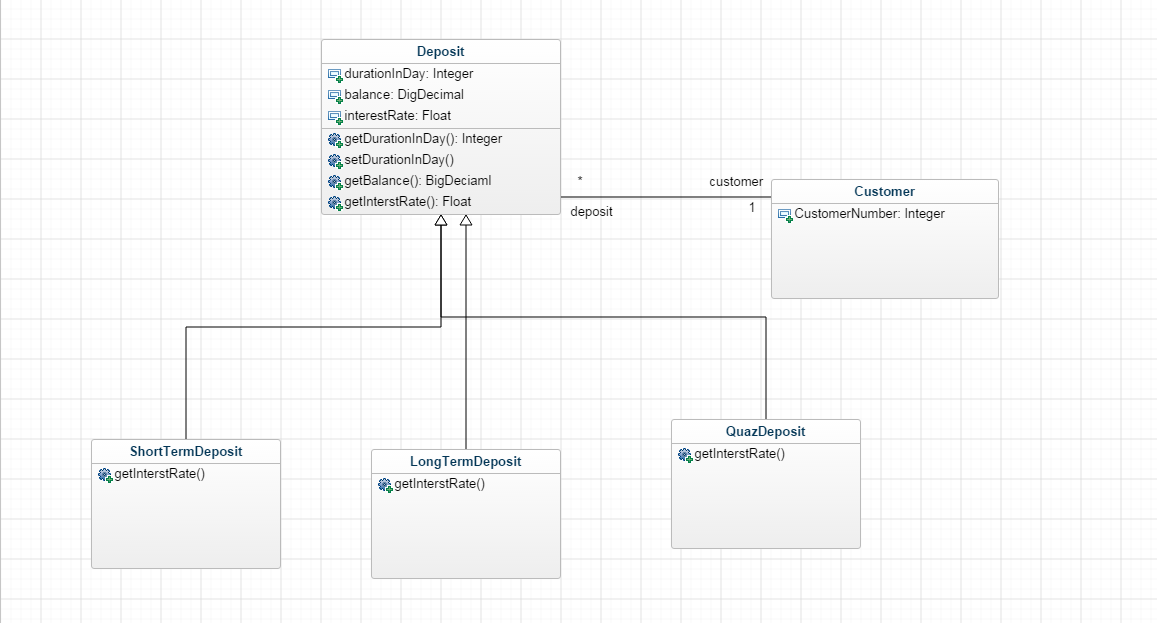
گزارش پیشرفت رضا صادقی خاص

پسورد سیستم : rezareza

گیت : <https://github.com/rezasparrow/DotinSchool>

روز اول : 16/01/1395

کلاس دیاگرام :



**بررسی Java Naming Convention :**

قرارداد نام گذاری در جاوا مجموعه قواعدی هستند که برای نام گذاری شناسه ها مثل نام متغیر ها ، کلاس‌ها ، توابع و غیر. که این امر باعث می‌شود که خوانایی کد برای خود یا دیگران راحت تر شود.

قواعدی که باید آنها را رعایت کنیم به صورت زیر هستند:

* نام کلاس‌ها : باید با حرف بزرگ شروع شود و یک اسم باشد مثل String , Color , Button و غیره.
* نام اینترفیس‌ها : باید با حرف بزرگ شروع شود و یک صفت باشد مثل Runnable , ActionListener و غیره .
* نام متد‌ها : باید با حرف کوچک شروع شود و یک فعل باشد مانند print() , main() و غیره .
* نام متغیرها: باید با حروف کوچک شروع شود مانند : firstName , customerNumber و غیره .
* نام پکیج : باید با حرف کوچک باشد مانند : java , lang , sql و غیره .
* نام ثابت‌ها : باید همه با حرف بزرگ باشند مانند : RED , YELLOW , MAX\_PRIORITY .

مشکلات :

* برای پارس کردن فایل xml با چندین روش روبرو بودم و باید یکی از این روش ها را استفاده می‌کردم در روش DOM کل فایل را یکباره به داخل حافظه میاورد ولی در عوض سریع تر می‌توان کد زد و سریع تر کارها را پیش برد روش دیگری که با آن روبرو بدم روش SAX بود که این روش بر خلاف روش قبلی یک درخت ایجاد نمی‌کند و بر اساس رویدادها عمل می‌کند و اینکه برای فایل های بزرگ خوب است ولی معایبی مثل نداشتن دسترسی رندم به فایل داریم و همواره باید روبه جلو حرکت کنیم و در اینجا چون فکر کردم که فایل می‌تواند بزرگ باشد و اینکه نیازی پیمایش رندم نداریم بنابراین سعی از StAX که مانند SAX است ولی استفاده از آن آسان تر است استفاده شده است.
* مشکل در اینکه ایا خوب است کلاس Deposit به صورت abstract باشه یا نه ؟ که چون در این کلاس یکسری ویژگی‌ها و متد ها یکسان است پس این کلاس را به صورت انتزاعی در نظر گرفتم و سپس موارد یکسان آن را در خودش پیاده سازی کردم و کلاس‌های دیگر که از این کلاس ارث میبرند مجبور به پیاده سازی مواردی هستند که در این کلاس بیان شده است که باید پیاده سازی شود.

گزارش روز دوم: 18/01/1395

**بررسی ها:**

۱. Reflection

به ما این قابلیت را می‌دهد که بتوانیم کلاس‌ها ، اینترفیس‌ها ، متدها و ... را بدون دانستن نام آنها در زمان کامپایل آنها را در زمان اجرا بررسی کنیم .

**مشکلات** :

۱. مشکل در اجرای متد payedInterest با استفاده از reflection :

مشکلی که وجود داشت این بود که در من برای استفاده از کردن از تابعی که در کلاس پدر وجود داشت سعی می‌کردم که با استفاده از reflection از خود آن کلاس ان متد را صدا بزنم بنابراین خطای اینکه این متد در این کلاس وجود ندارد برمی‌خوردم در حالی که باید این متد را از کلاس پدرش صدا میزدم که با انجام این کار مشکل در این قسمت حل شد.

۲. به علت استفاده نکردن از قابلیت نمایش تعداد اعداد اعشار در عدد BigDecimal در تقسیم خوردن به مشکل خوردم.

۳. مشکل در هندل کردن اکسپشن هایی که در حالت reflection وجود دارند که باید انها را با استفاده از اکسپش InvocationTargetException هندل کرد.

