به نام خدا

دستور کار کارگاه برنامه‌نویسی پیشرفته

نیم‌سال دوم ۹۷-۹۸

جلسه ششم

آشنایی با توسعه آزمون محور و JUnit پروژه

مقدمه

در جلسه قبل با مفهوم debugging آشنا شدید. همانطور که گفته شد فرآیند debugging از دو بخش تشکیل شده که عبارت است از پیدا کردن منشا خطا و رفع اشکال آن. یکی از مهمترین منشاهای خطا در محصولات نرم‌افزاری در زمان توسعه آن اتفاق می‌افتد. از آنجایی که درصد زیادی از برنامه‌ها توسط برنامه‌نویسان نوشته می‌شود همواره احتمال خطا در کدها وجود دارد. عواملی مانند خستگی، استرس، بی دقتی یا تسریع در انجام پروژه می‌تواند باعث شود که برنامه‌نویسان تمام حالت‌هایی که کد آنها باید به آن حالت‌ها پاسخ دهد را در نظر نگیرند و این کار باعث ایجاد خطا در نرم افزار می‌شود.

برای کاهش تعداد خطاها در زمان توسعه سیستم روش‌های متعددی وجود دارد. یکی از آنها توسعه‌ی آزمون محور است. در این روش هنگامی که برنامه‌نویس سعی دارد کلاس یا متدی را بنویسد، قبل از آنکه شروع به نوشتن کد بکند تعدادی تست کیس (ممکن است توسط خودش طراحی بشود یا دیگران آنها را طراحی کنند) به پروژه اضافه می‌کند که درستی کدهای خود را با آنها بیازماید.

تست کیس

تست کیس مستندی است برای آزمایش قسمتی از برنامه در محیطی خاص با ورودی‌های خاص است. خروجی تست کیس بصورت خودکار یا بصورت انسانی با خروجی[۲]های مطلوب مقایسه می‌شود. تست کیس‌ها می‌توانند انواع مختلفی داشته باشند. در این جلسه قصد داریم تا با یکی از روش‌های تست که Unit Testing نام دارد کدهای خود را آزمایش کنیم. تست کیس ها در Unit Testing قطعه کدهایی هستند که برای توسعه محصول استفاده نمی‌شوند و برای بررسی درستی قسمت‌هایی از کد برنامه کاربرد دارند. قسمتی از کد که توسط هر تست کیس مورد بررسی قرار می‌گیرد محدوده تست (test coverage) آن تست کیس نامیده می‌شود. در unit testing محدوده هر تست

[۱] Test Driven Development

[۲] Automated testing

کیس معمولا زیاد نمی‌باشد و معمولا برای تست کردن یک متد یا یک کلاس کاربرد دارد. با اجرا کردن این تست کیس‌ها، آن قسمت از برنامه که برنامه نویس قصد آزمایش آنرا دارد بررسی می‌شود و نتیجه‌ی آن بصورت خودکار با نتیجه‌ی متوقع مقایسه می‌شود. درصورتی که خروجی کد با خروجی متوقع متفاوت بود، تست کیس پاس نخواهد شد (به اصطلاح شکست[۳] خواهد خورد). در جاوا برای تست قطعه معمولا از چارچوب JUnit استفاده[۵] می‌شود.

JUnit

برای طراحی تست کیس‌ها چه در جاوا با JUnit و چه در زبان‌های دیگر، نیاز است تا از پیش متدها و کلاس‌های که باید مورد آزمایش قرار بگیرند روی کاغذ تعریف شده باشند. منظور از تعریف کردن تابع مشخص کردن ورودی و خروجی و نام تابع است.

توضیح پروژه

در این جلسه سعی داریم تا سامانه انتخاب واحد دانشگاه را شبیه‌سازی کنیم. این سامانه از پنج کلاس زیر تشکیل شده است:

