

**گزارش تکلیف اول درس مباحث ویژه در نرم افزار 1**

نسترن عشوری – 810101225

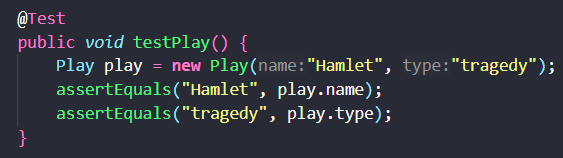
سوال اول )

الف –

* کلاس Play

این کلاس شامل نام نمایش و ژانر آن نمایش است.

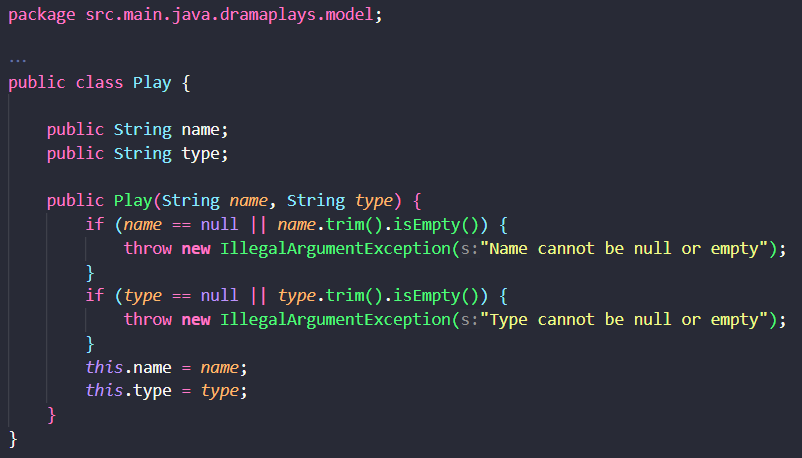
برای تست کردن، یک نمونه از کلاس با مقادیر اولیه ای می سازیم و سپس تست میکنیم آیا مقادیر متغیرهای آن کلاس با مقادیر داده شده برابر است یا نه.



میتوانیم تست کنیم که اگر رشته خالی یا null به این کلاس بدهیم چه اتفاقی رخ میدهد.



میبینیم که تست ها pass میشود. ولی این کار به نظر منطقی نیست. باید جلوی ست شدن متغیرهای کلاس با مقادیر null را گرفت. پس کلاس را به شکل زیر تغییر میدهیم.



حال مجموعه تست ها را به شکل زیر آپدیت میکنیم.

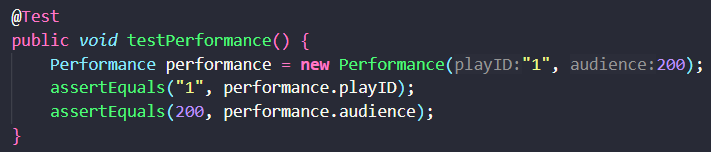




* کلاس Performance

این کلاس شامل آیدی نمایش و همچنین تعداد شرکت کنندگان آن اجرا است.

برای تست کردن، یک نمونه از کلاس با مقادیر اولیه ای می سازیم و سپس تست میکنیم آیا مقادیر متغیرهای آن کلاس با مقادیر داده شده برابر است یا نه.



میدانیم که آیدی نباید null باشد. همچنین تعداد شرکت کننده ها نباید منفی باشد. بعد از ساخت شی آیدی نباید تغییر کند اما تعداد شرکت کننده را میشود عوض کرد.

پس کلاس را به صورت زیر تغییر میدهیم.



حال برای هر یک از ویژگیهای ذکر شده تست مینویسیم.





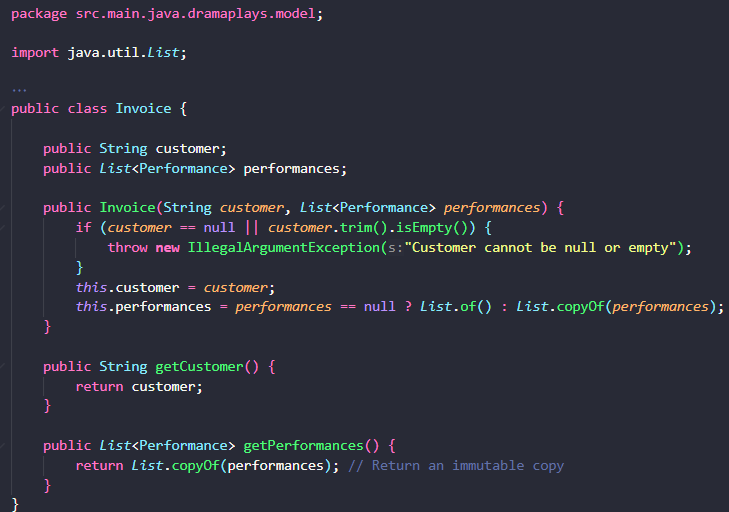
* کلاس Invoice

این کلاس شامل نام مشتری و همچنین لیستی از نمایش هاست. هنگام صدا شدن تابع سازنده این کلاس، این دو متغیر مقداردهی میشوند.

