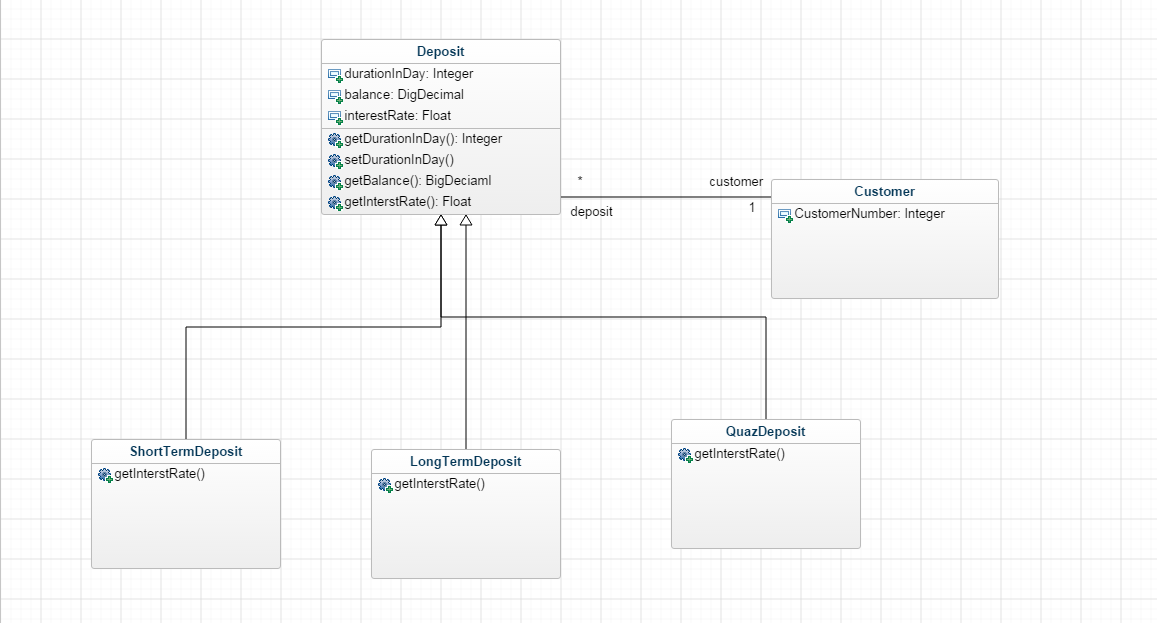
گزارش پیشرفت رضا صادقی خاص

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:**

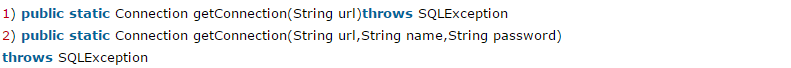
درایور‌ها به نوع دیتابیس‌ها وابسته هستند

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.

1. declaring [↑](#footnote-ref-1)