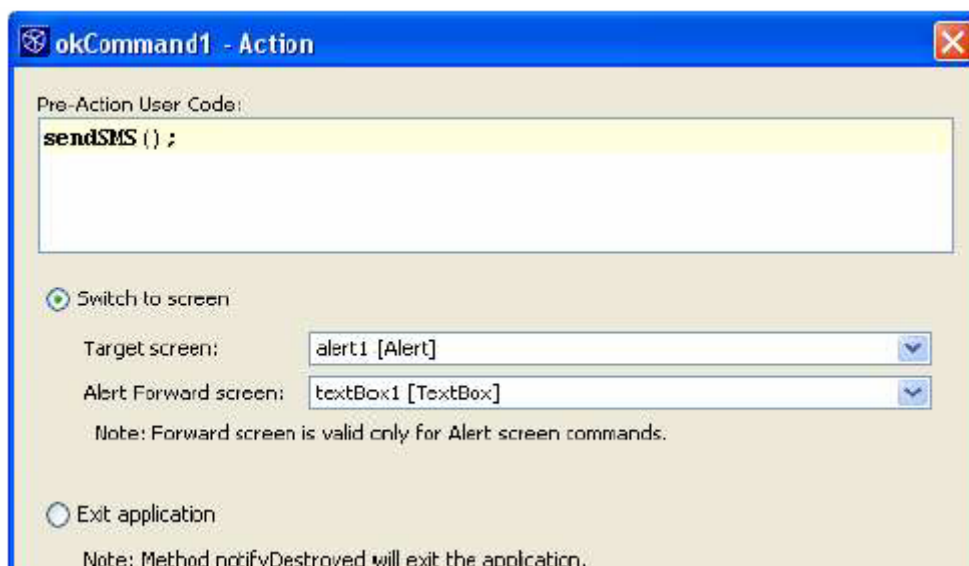
- بر روی جعبه متن كليك کرده و رشته <Edit Text> را حذف کنید.
- به كمك ویرایشگر property رشته "Enter SMS text:" را در خصیصه Title وارد کنید.
- بر روی دکمه Flow Design در میانه ابزار كليك کنید تا به قسمت Flow Design برگردید.
- از صفحه TextBox برچسب okCommand را كليك کردن و آنرا به صفحه Alert درگ کنید.
- از صفحه TextBox برچسب exitCommand را كليك کردن و آنرا به صفحه Mobile Device درگ کنید.
- پس از انجام این گامها Flow Designer شبیه به شکل زیر خواهد شد.



اگر شما در این زمان برنامه را اجرا کنید، در شروع برنامه يك جعبه متن و دو گزینه OK و Exit را نشان خواهد داد. اگر کلید OK را فشار دهید برنامه شما را به صفحه Alert خواهد برد. و اگر کلید Exit را فشار دهید از برنامه خارج خواهید شد.