**روز سوم :**

**بررسی ها :**

۱ . Polymorphism in Java:

چند ریختی در زبان جاوا به دو صورت زمان اجرا و زمان کامپایل است که چند ریختی در زمان کامپایل وقتی اتفاق می‌افتد که یک تابع استاتیک را دوباره‌نویسی کنیم اما در حالت اجرا یک تابع دوباره‌نویسی شده بسته به زمان اجرا، فراخوانی می‌شود.

**Upcasting:**

وقتی که reference variable کلاس پدر به یک شی از کلاس بچه اشاره کند.

روز پنجم :

Socket Programming:

قدم های تولید یک ارتباط TCP :

1. ایجاد یک شی ServerSocket و مشخص کردن اینکه ارتباط بر روی کدام پورت است.
2. سرور تابع accept از کلاس ServerSocket را فراخوانی می‌کند که این متد منتظر می‌ماند تا یک کلاینت بر روی این پورت وصل شود.
3. بعد از اینکه سرور منتظر ماند یک کلاینت یک شی Socket ایجاد می‌کند و نام سرور و پورت ان را برای اتصال به آن مشخص می کند.
4. سازنده socket سعی در ایجاد ارتباط می‌کند و اگر این ارتباط موفقیت آمیز بود که از این پس یک شی socket داریم که می‌توانیم با آن ارتباط برقرار کنیم.
5. در سمت سرور متد accept یک شی از socket برمی‌گرداند که به کلاینت متصل است.

روز ششم:

استفاده از maven :

در موادی که از یکسری مخزن یکسری کتابخانه را می خواهیم نصب کنیم یک راه سنتی این است که خودمان به صورت دستی آن ها رار دانلود کنیم و سپس از آنها در پروژه استفاده کنیم و یک راه دیگر این است که از ابزاری‌هایی استفاده کنیم که ‌آنها خودشان این کار را برای ما انجام می‌دهند و این کتاب‌خانه‌ها را دانلود می‌کنند و در اختیار ما قرار میدهند و می‌توانیم از انها استفاده کنیم که ابزار می‌ون یکی از کارهایی که برای ما انجام می‌دهد این کار است.

روز هفتم :

بررسی مشکلات در استفاده از مالتی ترد بودن برنامه‌:

یکی از مشکلاتی که در این مورد به ان برخورم مشکل دراستفاده از قفل ها برای اجرای برنامه بود که این قفل‌ها در یک شی چگونه عمل می‌کنند و اینکه اگر ما یک قفل را در داخل یک شی قفل کنیم ایا در شی ‌های دیگر هم به این بخش دسترسی دارد یا دستری ندارند که پس از بررسی به این نتیجه رسیدم که برای هماهنگ کردن یک بخش باید یک شی به متد synchronized بدهیم که عملیات هماهنگ سازی بر روی آن انجام می‌شود حال اگر ما ورودی هر کدام ازاین بخش ها را خود ان شی بدهیم بنابراین باعث می‌شود که یک بخش در یک شی با شی‌های دیگر مجزا باشد و مشکل دیگری که وجود داشت بحث دستری به دو متد مختلف به صورت همزمان است که دو متد در یک شی به صورت همزمان اجرا نشوند که این مشکل هم با همان دادن خود شی به عنوان پارامتر هماهنگ کننده حل می‌شود.

مشکل دیگری که فکر کردم شاید اتفاق بیفتد مشکل محدود بودن منابع سیستم است که باید برای این منظور تعداد کانکشن‌هایی که با سیستم برقرار می‌کنیم را محدود کنیم یعنی باید تعداد thread هایی که در حال جواب دادن به مشتری ها هستند را محدود کنیم که این کار را با استفاده thread pool انجام دادم.

1395/01/26:

Try-with-resource:

وقتی که ما بخواهیم تضمین کنیم که یک منبع حتما بسته می شود به این روش ان را انجام می‌دهیم که همه‌ی انهایی که می خوهیم بسته شوند را در یک try قرار می‌دهیم و باید توجه داشته باشیم که کلاسی در این وضعیت تعریف می‌کنید باید حتما اینترفیس AutoCloseable یا Closeable را تعریف کرده باشند. با استفاده از این روش خیلی نیازی به try – catch های تودرتو نیست.

**معماری چند لایه :**

در سیستم‌های مهندسی بخصوص سیستم‌های مهندسی نرم‌افزار برای غلبه بر پیچیدیگی که سیستم را به چند زیر‌سیستم تقسیم می‌کنند.در طول طراحی هر زیر سیستم، ارتباط آن را با سایر زیر سیستم‌ها در نظر گرفته می‌شود و در نهایت این مجموعه را به عنوان یک مجموعه واحد در نظر می‌گیریم.

به طور کلی در معماری‌های چند لایه، هر لایه سرویس‌هایی را از لایه‌ی قبلی خود دریافت می‌کند و سرویس‌هایی را به لایه‌ی بعدی خود ارائه می‌کند.

**معماری سه لایه :**

این معماری حالت خاصی از معماری چند لایه است که بر اساس ان سیستم به سه لایه تقسیم می‌شود. ایده‌ی بوجود آمدن معماری سه لایه بعد از به وجود آمدن صفحات وب پویا بوده است این سه لایه عبارتند از :

**۱ . لایه نمایش (Presentation Layer ):**

این لایه که به آن لایه اینترفیس نیز گفته می‌شود شامل تمام عناصری است که توسط کاربر قابل رویت که مربوط به رابط گرافیکی کاربر می‌باشد . مواردی مانند فرم‌ها ، تصاویر و منوها و در کل هر انچه که به کاربر نهایی مرتبط است در این بخش قرار میگیرد. کاربر نهایی فقط با این بخش در ارتباط است و با بخش های دیگر ارتباطی ندارد. وظیفه این لایه نمایش اطلاعات به کاربر و همچین گرفتن اطلاعات لازم از کاربر استو در صورت لزوم باید یکسری اعتبار سنجی بر روی اطلاعات انجام دهد مانند بررسی طول فیلد‌ها و اجباری بودن و ... .

در این بخش نباید اثری از منطق اصلی برنامه یا اتصال به دیتابیس وجود داشته باشد.

**۲ . لایه منطق تجاری ( Business Logic Layer):**

این لایه حاوی منطق اصلی برنامه است.‌ همه‌ی درخواست های کاربر به این لایه منتقل می‌شود و سپس پردازش لازم را بر روی آنها انجام می‌دهیم و نتیجه‌ی این پردازش به لایه‌ی نمایش منتقل می‌شود و نتایج به کاربر نمایش داده می‌شود. در برخی موارد درخواست های کاربر بگونه‌ای است که باید با لایه‌ی داده‌ها یعنی لایه‌ی زیرین خود در ارتباط باشدکه در این حالت این به لایه‌ی داده درخواست خود را ارسال می‌کند و سپس لایه‌ی داده درخواست را انجام داده و داده‌ها را دریافت کرده و سپس اطلاعات را به لایه‌ی بالایی خود یعنی منطق ارسال می‌کند و سپس لایه منطق این داده‌ها را بر اساس درخواست کاربر به او نشان میدهد.

