xبه نام خدا

دستور کار کارگاه برنامه‌نویسی پیشرفته

نیم‌سال دوم ۹۷–۹۸

جلسه نهم

آشنایی با abstract class و interface در جاوا

مقدمه

در جلسه قبلی با مفهوم ارث‌بری و کاربرد آن در جاوا آشنا شدید. در طراحی کلاس‌ها می‌توان متدها و متغیرهای مشترک چند کلاس را در قالب یک کلاس پدر نوشت که بقیه کلاس‌ها از این کلاس پدر ارث بری می‌کنند. این‌کار باعث کاهش حجم بسیاری از کد شده و کار توسعه[۱] و نگهداری[۲] را برای توسعه‌گران آسان می‌کند.

در جلسه گذشته نیز با مفهوم نمودار کلاسی و بعضی از اجزای آن آشنا شدید. این نمودار به شما کمک می‌کند تا بتوانید کلاس‌های خود را با جزییات بسیار زیادی و نزدیک به زبان برنامه‌نویسی توصیف کنید و آنرا به دیگران انتقال دهید.

در این جلسه قصد داریم تا مثال داده شده در جلسه قبل را با مقداری تغییر پیاده‌سازی کنیم. قبل از اینکه به پیاده‌سازی آن بپردازیم قصد داریم تا با استفاده از مفاهیم **abstract class** و **interface** طراحی خود را بهبود بخشیم.

abstract Class

abstract class یا کلاس انتزاعی به آن دسته از کلاس‌ها گفته می‌شود که بصورت کامل پیاده‌سازی نشده‌اند و ممکن است پیاده‌سازی بعضی از متدهای آنها بر عهده فرزندانشان باشد. در طراحی زمانی از این نوع کلاس‌ها استفاده می‌کنیم که قرار است کلاس‌های فرزند یک کلاس پدر، همه متدهای خاصی را پیاده‌سازی کنند ولی پیاده‌سازی این متدها در هر کلاس فرزند متفاوت است. به روایتی بوسیله‌ی abstract method ها می‌توان کلاس‌های فرزند یک کلاس را مجبور کرد تا یک سری رفتارها را پیاده‌سازی کند (یا اینکه خود یک کلاس abstract شوند). پس می‌توان برای شکل دادن به رفتار کلاس‌ها از abstract class ها استفاده کرد. در مثال جلسه قبلی برای کلاس Employee که کلاس

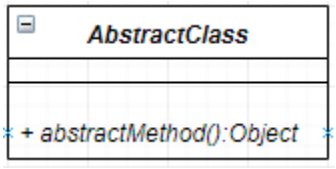
[۱] Development

[۲] Maintainance

پدر برای دو فرزند ServiceEmployee و Professor بود دیدید که رفتار گزارش حقوق هر ترم آنها که در قالب متد calCurrentIncome طراحی شده است. از طرفی پیاده‌سازی این متد برای تمامی کارمندان این دانشگاه نیاز است و از طرف دیگر هر دسته از کارمندان این سازمان پیاده‌سازی خاص خود را دارند. لذا در این حالت باید کلاس Employee را به یک کلاس abstract تبدیل کرد و متد calCurrentIncome را برای این کلاس بصورت abstract method تعریف کرد. برای آنکه بقیه توسعه‌دهندگان متوجه شوند که کلاس Employee یک کلاس انتزاعی است بهتر است نام کلاس را به AbstractEmployee تغییر دهید. علاوه بر این در این سامانه می‌خواهیم برای کارمندان متدی بنویسیم که آیا این کارمندان می‌توانند درخواست ارتقای شغلی بدهند یا خیر. از آنجایی که تمام کارمندان چه کارمندان خدماتی و چه اساتید دانشگاه قابلیت ارتقا دارند پس این متد بین فرزندان کلاس مشترک است. می‌توان اسم این تابع را isPromoteable گذاشت که خروجی آن یک boolean است به معنای آنکه آیا این کارمند توانایی ارتقا دارد یا خیر. منتها تعریف ارتقا برای کارمندان متفاوت است بدین صورت که برای کارمندان خدماتی باید حداقل ۳ سال از تاریخ آخرین ارتقای آنها گذشته باشد و برای اساتید بدین گونه است که باید از زمان آخرین ارتقای خود تا کنون ۱۰ مقاله ارایه داده باشند تا اجازه‌ی ارتقا داشته باشند.

انجام دهید

نمودار کلاسی جلسه پیش را که در قالب یک فایل XML هست دانلود کنید و آنرا در نرم افزار draw.io باز کنید. سپس نمودار کلاسی را براساس تغییراتی که بالا بیان شد بروز کنید. برای نشان دادن انتزاعی بودن متدها و کلاس‌ها باید نام آنها را بصورت *italic* بنویسید. شکل ۱ نحوه‌ی نمایش کلاس و متد انتزاعی را در نمودار کلاسی نشان می‌دهد. سپس یک پروژه برای پیاده‌سازی پروژه ایجاد کنید. کلاس‌های Course و Department به عنوان مثال برای تبدیل اجزای نمودار کلاسی به کلاس‌های جاوا به شما داده شده. آنها را دانلود کنید و به پروژه خود اضافه کنید. کلاس AbstractEmployee را به همراه دو فرزند آن پیاده‌سازی کنید. بقیه کلاس‌ها را نیز برا اساس نمودار داده شده پیاده کنید.