کلاس دانشگاه:

```
public class University{}
```

کلاس دانشکده:

```
public class Department{}
```

کلاس درس:

```
public class Course{}
```

کلاس دانشجو:

```
public class Student{}
```

کلاس استاد:

```
public class Professor{}
```

[۳] Pass

[۴] Fail

[۵] Unit testing

این کلاس‌ها را به همراه متدهایشان دانلود کنید و در یک پروژه کپی کنید. حال میخواهیم پیاده‌سازی یکی از کلاس‌ها را شروع کنیم. به عنوان مثال کلاس Student را انتخاب می‌کنیم. در ابتدا باید یک کلاس بسازیم که متدهای این کلاس را با ورودی‌های مختلف تست کند. داخل پکیج test کلاس StudentTest را باز کنید. داخل این کلاس تعدادی متد مشاهده می‌کنید که هر کدام وظیفه آزمودن قسمتی از کد (معمولا یک متد خاص) را بر عهده دارد (test case های کلاس Student). در ابتدا ساختار annotation های کلاس را بررسی می‌کنیم. در annotation JUnitهای متعددی وجود دارد که هرکدام معنی خاصی دارند. جدول زیر معنای بعضی از این annotation ها را توضیح می‌دهد.

@Test	بیانگر این است که متدی با این annotation یک تست کیس است.
@BeforeEach[۶]	بیانگر این است که متدی با این annotation باید قبل از اجرای هر تست کیس اجرا شود.
@AfterEach[۷]	بیانگر این است که متدی با این annotation باید بعد از اجرای هر تست کیس اجرا شود.
@BeforeAll[۸]	بیانگر این است که متدی با این annotation باید قبل از اجرای همه تست کیس‌ها اجرا شود.
@AfterAll[۹]	بیانگر این است که متدی با این annotation باید بعد از اجرای همه تست کیس‌ها اجرا شود.
@Disabled[۱۰]	بیانگر این است که متدی با این annotation تست کیسی است که نیاز به اجرا ندارد.

به کمک این annotation ها میخواهیم چند سناریو را شبیه‌سازی کنیم. سناریو اول این است که قرار است یک شی از کلاس Student ساخته شود که نام، شماره دانشجویی، رشته و دپارتمان آن به عنوان

[۶] در صورتی که از Junit 4 استفاده می‌کنید از Before@ استفاده کنید.

[۷] در صورتی که از Junit 4 استفاده می‌کنید از After@ استفاده کنید.

[۸] در صورتی که از Junit 4 استفاده می‌کنید از BeforeClass@ استفاده کنید.

[۹] در صورتی که از Junit 4 استفاده می‌کنید از AfterAll@ استفاده کنید.

[۱۰] در صورتی که از Junit 4 استفاده می‌کنید از Ignore@ استفاده کنید.

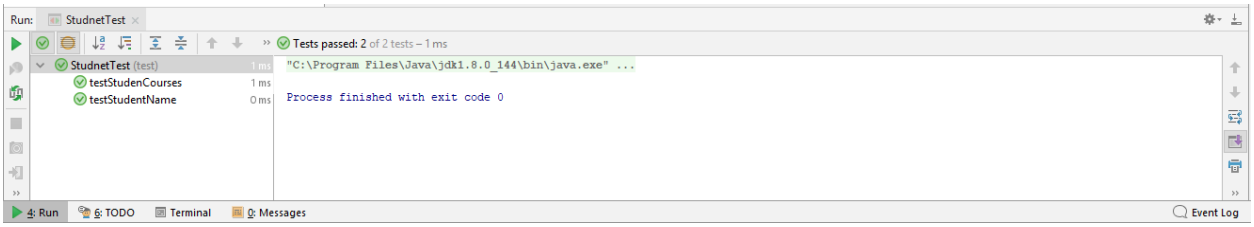
ورودی به سازنده کلاس داده می‌شود. سپس قرار است تابع getName آن را بررسی کنیم که باید در خروجی نام دانشجو (مثلا Alireza) را بازگرداند. در سناریو بعدی میخواهیم تابع getCourses را بررسی کنیم که لیست کلاس‌های دانشجو را بازمی‌گرداند.

از آنجا که در تمامی متد های کلاس StudentTest (تست کیس‌ها) قرار است با یک شی از کلاس Student کار کنیم، ابتدا آنرا یکبار مقداردهی می‌کنیم. معمولا متدهایی که @BeforeAll دارند برای فراهم کردن محیط تست ساخته می‌شوند. در کلاس StudentTest مشاهده می‌شود که یک متد @BeforeClass وجود دارد. همانطور که عنوان متد نیز بیان می‌کند (createStudent)، این متد وظیفه‌ی ساختن شی student را دارد.