وضیفه‌ی اصلی لایه‌ی منطق برنامه اعمال منطق اصلی برنامه بر روی درخواست کاربران و همچنین برقرار ارتباط با لایه‌ی داده و نمایش می باشد.

**۳ . لایه دسترسی به داده‌ (Data Access Layer):**

وظیفه‌ی آن مدیریت اطلاعات موجود در بانک های اطلاعاتی را بر عهده دارد و همچنین بر اساس درخواست‌هایی که از لایه‌ی بالایی دریافت می‌کند باید عملیاتی از قبیل: حذف، اضافه، اصلاح، خواندن اطلاعات و ... را بر روی بانک‌های اطلاعاتی انجام میدهد و سپس اطلاعات را به لایه‌ی بالایی خود انتقال میدهد باید توجه داشته باشیم که دسترسی به بانک‌های اطلاعاتی فقط از این لایه صورت می‌گیرد.

**مزایا و معایب :**

از انجا که ما این لایه‌ها نسبت به هم وابسته نیستند و هر لایه‌ مستقل از سایرین عمل می‌کند بنابرین در صورتی که نیاز داشته باشیم که این لایه‌ها را تغییر می‌توانیم با سرعت و با هزینه پایین انها را تغییر دهیم.

اما معماری سه همواره راه‌حل مناسبی نیست برای طراحی هر سیستمی به دلیل مقرون به صرفه نبودن نمی‌توان از این معماری استفاده کرد. مثلا برای سیستم‌های کوچک استفاده از معماری سه لایه زمان‌بر و هزینه بر است و در واقع هزینه نگهداری و پیاده‌سازی را افزایش میدهد و همچنین به دلیل سلسله مراتبط استفاده از لایه ها زمان در این عملکرد پایینه.

JDBC:

یک رابط برنامه‌نویسی جاوا است برای وصل شدن به دیتابیس و اجرا کردن کوئری‌ها بر روی آن.

قبل از JDBC ، ODBC API رابط برنامه نویسی برای وصل شدن و اجرا کردن کوئری در دیتابس بود که به دلیل اینکه ODBC API از درایور‌های ODBC استفاده می‌کرد که آنها در زبان سی نوشته شده بودند (یعنی اینکه وابسته به پلتفرم و ناامن بودند)استفاده می‌کرد.به این دلیل جاوا رابط برنامه‌نویسی خودش را نوشت که از درایور JDBC استفاده می‌کند که در زبان جاوا پیاده سازی شده است.

JDBC Driver:

یک جز نرم‌افزاری است که به برنامه‌های جاوا این قابلیت را می‌دهد که بتوانند با دیتابیس ارتباط برقرار کنند. که چهار نوع است :

**۱. JDBC-ODBC bridge driver:**

این درایور از دایور ODBC برای وصل شدن به دیتابیس استفاده می‌کند. که در این جا درایور JDBC-ODBC متدهای JDBC را به متدهای ODBC تبدیل می‌کند و سپس توابع آن را اجرا می‌کند.

**مزایا** : ۱ . اسانی در استفاده ۲. به راحتی می‌توان به هر دیتابیسی متصل شد.

**معایب** : ۱. عملکرد پایین می‌اید به این دلیل که ما باید متدها را به همدیگر تبدیل کنیم. ۲ . درایور ODBC باید بر روی ماشین کلاینت هم نصب شده باشد.



**۲. Native-API driver**

در این روش از کتاب‌‌خانه‌های سمت کلایت استفاده می‌کندو درایور متدهای JDBC را به فراخوانی های native دیتابیس تبدیل می‌کند.به صورت کامل در زبان جاوا نوشته نشده است.



**مزایا**:

۱ . عملکرد نسبت به حالت قبل بیشتر شده است.

**معایب :**

**۱.**باید درایورهای محلی باید بر روی هر کلاینت نصب شده باشد

**۲.**کتاب‌خانه‌ها باید بر روی هر کدام از کلاینت‌ها باشند.

**۳ . Network Protocol driver**

از یک میدل‌ور (برنامه سرور ) که فراخونی‌های JDBC را به صورت مستقیم یا غیر مستقیم به پروتوکل سمت کلاینت تبدیل می‌کند که به صورت کامل در جاوا نوشته شده است.



**مزایا:**

نیاز به کتاب‌خانه‌ی سمت سرور نیست زیرا برنامه سمت سرور کارهای زیادی مانند گرفتن لاگ، متعادل کردن بار و ... را انجام می‌دهد.

**معایب:**

نیاز به پشتیبانی شبکه در سمت ماشین کلاینت

نیاز به یک برنامه خاص در لایه میدل‌ور

استفاده از پروتکل های شبکه هزینه‌بر است زیرا نیاز به برنامه نویسی مخصوص دیتابیس در لایه میانی است

**۴. Thin driver**

این درایور مستقیما فراخوانی ها رابه پروتکل‌های مخصوص دیتابیس تبدیل می‌کند.



**مزایا :**

عملکرد بهتر نسبت به بقیه

نیاز به برنامه ای در سمت سرور یا کلاینت نیست

**معایب:**

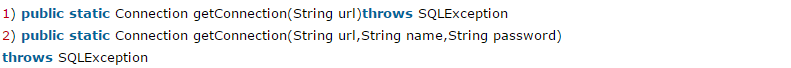
درایور‌ها به نوع دیتابیس‌ها وابسته هستند

پنج قدم برای وصل شدن به دیتابیس:

* **Register the driver class**



* **Creating connection**





* **Creating statement**





* **Executing queries**





* **Closing connection**



**مشکل حذف کردن مشتری‌ و شماره مشتری:**

اگر که از جدول مشتری داده‌ها را حذف کنیم چون که هر کس که به ان اشاره داشته باشد هم حذف می‌شود پس این مشکل برای حالتی که ما داده‌های این جدول را حذف کنیم پس داده‌های جدول های مشتری حقیقی و مشتری حقوقی هم حذف می‌شود ولی بر عکس آن اتفاق نمی‌افتد و در صورتی که داده‌ای از هر کدام از این جداول حذف شود سطر متناظر با آن در جدول مشتری حذف نمیشد که برای حل این کار از trigger استفاده کردم که در صورت حذف شدن یکی داده‌های جدول بالایی آن نیز حذف شود.