شکل ۱ نمایش کلاس و متد انتزاعی در UML

نکته: کلاس‌های خود را در یک پکیج به نام org.university.core قرار دهید.

سامانه مدیریت مالی

در مثال دانشگاه فرض کنید می‌خواهیم یک سامانه حسابداری ایجاد کنیم که وظیفه امور مالی دانشگاه را برعهده داشته باشد. این سامانه حسابداری باید توانایی مدیریت حساب مالی کاربران، افزودن حساب مالی و ارایه لیست صورت حساب‌های پرداختی را داشته باشد. کلاس حساب مالی دارای یک صاحب حساب است که از نوع کارمند است و یک مقدار عددی که موجودی حساب را بیان می‌کند و در ابتدا صفر است. حساب مالی همچنین عملکردی به نام checkout دارد. هنگامی که این متد فراخوانی شود باید مبلغی که کارمند از دانشگاه طلب دارد را گرفته و آنرا به موجودی حساب اضافه کند. هر کارمند که به سامانه اضافه می‌شود در این سامانه یک حساب مالی برای او ایجاد می‌شود پس سامانه حسابداری باید توانایی ایجاد حساب جدید را داشته باشد. این سامانه همچنین یک عملکردی به نام settle دارد که وظیفه‌ی تسویه حساب تمامی حساب‌های مالی را دارد. این عملکرد آخر هر ترم اجرا می‌شود و میزان طلب هر کارمند به مقدار موجودی حساب او واریز می‌شود و موجودی حساب افزایش می‌یابد. کارمندان از این مبلغ موجودی می‌توانند برای استفاده از خدمات دانشگاه استفاده کنند (مثلا از سلف یا سالن تربیت بدنی دانشگاه). همچنین این سامانه یک عملکردی به نام دریافت لیست صورت‌حساب‌ها دارد که لیستی از Statement ها را بازمی‌گرداند. هربار که عملکرد checkout یک حساب فراخوانی شد، یک Statement باید به لیست تراکنش‌های سامانه مالی اضافه شود.

در کلاس کارمندان متد calCurrentIncome وظیفه‌ی بازگرداندن مقدار طلب کارمندان از دانشگاه در ترم جاری است. لذا متد checkout در کلاس حساب‌مالی، متد calCurrentIncome صاحب حساب را (که یک کارمند است) صدا می‌زند. سپس خروجی آنرا به حساب کارمند واریز می‌کند.

انجام دهید

کلاس سیستم مالی را با نام AccountingManagment و کلاس حساب مالی را با نام Account ساخته و متدهای مربوط به هر کلاس را پیاده‌سازی کنید.

اگر بخواهیم برای بقیه کلاس‌ها نیز کاری کنیم که بتوانند با این سامانه مالی ما در ارتباط باشند باید چه کار بکنیم؟ به عنوان مثال علاوه بر کارمندان که یک دستمز دارند، دانشجویان ارشد نیز بخاطر آنکه دستیار آموزشی هستند هم پولی دریافت کنند و حساب مالی داشته باشند. از این رو باید برای کلاس GraduateStudent نیز تابع calCurrentIncome را پیاده‌سازی کرد که بر اساس تعداد مقالاتش یک مبلغی را حساب کرده و آنرا باز می‌گرداند. در این سامانه هر کلاس دیگری که نیاز به ارتباط با سامانه مالی را داشته باشد باید متد calCurrentIncome را

پیاده‌سازی کنند تا سامانه حسابداری بتواند مقدار طلب آنها را بدست آورد. درصورتی که بخواهیم با استفاده از method overloading به ازای هر کلاسی میخواهیم با سامانه حسابداری در ارتباط باشد یک متد بنویسیم (که ورودی آن متد از نوع کلاسی باشد که می‌خواهیم آنرا به سامانه مالی اضافه کنیم)، کدهای بسیار کثیف و غیر قابل نگهداری و توسعه خواهیم داشت. اگر بخواهیم از abstract class برای پیاده‌سازی قابلیت ارتباط با سامانه حسابداری استفاده کنیم باید کلاس Person را abstract کنیم (تا هم کلاس AbstractEmployee و هم کلاس GraduateStudent رفتار calCurrentIncome را پیاده کنند) و متد calCurrentIncome را بصورت abstract برای این کلاس تعریف کنیم. در این صورت ورودی تابع checkout و ایجاد حساب یک شی از کلاس فرزندان Person خواهد بود و نیازی به overload کردن توابع نیست (ورودی تابع checkout یک شی از کلاس Person خواهد بود و فقط یکبار نیاز به تعریف کردن دارد). سپس ورودی تابع checkout و تابعی که حساب کاربری درست می‌کند به جای کارمند باید یک Person شود. مشکل این روش پیاده‌سازی در این است که اگر کلاس Person یک کلاس abstract شود تمام فرزندان این کلاس باید این متد را پیاده کنند. این درحالی است که شاید تمامی کلاس‌های این سامانه (مثلا دانشجویان کارشناسی) نیازی به پیاده‌سازی این متد نداشته باشند.