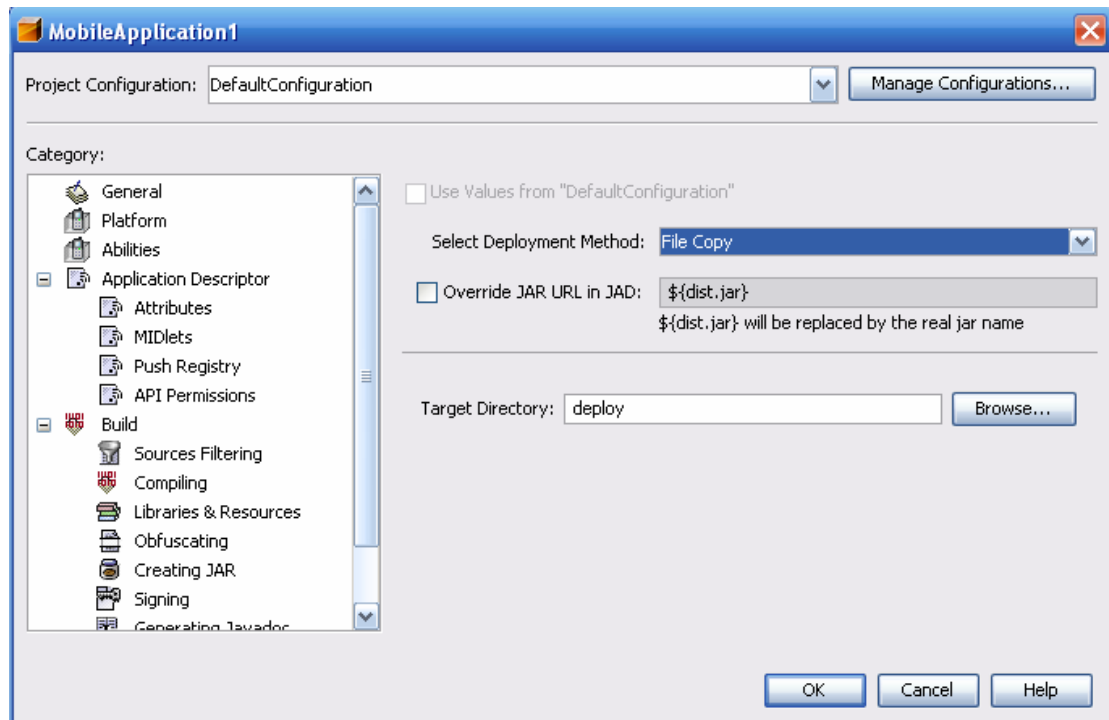
- در قسمت Properties مربوط به okCommand1 روی خصیصه Action کلیک کرده و جعبه محاوره ای Action را باز کنید.
- در جعبه محاوره ای Action در فیلد Callback Validation Method Name متن sendSMS را همان طور که در شکل زیر نشان داده شده وارد کنید.
- اگر فیلد Callback Validation Method Name در پنجره Action موجود نبود از جعبه رادیویی گزینه Exit Application را انتخاب کرده و دکمه OK را کلیک کنید. سپس به پنجره Inespector رفته و از بر روی okCommand1 کلیک راست کرده و گزینه Go To Source را انتخاب کنید. در قسمت کد در زیر شرط `if(commad==okCommand1)` به جای متد `exitMIDlet()` متد `sendSMS()` را وارد کنید و این متد را ایجاد کنید.
- این عمل متد `sendSMS()` را ایجاد خواهد کرد. دستورات این متد جای است که شما کدهای مربوط ارسال SMS را خواهید نوشت. هر زمان که دکمه OK فشرده شود این متد فراخوانی خواهد شد و اگر متد مقدار `false` را برگرداند این به این معنی است که انتقال انجام نشده.



– در میله ابزار Visual-Designer بر روی دکمه Source برای دیدن کدهای منبع برنامه کلیک کنید. کدها به طور خودکار ایجاد شده اند و متد `sendSMS()` را خواهید دید. حال شما کدهای لازم برای ارسال SMS را می توانید وارد کنید. هم اکنون شما یک برنامه ساده MIDP را ایجاد کرده اید.

Deploy کردن برنامه بصورت خودکار
چندین روش برای Deploy کردن برنامه های موبایل وجود دارد که شامل گذاشتن برنامه در یک فایل JAR است. برای این کار بهگامهای زیر را دنبال کنید.

- بر روی پروژه کلیک راست کرده و گزینه Project Properties را انتخاب کنید.
- پانل Deploying را انتخاب کنید. این پانل در شکل زیر نشان داده شده.



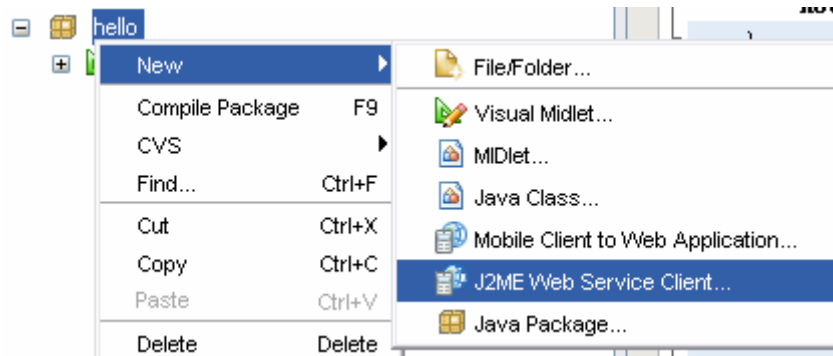
- در combo box با برجسب Select Deployment Method روشي را كه مي خواهيد از آن براي Deploy کردن استفاده كنيد انتخاب كنيد.
- متد Copy در فهرستي كه در قسمت Target Directory مشخص مي شود اين فايل را خواهد ساخت. براي استفاده از متد FTP بايد نام سرويس دهنده راه دور و مقدار مجوزهاي لازم را وارد كنيد.
- به طور اختياري با تنظيم MIDlet-Jar-URL مي توانيد نام فايل JAR را تغيير دهيد.