**مشکل پارامترهای اولیه در routing :**

برای اینکه در روت‌های اولیه با یکسری پارامتر خاص به توابع ما ارجاع داده شود این پارامترها را در فایل web.xml در بخش servlet بخش init-parameter را مقدار دهی می‌کنیم و سپس برای گرفتن مقادیر آنها باید در توابع خود تابع زیر را فراخوانی کنیم که بتوانیم داده‌ها را از آن بخوانیم.



الگوری طراحی Singleton :

از این الگو زمانی استفاده می‌کنیم که می‌خواهیم از یک کلاس در کل برنامه فقط یکی داشته باشیم . یعنی کلاس باید اطمینان حاصل نماید که از آن فقط یکی ساخته شده و همه‌ی کلاس‌ها از ان استفاده می کنند. این الگوی طراحی به دو شکل زیر وجود دارد :

**Early Instantiation**: که کلاس را در زمان بارگذاری میسازد.

**Lazy Instantiation:** ساختن کلاس در زمان گرفتن instance از آن صورت میگیرد.

مزایای استفاده از این الگو این است که به دلیل اینکه در هر درخواست یکی از ان ساخته نمی‌شود پس در استفاده از حافظه صرف‌جویی می‌کنیم.

این الگو بیشتر در برنامه‌های مالتی‌ترد و برنامه‌هایی که دیتابیس دارند استفاده می‌شود.

چگونگی استفاده از این الگو :

برای استفاده از این الگو ما باید در کلاس خود یک عضو استاتیک از کلاس و یک constructor خصوصی و همچنین یک تابع استاتیک برای ساختن داشته باشیم.

**عضو استاتیک** : به این علت که استاتیک است فقط یکبار حافظه به آن تخصیص داده می‌شود و شامل یک نمونه از خود کلاس است.

**سازنده خصوصی**: از این جلوگیری می کند که کلاس را از بیرون بتوانند بسازند.

**Static factory method:** این متد اجازه‌ی دستری به این کلاس را می‌دهد.

**Early Instantiation:**

در این حالت ما یک نمونه از کلاس را در زمان مشخص کردن و بیان کردن[[1]](#footnote-1) عضو استاتیک می‌سازیم بنابراین نمونه کلاس در زمان لود شدن کلاس بوجود می‌آید.

**Lazy Instantiation:**

در این حالت ما نمونه کلاس را در یک متد همزمان یا یک بلاک همزان می‌سازیم بنابراین در این حالت کلاس زمانی که نیاز باشد ساخته می‌شود.

**نوشتن اطلاعات فارسی در دیتابیس :**

باید توجه داشت که در زمان ساخت دیتابیس نوع collection را بر روی utf8-presian-ci قرار دهیم که بتوانیم اطلاعات فارسی را در دیتابیس‌ها ذخیره کنیم علاوه بر این کار باید هنگامی که از طریق jdbc به این دیتابیس می خواهیم وصل شویم به صورت زیر عمل کنیم و نوع کد داده‌های ارسالی را مشخص کنیم

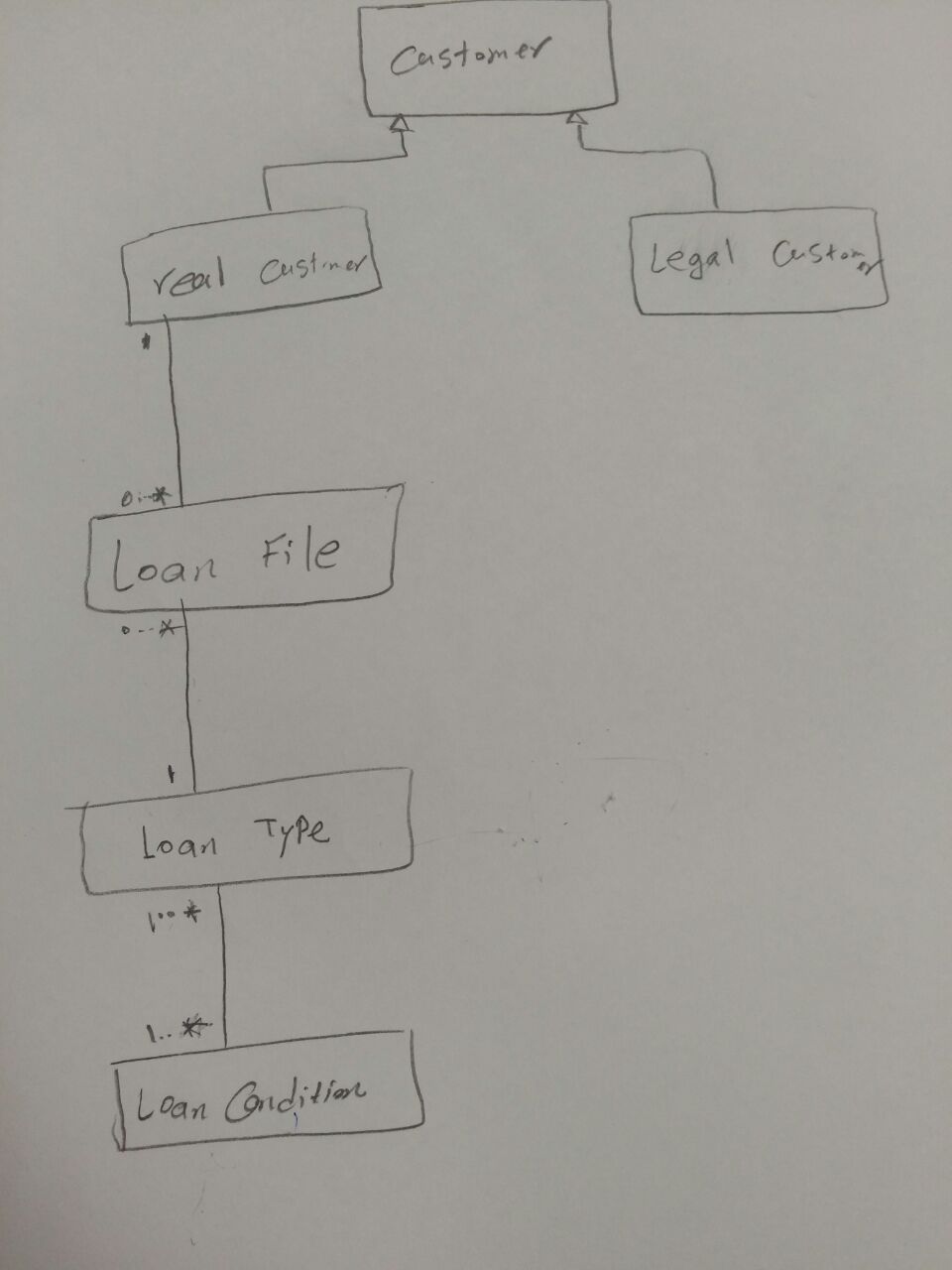


اما اگر از مورد بالا استفاده کنیم باز هم همچنان مشکل ذخیره اطلاعات در پایگاه‌داده وجود دارد و باید از قسمت بالا بخش useUnicode=true را حذف کرد.