دو متد دیگر مشاهده می‌کنید که دارای @Test هستند. متد اول به نام testStudentGetName سناریو اول را بررسی می‌کند. در بدنه این متد مشاهده می‌کنید که یک تابع assertEqual وجود دارد. این تابع بررسی می‌کند که آیا خروجی تابع getName شی student با مقدار متوقع "Alireza" برابر است یا خیر. تابع دوم به نام testStudentGetCourses سناریو دوم را بررسی می‌کند. همانطور که در بدنه متد مشاهده می‌کنید دو تابع assert تعریف شده است که اولین آن assertNotNull برای بررسی null نبودن خروجی متد getCourses است و assertEquals برای بررسی طول آرایه درس‌های شیstudent است که باید صفر باشد چون هنوز درسی برای دانشجو برنداشته نشده است.

برای اجرای کدهای تست کیس روی اسم کلاس راست کلیک کنید و عبارت Run ‘StudentTest’ را انتخاب کنید. در پایین صفحه نتیجه‌ی زیر را مشاهده خواهید کرد.

در قسمت چپ نام کلاس تست ما و متدهای آنرا مشاهده می‌کنید که علامت ! در کنار آنها وجود دارد. این به این معناست که اجرای تست کیس ها موفق آمیز نبوده و تابع‌های assert نتیجه‌های خطا را گزارش داده‌اند. علت این خطا این است که بدنه این توابع هنوز پیاده‌سازی نشده است. حال باید مرحله دوم که مربوط به توسعه متدهاست را اجرا کنیم. در این قسمت بدنه توابعی که در تست کیس‌ها مورد مقایسه قرار گرفته (با تابع assert) باید پیاده‌سازی شود. زمانی پیاده‌سازی توابع تمام می‌شود که تست کیس‌ها جواب درست بدهند. در صورتی که توابع را پیاده‌سازی کنید خروجی اجرای تست اینگونه می‌شود.



نکته: برای این تمرین یک پروژه در git بسازید و بعد از اتمام هر انجام دهید کارهای خود را commit و push کنید.

انجام دهید

۱. توابع getName و getCourses را پیاده کنید. تابع getName باید نام دانشجو که به عنوان ورودی در سازنده داده می‌شود را بازگرداند و تابع getCourses باید یک آرایه بازگرداند که شامل لیست تمام دروسی است که دانشجو آنها را برداشته است. بعد از پیاده سازی آنها تست کیس‌ها را دوباره اجرا کنید تا جواب تست کیس‌ها درست شود.

حال میخواهیم بقیه متدهای کلاس Student را با همین روش پیاده سازی و ارزیابی کنیم.

انجام دهید

۲. برای سناریوهای زیر تست‌های مناسب را به کلاس StudentTest اضافه کنید و توابع آنها را پیاده کنید. قبل از پیاده سازی متدهای زیر در کلاس Student، تست های مربوط به آنها را نوشته و اجرا کنید. واضح است که نتیجه این تست ها به علت عدم پیاده سازی متدها، نباید موفقیت آمیز باشد.
   a. تابع getID باید شناسه دانشجو را که به عنوان ورودی به تابع سازنده داده شده بود بازگرداند و نباید null باشد.
   b. تابع getDepartment باید دانشکده‌ی دانشجو را که در زمان ساخت شی student به آن داده شده بود را بازگرداند. خروجی تابع نباید null باشد.
   c. تابع getMajor باید رشته دانشجو را که در زمان ساخت شی به آن داده شده بود را بازگرداند. خروجی تابع نباید null باشد.
   d. تابع addCourse باید شی course را که در ورودی تابع به آن داده شده است را به لیست دروس دانشجو اضافه کند. در صورتی که ورودی تابع null بود نباید چیزی به لیست دروس اضافه شود.

بعد از آنکه توابع کلاس Student پیاده‌سازی شد باید کلاس دیگری را انتخاب کرد. کلاس دومی که قصد پیاده‌سازی آنرا داریم کلاس Course است. فرض کنید دانشگاهی که قرار است از این سامانه شما استفاده کند به شما می‌گوید که دانشجویی که در دانشکده A درس می‌خواند حق برداشت

درس در دانشکده B را ندارد. این قانون باید در سامانه شما پیاده شود و در زمان طراحی تست کیس‌ها باید به این نکته مهم توجه کرده و برای ارزیابی آن تست کیس های مناسب نوشته شود.