ايجاد برنامه Client/Server

شما مي توانيد يك Client MIDlet ايجاد كنيد كه به يك سرويس وب متصل شود. اين اتصال به كمك يك برنامه ميان افزار (middleware) وب مي باشد. اين ويزارد يك ارتباط two-tier با استفاده از استاندارد JSR-172 كه مخصوص شناسايي سرويس هاي وب مي باشد ايجاد خواهد كرد بديهي است كه دستگاه موبايل بايد از JSR-172 پشتیباني كند. اگر دستگاه موبايل شما از JSR-172 پشتیباني نمي كنيد برنامه را به صورت three-tier ايجاد كنيد. در روش three-tier ويزارد به طور خود كار يك Servlet شامل برنامه client براي اتصال به سرويس وب ايجاد خواهد كرد و قبل از ايجاد با ويزارد بايد يك Web Project شامل Web Service Client را ايجاد کرده باشید.

براي استفاده از ويزارد گامهاي زير را دنبال كنيد.

- بر روي بسته برنامه موبايل كليك کرده، همانند شكل زير و از زير منوهاي گزينه New گزينه J2ME Web Service Client را انتخاب كنيد.



- بعد از اجرای ویزارد در قسمت WSDL Service محل فایل شرح دهنده سرویس وب را مشخص می کنیم. برای این کار دو روش وجود دارد.
 - ۱. اگر سرویس وب هم اکنون در اینترنت در حال اجرا باشد URL مخصوص فایل WSDL را وارد کرده و محل ذخیره شدن فایل دریافت شده را مشخص کنید. اگر از دیوار آتش مخصوص اینترنت در شبکه شما استفاده می شود بر روی دکمه Proxy Setting کلیک کرده و نام میزبان و پورت را وارد کنید. برای دریافت فایل بر روی دکمه Retrieve WSDL کلیک کنید.
 - ۲. اگر فایل WSDL هم اکنون در سیستم شما موجود است محل آنرا برای IDE مشخص کنید.
 - زمانی که فایل WSDL گرفته شد باقیمانده فیلدها بطور خودکار پر می شوند
 - بر روی دکمه Finish کلیک کنید.
- حال IDE به طور خود کار client stub به همراه تنظیمات را ایجاد می کند.
- استفاده از ویزارد Mobile Client to Web Application
- بر روی بسته برنامه موبایل کلیک کرده، همانند شکل زیر و از زیر منوهای گزینه New گزینه Mobile Client to Web Application را انتخاب کنید.
 - در صفحه انتخاب Web Application and Client type يك پروژه، Servlet یا يك بسته (Package) برای ایجاد کدهای سرور انتخاب کنید
 - در صفحه انتخاب سرویس، سرویس صادر شونده از سرور به برنامه سرویس گیرنده را مشخص کنید.
 - در صفحه Client Option نام و بسته برای کلاس ایجاد شونده برای سرویس گیرنده را انتخاب کنید
 - در همان صفحه می توانید انواع کدهای ایجاد شونده را تعیین کنید.

مراجع :

۱ مجله علم الكترونيك و كامپيوتر <http://www.iranPcWordMaz.com>

۲ برنامه نويسي شي گرا با جاوا در ۲۱ روز- محمد باقر معموري - انتشارت نص

۳ EJB – ريچارد مانسون هافل – انتشارات OReilly

۴ <http://developers.sun.com/>

۵ Java Servlet Programing – جيسون هانتر- انتشارات OReilly

۶ J2EE and XML Development - كورت گابريك – انتشارات MANNING

۷ <http://java.sun.com/docs/books/tutorial>

۸
NetBeans IDE Field Guide, Developing J2ME Mobility
Applications