کار دیگری که باید کرد تغییر نوع کد درخواست هاست که برای این کار باید در متد های dofilter(..) نوع کد ها را به utf8 مقدار دهی کرد که برای این کار از کد زیر استفاده می‌کنیم.



**کلاس دیاگرام پروژه شماره چهار:**



**‌JSP:**

Java Server Page (JSP) یک تکنولوژی برنامه نویسی سمت سروس که قابلیت ایجاد برنامه‌های وب پویا و مستقل از پلتفرم را به ما می دهد به این صورت که به برنامه‌نویس این اجازه را می‌دهد که کد جاوا در کد HTML به وسیله‌ی تگ‌های مخصوص JSP بنویسد که اکثر این تگ‌ها با <% شروع می‌شوند و با تگ %> تمام می‌شود. JSP به خانواده‌ی Java API مانند JSBD API برای دسترسی به دیتابیس را دارد.

**مزایای استفاده از JSP‌:**

**در مقابل ASP**:

بخش داینامیک به زبان جاوا نوشته شده است نه به زبان و ویژوال‌بیسیک بنابراین برای استفاده آسانتر و قویتر است. و همچنین قابلیت انتقال است و وابسته به سیستم‌عامل نیست.

**در مقابل سرولت خالص** **:**

راحت تر است که خومان کد‌های HTML را بنویسیم نه اینکه این کد‌ها را با استفاده از println تولید کنیم.

**در مقابل جاوا اسکریپت**:

جاوا اسکریپت بیشتر برای تولید HTML سمت کاربر است و به سختی می‌تواند با وب‌سرور ارتباط برقرار کند و کارهای پیچیده مانند دسترسی به دیتابیس را انجام دهد.

**در مقابل HTML استاتیک :**

واضح است که در این حالت نمی‌توانیم اطلاعات پویا را نمایش دهیم.

**پروسه ایجاد یک صفحه بر اساس JSP‌:**

۱. مانند یک صفحه عادی، مرورگر یک درخواست HTTP برای وب‌سرور ارسال می‌کند.

۲. وب‌سرور شناسایی می‌کند که یک این درخواست برای یک صفحه JSP است و سپس آن را برای موتور JSP ارسال می‌کند. که این با استفاده از اینکه در انتهای ‌آدرس .jsp وجود دارد به جای .html انجام می‌شود.

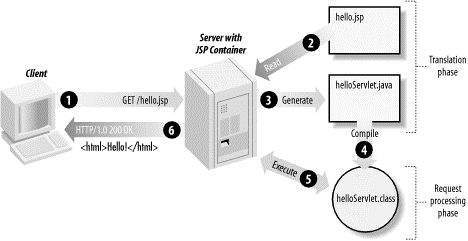
۳. موتور JSP صفحات جی‌اس‌پی را از دیسک بارگذاری می کند و آنها را به محتویات سرولت تبدیل می‌کند. این تبدیل بسیار ساده است به این صورت که همه‌ی متن‌های عادی به println() تبدیل می‌شوند و تگ‌های JSP به کد جاوا تبدیل می‌شود که پاسخ دهنده به رفتار داینامیک هستند.

۴. موتور JSP سرولت را به کلاس قابل اجرا کامپایل میکند و درخواست اصلی را به موتور سرولت ارسال می‌کند.

۵. بخشی از وب‌سرور موتور سرولت را فراخوانی می‌کند که کلاس را بارگذاری و اجرا کند. در طول اجرا سرولت یک خروجی در فرمت HTML تولید می‌کند که موتور سرولت آن را به عنوان پاسخ به وب‌سرور در قالب یک پاسخ HTTP ارسال می کند.

6. وب سرور پاسخ را به عنوان یک HTML استاتیک ارسال می کند.

۷. در نهایت مرورگر پاسخ صفحه ای که به صورت داینامیک تولید شده بود را همانطور که یک صفحه‌ی استاتیک را بررسی می‌کند را پردازش می‌کند.



موتور جی‌اس‌پی بررسی می کند که ایا یک فایل سرولت برای فایل مورد نظر وجود دارد و اینکه زمان تغییر فایل جی‌اس‌پی قدیمی‌تر از فایل سرولت است یا نه. اگر تاریخ آن قدیمی‌تر بود بنابراین فرض می‌شود که فایل جی‌اس‌پی تغییر نکرده است و سرولت تولید شده هنوز همه‌ی محتویات فایل جی‌اس‌پی را در خود دارد. که این باعث می‌شود که بهینه‌تر نسبت به زبان‌های اسکریپتی ( مانند PHP) و بنابراین سریع‌تر باشد. بنابراین صفحات JSP نوعی دیگر از نوشتن سرولت است.

**چرخه‌ی حیات JSP:**

چرخه‌ی حیات یک جی‌اس‌پی شامل چهار مرحله می‌شود:

۱. Compilation ۲. Initialization  ۳. Execution ۴. Cleanup



1. **Compilation:**

وقتی که یک مرورگر درخواست یک JSP می‌کند . موتور JSP در ابتدا بررسی می‌کند که آیا این فایل نیاز به کامپایل دارد یا نه. اگر که فایل هرگز کامپایل نشده بود یا تغییراتی در آن نسبت به آخری اگر نیاز به آخری کامپایل اعمال شده بود بنابراین موتور JSP صفحه را کامپایل می‌کند.

فرایند کامپایل شامل سه مرحله‌ی زیر می شود:

**۱. پارس کردن فایل JSP**

**۲. تبدیل JSP به سرولت**

**3. کامپایل کردن سرولت**

1. **Initialization:**

وفتی که JSP بارگذاری شد قبل ازا اینکه هر نوع درخواستی بررسی شود متد jspInit() فراخوانی می‌شود.