انجام دهید

۳. کلاس CourseTest را ساخته و سناریوهای زیر را برای کلاس Course پیاده کنید. بعد از نوشتن تست کیس‌ها و اجرای تست کیس‌ها، متدها را همانند انجام دهید قبلی پیاده‌سازی کنید.
   a. تابع getID باید شناسه درس را که به عنوان ورودی به تابع سازنده داده شده بود بازگرداند و نباید null باشد.
   b. تابع getDepartment باید دانشکده‌ی درس را که در زمان ساخت شی به آن داده شده بود را بازگرداند. خروجی تابع نباید null باشد.
   c. تابع getName باید نام درس را که به عنوان ورودی به تابع سازنده داده شده بود بازگرداند و نباید null باشد.
   d. تابع getProfessor باید استاد درس را که به عنوان ورودی به تابع سازنده داده شده بود بازگرداند و نباید null باشد.
   e. تابع enrollStudents باید دانشجوی ورودی را به لیست دانشجوهای عضو کلاس اضافه کند.
   f. تابع enrollStudent نباید دانشجویی که از دانشکده دیگر است را به لیست دانشجوهای کلاس اضافه کند (از assertNotEqual هم میتوانید استفاده کنید. یک حلقه for روی student های عضو course بزنید و بررسی کنید که آیا دانشجو داده شده در ورودی عضو درس شده یا خیر).

انجام دهید

٤. کلاس DepartmentTest را ساخته و سناریوهای زیر را برای کلاس تست آن بنویسد. سپس کلاس Department را پیاده‌سازی کنید.
   a. متد addStudent دانشجو ورودی را گرفته در صورتی که دانشکده دانشجو با دانشکده مطابقت نداشت آنرا به لیست دانشجوها اضافه نمی‌کند.
   b. متد addStudent دانشجو ورودی را گرفته در صورتی که دانشکده دانشجو با دانشکده مطابقت داشت آنرا به لیست دانشجوها اضافه می‌کند.
   c. متد getStudnets لیست دانشجوهای دانشکده را بازمی‌گرداند و نباید null باشد.
   d. تابع removeStudent دانشجو ورودی را گرفته در صورتی که دانشکده دانشجو با دانشکده مطابقت نداشت کاری انجام نمی‌دهد.

e. تابع removeStudent دانشجو ورودی را گرفته در صورتی که دانشکده دانشجو با دانشکده مطابقت داشت شروع به جست و جو در لیست دانشجو ها می‌کند. در صورتی که دانشجو داده شده در لیست پیدا شد آنرا حذف می‌کند.
f. تابع getName نیز نام دانشکده که در تابع سازنده به شی داده می‌شود را باز می‌گرداند.
g. متد addCourse درس ورودی را گرفته در صورتی که دانشکده درس با دانشکده مطابقت نداشت آنرا به لیست درس‌ها اضافه نمی‌کند.
h. متد addCourse درس ورودی را گرفته در صورتی که دانشکده درس با دانشکده مطابقت داشت آنرا به لیست درس‌ها اضافه می‌کند.
i. متد getCourses لیست درس‌های دانشکده را بازمی‌گرداند و نباید null باشد.
j. تابع removeCourse درس ورودی را گرفته در صورتی که دانشکده درس با دانشکده مطابقت نداشت کاری انجام نمی‌دهد.
k. تابع removeCourse درس ورودی را گرفته در صورتی که دانشکده درس با دانشکده مطابقت داشت شروع به جست و جو در لیست درس‌ها می‌کند. در صورتی که درس داده شده در لیست پیدا شد آنرا حذف می‌کند.
l. متد addProfessor استاد ورودی را گرفته در صورتی که دانشکده استاد با دانشکده مطابقت نداشت آنرا به لیست استادها اضافه نمی‌کند.
m. متد addProfessor استاد ورودی را گرفته در صورتی که دانشکده استاد با دانشکده مطابقت داشت آنرا به لیست استادها اضافه می‌کند.
n. متد getCourses لیست استادهای دانشکده را بازمی‌گرداند و نباید null باشد.
o. تابع removeCourse استاد ورودی را گرفته در صورتی که استاد داده شده در لیست اساتید دانشکده بود آنرا از لیست حذف می‌کند.

انجام دهید

۵. برای کلاس‌های University و Professor نیز مطابق تعریف‌های متدهای داده شده در کلاسشان تست کیس بنویسید و متدهای هر کلاس را تکمیل کنید.