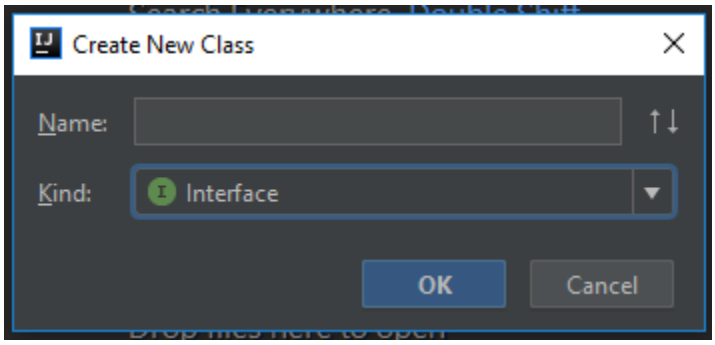
برای رفع این مشکل از interface استفاده می‌کنیم. به روایتی برای این سامانه باید یک interface به نام AccountingInterface ایجاد کنیم و در آن یک تابع انتزاعی به نام calCurrentIncome بسازیم که خروجی آن یک double است.

interface

interface به کلاسی گفته می‌شود که تمامی متدهای آن public و abstract هستند و هیچگونه پیاده‌سازی در این کلاس صورت نگرفته است. سوالی که شاید ذهن خیلی از برنامه‌نویسان (و شاید شما) را مشغول کند این است که چرا باید از interface استفاده کرد در حالی که هیچ گونه پیاده‌سازی در این کلاس صورت نگرفته. پاسخ این سوال این است که هدف از ساختن interface شکل دادن به رفتار کلاس‌هایی است که قرار است پیاده شود. سوال دیگری که مطرح می‌شود این است که چرا به جای interface از abstract class استفاده نمی‌شود؟ اگر یک abstract class هیچگونه متدی را پیاده‌سازی نکند بهتر است این کلاس در قالب interface باشد چرا که جاوا قابلیت ارث‌بری چندگانه یک کلاس از کلاس‌ها دیگر را ندارد ولی قابلیت پیاده‌سازی چند interface در یک کلاس را به برنامه‌نویسان می‌دهد.

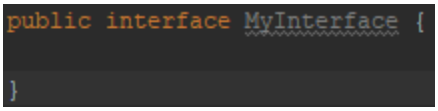
لذا interface **ها قابلیت این را دارند که بدون اعمال تغییر در ساختار درخت ارث‌بری رفتار کلاس‌ها را شکل‌دهی و استاندارد سازی کنند.**

نحوه‌ی ساخت interface **در** intellij **همانند کلاس است. ابتدا همانند ساخت کلاس جدید روی پوشه** src **راست کلیک کنید و گزینه کلاس جدید بسازید. در پنجره‌ای که در شکل زیر نمایش داده شده گزینه** interface **را بجای** class **انتخاب کنید.**



تصویر ۱ ساخت interface در intellij

سپس نام MyInterface **را وارد کنید. خواهید دید که همانند کلاس یک فایل با این نام برای شما ساخته شده که محتوای آن مانند شکل زیر است.**



تصویر ۲ نمایی از یک interface

حال باید توابع abstract **این** interface **را تعریف کنید. تابع** absMethod() **را به این** interface **بصورت زیر اضافه می‌کنیم.**

```
public interface MyInterface {
        public abstract void myMethod();
}
```

حال اگر بخواهیم از این interface در یک کلاس استفاده کنیم باید از واژه implements استفاده کنیم.

یک کلاس MyClass بسازید. قطعه کد زیر پیاده‌سازی MyInterface را نشان می‌دهد.

```
public class MyClass implements MyInterface {
}
```

حال همانند abstract class که باید متدهای انتزاعی را پیاده‌سازی می‌کردید، در اینجا نیز باید متد myMethod را نیز پیاده کنید.

انجام دهید

AccoutingInterface را تعریف کنید. حال می‌خواهیم از این interface استفاده کنیم. در سامانه مالی متغیرهای AbstractEmplyee را به AccountingInterface تغییر دهید. واسط AccountingInterface را به کلاس دانشجویان تحصیلات تکمیلی اضافه کنید. سپس متدهای abstract آنرا پیاده‌سازی کنید. این کار باعث می شود تا سامانه مالی قابلیت ارتباط با کلاس GradStudent را داشته باشد. در این حالت هم کارمندان دانشگاه و هم دانشجویان ارشد قابلیت ایجاد حساب مالی و دریافت دستمزد از دانشگاه را دارند.