1. **Execution:**

در این بخش از چرخه‌ی حیات JSP همه‌ی درخواست‌ها تا زمان خراب خراب شدن فایل JSP بررسی می‌شود. وقتی که یک مرورگر یک JSP درخواست می‌کند و صفحه لود می‌شود و مقداردهی می‌شود سپس متد \_jspService() فراخوانی می‌شود. این متد با هر درخواست یک بار فراخوانی می‌شود و مسئول پاسخ گویی به همه‌ی درخواست ها و همهس اونواع روش‌های درخواست HTTP است.

**۴. Clean:**

مرحله تمیز کردن JSP بیان‌گر زمانی از چرخه‌ی حیات است که JSP در حال حذف شدن است. که jspDestroy() با حذف کردن در سرولت است.

برای نوشتن کد جاوا در JSP به صورت زیر عمل می‌کنیم :



برای معرفی کردن به صورت زیر عمل میکنیم:



JSP Expression:



در این حالت پس از اجرای این خط از کد مقدار آن به رشته تبدیل می‌شود و در آنجا قرار داده می‌شود. در اینجا می‌توان از هر عبارتی که در جاوا مجاز است استفاده کنیم با این شرط که نباید آخر آن سمیکالون داشته باشد.

JSP Comment:



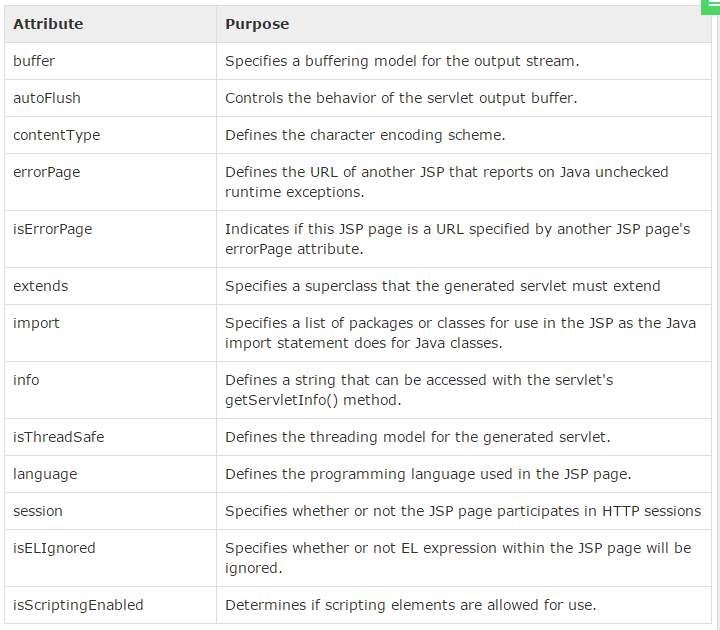
برای کامنت گذاشت در JSP به صورت بالا عمل می‌کنیم.

**Directive:**

جهت و ساختار چگونگی بررسی کردن یک جنبه خاص JSP با بیان می‌کند و بر روی ساختار کلی کلاس‌های سرولت تاثیر می‌گذارد که معمولا به شکل زیر است:



ویژگی‌هایی که برای یک دایرکتیو صفحه می‌توان استفاده کرد:

 دایرکتیو صفحه سعی در ارائه ساختاری است که مربوط به صفحه‌ی JSP کنونی است.

**دایرکتیو Include** : این دایرکتیو برای این است که بتوان یک فایل را در مرحله ترجمه به کد اضافه کنیم این دایرکتیو به کانتینر می‌گوید که یک فایل دیگر را در مرحله پردازش با این فایل JSP در زمان ترجمه کردن ادغام کند. حالت کلی این دایرکتیو به صورت زیر است‌:



taglib Directive:

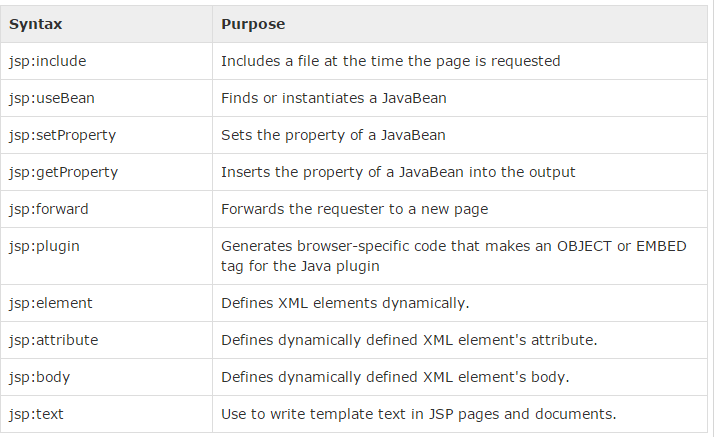
JSP API این اجازه را می‌دهد که بتوانیم تگ‌های سفارشی سازی شده مانند HTML و XML را تعریف کنیم و یا یکسری کتابخانه برای رفتار‌های سفارشی سازی شده پیاده‌سازی کنیم. که دایرکتیو taglib مشخص می‌کند که صفحه‌ی JSP ما از مجموعه‌ای از تگ‌های سفارشی استفاده می‌کند.

# Actions:

از ساختاری در xml برای کنترل کردن رفتار موتور سرولت استفاده می‌کند. ما می‌توانیم به صورت داینامیک یک فایل اضافه کنیم ، استفاده مجدد از کامپوننت‌های ‌javaBean ، انتقال کاربر به صفحه‌ی دیگر یا تولید html برای پلاگین جاوا.

تنها یک نوع نحوه‌ی نوشتن برای عنصر action وجود دارد که آن هم مطابق با xml استاندارد است و به صورت زیر است :





ویژگی‌های id و scope برای همه‌ی عناصر action وجود دارند :

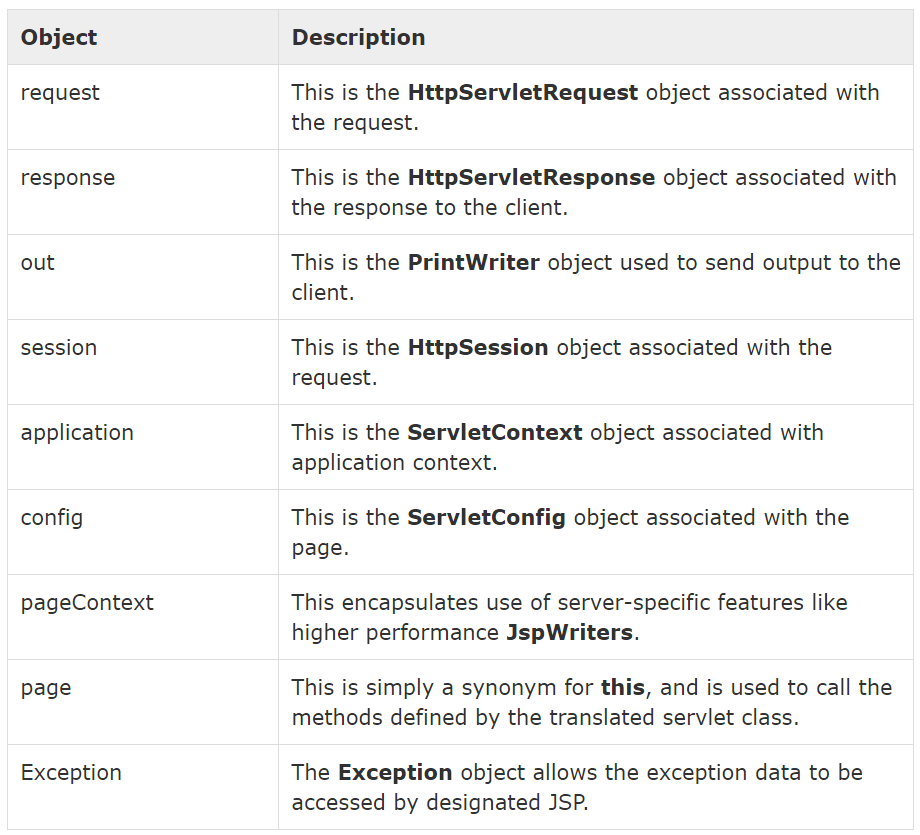
**ویژگی id‌:** این ويژگی موجب شناسایی منحصر به فرد یک اکشن می‌شود و این قابلیت را به ما می‌دهد که بتوانیم از داخل jsp به آن ارجاع بدهیم.

**ویژگی scope:**

این ویژگی بیان کننده‌ی چرخه‌ی حیات یک عنصر اکشن است. ویژگی‌های id و scope به صورت مستقیم به همدیگر مرتبط هستند. دامنه یک ویژگی یکی از چهار موارد روبروست : ۱.صفحه ۲. درخواست ۳. جلسه (session) ۴.برنامه

# Implicit Objects:

یک سری شی از قبل تعریف شده در هر شامل شونده‌ی JSP وجود دارد که می تواند به آنها درسترسی داشت که به صورت زیر هستند:



# فیلتر‌ها:

فیلتر ها در jsp و سرولوت کلاس‌های جاوا هستند که در سرولت و جی‌اس‌پی به منظور‌های زیر استفاده می‌شود:

۱. جلوگیری از دستری یک کاربر به منابع قبل از ورود به بک‌اند.

۲. دستکاری پاسخ سرور قبل ازاینکه به سمت کلاینت ارسال شود.

فیلتر‌ها در فایل web.xml به سرولت‌ها یا جی‌اس‌پس‌ها یا آدرس ها نگاشت می‌شود.

JSP - Standard Tag Library (JSTL) Tutorial

JSTL یک مجموعه‌ای کاربردی از تگ‌های جی‌اس‌پی است که قابلیت‌های مشترک در خیلی از برنامه‌های جی‌اس‌پی را دار خود دارد. تگ‌های جی‌اس‌پی بر اساس نوع کاربردی که دارند می‌تواند به دسته های زیر تقسیم شوند‌:

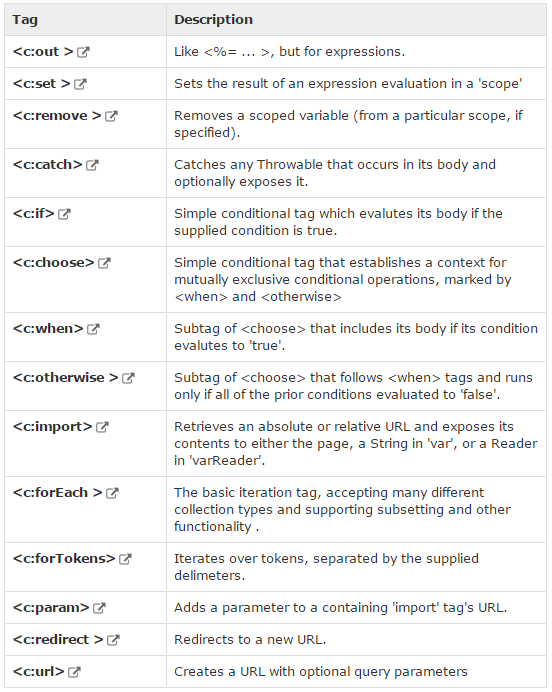
* **Core Tags**
* **Formatting tags**
* **SQL tags**
* **XML tags**
* **JSTL Functions**

**Core Tags:**

تک‌های پر کاربرد در این دسته قرار می‌گیرند. نوع بیان‌کردن تگ‌ها به صورت زیر است:



و تگ‌های موجود در این کتابخانه به صورت زیر هستند:



# Hibernate :

فریمورک هایبرنیت ارتباط با دیتابیس را در برنامه‌های جاوا آسان می‌کند. هایبرنیت یک ابزار متن‌باز و ORM است.

یک ORM ایجاد داده‌‌ها، نگه داری داده‌ها و دسترسی به آنها را ساده‌سازی می‌کند. یک تکنیک برنامه‌نویسی است که شی‌ها را به داده‌های قابل ذخیره در دیتابیس نگاشت می‌کند.



ابزارهای ORM به صورت داخلی از JDBC API برای ارتباط با دیتابیس استفاده می کنند.

## **مزایای استفاده از فریمورک هایبرنیت :**

**۱. متن‌باز بودن و سبک**

**۲. عملکرد سریع :** به دلیل استفاده از کش سطح اول در فریمورک هایبرنیت عملکرد آن سریع استو دو نوع کش در هایبرنیت وجود داد : کش سطح اول و کش سطح دوم. کش سطح اول به صروت پیشفرض فعال است.

**۳. استقلال دیتابیس از کوئری :** HQL زبان جستوجو هایبرنیت یک ورژن شی‌گرای از sql است. که کوئری‌ها را مستقل از دیتابیس تولید می‌کند. بنابراین نیاز نداریم که کوئری‌ها را برای هر دیتابیس به صورت خاص بسازیم. قبل از هایبرنیت اگر دیتابیس ما تغییر کند بنابراین نیاز داریم که کوئری‌های خود را تغییر بدهیم که این یک مشکل اساسی در نگه داری است.

**۴. ایجاد خودکار جداول:** هایبرنیت به ما این اجازه را می‌دهد که بتوانیم بجای ساخت جدول‌ها به صورت دستی آنها را به صورت اتوماتیک ایجاد کنیم.

**۵. آسان‌سازی اتصالات پیچیده‌ :** برای گرفتن داده‌ها از چندین جدول این کار در فریمورک هایبرنیت آسان است.

**6. ارائه آمار کوئری‌ها و وضعیت پایگاه‌داده‌ها:** هایبرنیت از کش در جستوجوها پشتیبانی استفاده می‌کند و آمار‌هایی در مورد وضعیت دیتابیس و جستوجو‌ها به ارائه می‌دهد.

## **معماری هایبرنیت :**

معماری هایبرنیت شامل شی‌های زیادی از persistent object, session factory, transaction factory, connection factory, session, transaction و غیره است.

چهار لایه در معماری هایبرنیت وجود دارد:

۱. برنامه جاوا

۲. لایه فریمورک هایبرنیت

۳. Backend API

۴. دیتابیش

## **عناصر معماری هایبرنیت :**

**Session:**

یک اینترفیس بین برنامه و دیتابیس ارائه میدهد.یک شی کوتاه مدت است و ارتباط JDBC را احاطه می‌کند. یک کارخانه[[2]](#footnote-2) از تراکنش[[3]](#footnote-3) ، جست‌وجو و ضوابط[[4]](#footnote-4) است و اینکه کش سطح اول را به صورت الزامی در خود دارد. رابط org.hibernate.Session متدهای insert , update و delete را دارد و همجنین متد‌هایی را برای تراکنش، جستوجو و ضوابط نیز دارد.

**Transaction**:

یک شی تراکنش بیان‌گر این است بخش اتمیک کار است.که اختیار است. Org.hibernate.Transaction رابط برای مدیریت تراکنش را ارائه می‌دهد.

**ConnectionProvider**:

یک کارخانه از ارتباطات JDBC است و اختیاری است .

**TransactionFactory**:

یک کارخانه از تراکنش‌ها است و اختیاری است.

برای ساخت یک برنامه هایبرنیت ما باید مراحل زیر را انجام بدهیم :

1. Create the Persistent class
2. Create the mapping file for Persistent class
3. Create the Configuration file
4. Create the class that retrieves or stores the persistent object

**Create the Persistent class:**

یک کلاس ماندگار باید قوانین زیر را داشته باشد :

۱. **یک سازنده بدون آرگومان**: توصیه می‌شود که کلاس ما یک سازنده بدون آرگومان داشته باشد که توسط پکیج رویت باشد که هایبرنیت بتواند با متد newInstance() به آن دسترسی داشته باشد.

**۲. ارائه‌ی یک ویژگی شناسه (اختیاری):** که به کلید اصلی در دیتابیس نگاشت می‌شود.

**۳. تعریف کردن getter و setter (اختیاری):** هایبرنیت به صورت پیش فرض **getter و setter** ها را از طریق نامشان شناسایی می‌کند.

**4.کلاس غیر‌نهایی[[5]](#footnote-5) باشد:** هایبرنیت از مفهوم پروکسی استفاده می‌کند که به کلاس پایدار[[6]](#footnote-6) وابسته است.

**Create the mapping file for Persistent class:**

نام فایل به صورت قراردادی باید به صورت class\_name.hbm.xml باشد.

عناصر موجود در فایل نگاشت :

۱. **hibernate-mapping: ‌ عنصر روت در فایل نگاشت است.**

**۲. class: زیر‌عنصر hibernate-mapping است و مشخص کننده کلاس persistent است.**

**۳. id: زیر عنصر کلاس است و مشخص کننده کلید اصلی است.**

**۳. Generator: یک ریز عنصر id است که برای تولید کلید اصلی استفاده می‌شود. روش‌های زیادی برای تولید کلید اصلی وجود دارد مانند تخصیص ( در حالتی که id توسط کاربر وارد می‌شود)، افزایشی ، ترتیبی و غیره.**

**۴ .property :** یک زیر عنصر کلاس است که مشخص کننده نام اعضای کلاس پایدار است.

**Create the Configuration file:**

فایل پیکربندی شامل اطلاعاتی در مورد دیتابیس و فایل نگاشت است به صورت قرار دادی نام آن باید hibernate.cfg.xml باشد.

**Create the class that retrieves or stores the object:**

**در این کلاس ما داده‌ها را در دیتابیس ذخیره می‌کنیم.**



## **ارث‌بری در هایبرنیت:**

برای پیاده‌سازی ارث‌بری در دیتابس می‌تونیم به این صورت عمل کنیم که یا همه را در یک جدول ذخیره کنیم و سپس با استفاده از یک جدا کننده مشخص کنیم که این سطر برای کدام نوع است یا اینکه برای هر کدام یک جدول جداگانه تولید کنیم و روش سوم این است که برای هر کلاس یک جدول داشته باشیم با این تفاوت که در این کلاس‌ همه‌ی متغیر‌ها را در خود داشته باشد.

در هایبرنیت هم همه‌ی این موارد پیاده‌سازی شده است که یکی از روش‌های استفاده از آنها استفاده از علامت‌گذاری[[7]](#footnote-7) است. که این علامت‌گذاری‌ها به صورت زیر هستند:

* Table Per Hierarchy: @Inheritance (strategy = InheritanceType.***SINGLE\_TABLE***)
* Table Per Concrete: @Inheritance(strategy = InheritanceType.***TABLE\_PER\_CLASS***)
* Table Per Subclass: @Inheritance(strategy = InheritanceType.***JOINED***)

یکی از مواردی که در حالت join باید به آن توجه کرد این است که نباید در هر جدول یک کلید اصلی داشته باشیم.

## @OneToMany vs @ManyToOne

دو نوع علامتی‌گذاری برای رابطه One-to-Many در هایبرنیت وجود دارد:

@OneToMany

@ManyToOne

دلیل اینکه دو نوع علامت‌گذاری وجود دارد این است که ما می خوایم به هایبرنیت بگوییم که کدام شی بخش بچه یا بخش many در رابطه است و کدام بخش پدر یا بخش one در رابطه است. که ما با استفاده از @OneToMany می‌گوییم که کدام شی پدر است و همچنین با استفاده از @ManyToOne ما به هایبرنیت می‌گوییم که کدام شی بچه است. اما اگر فقط بخش @ManyToOne را پیاده‌سازی کنیم نیز کار می‌کند.

1. declaring [↑](#footnote-ref-1)
2. factory [↑](#footnote-ref-2)
3. transaction [↑](#footnote-ref-3)
4. criteria [↑](#footnote-ref-4)
5. Non-final [↑](#footnote-ref-5)
6. persistent [↑](#footnote-ref-6)
7. annotation [↑](#footnote-ref-7)