برای تست کردن، یک نمونه از کلاس با مقادیر اولیه ای می سازیم و سپس تست میکنیم آیا مقادیر متغیرهای آن کلاس با مقادیر داده شده برابر است یا نه.

همچنین باید تست شود customer برابر null نباشد. اما performances میتواند null باشد.

پس تابع را به صورت زیر تغییر میدهیم.



حال با مجموعه تست زیر کلاس را تست میکنیم.



با تغییرات ایجاد شده همه تست ها pass میشوند.

ب –

توضیح تابع:

کلاس `FactorPrinter` برای چاپ صورتحساب (فاکتور) مربوط به اجراهای تئاتر یک مشتری استفاده می‌شود. این کلاس شامل یک متد به نام `print` است که دو ورودی می‌گیرد: یک شیء از کلاس `Invoice` که اطلاعات مشتری و اجرای نمایش‌ها را نگه می‌دارد، و یک نقشه (`Map`) که اطلاعات نمایش‌ها را بر اساس شناسه آن‌ها ذخیره می‌کند. این متد محاسبات مربوط به هزینه هر نمایش و اعتباراتی که مشتری به دست آورده را انجام می‌دهد و سپس خروجی نهایی را به صورت یک رشته قالب‌بندی شده بازمی‌گرداند.

در این متد، بسته به نوع نمایش (تراژدی یا کمدی)، هزینه‌های مختلفی محاسبه می‌شود. برای نمایش‌های تراژدی، اگر تعداد تماشاگران بیش از ۳۰ نفر باشد، هزینه اضافی برای هر تماشاگر محاسبه می‌شود. در مورد نمایش‌های کمدی، علاوه بر هزینه اولیه، هزینه اضافی برای تعداد تماشاگران بیش از ۲۰ نفر و همچنین هزینه اضافی برای هر تماشاگر محاسبه می‌شود. همچنین، اعتباراتی بر اساس تعداد تماشاگران و نوع نمایش به دست می‌آید. در نهایت، متد `print` یک رشته شامل جزئیات فاکتور، مبلغ کل بدهی و اعتبارهای کسب شده را به صورت قالب‌بندی شده برمی‌گرداند.

سناریوهای زیر تست شده است.

1. یک اجرای تراژدی با کمتر از 30 تماشاگر
2. یک اجرای تراژدی با بیشتر از 30 تماشاگر
3. یک اجرای کمدی با کمتر از 20 تماشاگر
4. یک اجرای کمدی با بیش از 20 تماشاگر
5. یک فاکتور با چند اجرا
6. نمایش با تایپ unknown
7. نمایش تراژدی با صفر تماشاگر
8. نمایش کمدی با صفر تماشاگر
9. چند نمایش تراژدی
10. چند نمایش کمدی

مشاهده میکنیم که همه تست ها pass میشوند. در کد جزییات تست ها آورده شده است.

سوال دوم )

تست های زیر طراحی شده و همه pass شده اند. جزییات در کد قابل مشاهده است.

testHasPassedPre\_AllPrePassed:

در این تست، دو درس پیش‌نیاز ۱۰۱ و ۱۰۲ با نمرات ۱۰ و ۱۲ گذرانده شده‌اند.

تابع باید true بازگرداند زیرا تمام پیش‌نیازها پاس شده‌اند.

testHasPassedPre\_OnePreNotPassed:

در این تست، درس ۱۰۱ با نمره ۱۰ و درس ۱۰۲ با نمره ۹ گذرانده شده‌اند.

تابع باید false بازگرداند زیرا یکی از پیش‌نیازها پاس نشده است.

testHasPassedPre\_AllPrePassedWithMehman:

در این تست، دو درس پیش‌نیاز ۱۰۱ و ۱۰۲ با نمرات ۱۲ و ۱۳ گذرانده شده‌اند و هر دو دانشجوی مهمان (مه‌مان) هستند.

تابع باید true بازگرداند زیرا تمام پیش‌نیازها با نمرات بالای ۱۲ پاس شده‌اند.

testHasPassedPre\_OnePreNotPassedWithMehman:

در این تست، درس ۱۰۱ با نمره ۱۲ و درس ۱۰۲ با نمره ۱۱ گذرانده شده‌اند و هر دو دانشجوی مهمان هستند.

تابع باید false بازگرداند زیرا یکی از پیش‌نیازها با نمره زیر ۱۲ پاس نشده است.

testHasPassedPre\_NoPreRequired:

در این تست، درسی بدون هیچ پیش‌نیاز بررسی می‌شود.

تابع باید true بازگرداند زیرا هیچ پیش‌نیازی وجود ندارد که پاس نشده باشد.

testHasPassedPre\_NoRecords:

در این تست، هیچ رکوردی از نمرات وجود ندارد و درس پیش‌نیازهایی دارد.

تابع باید false بازگرداند زیرا هیچ پیش‌نیازی پاس نشده است.

testHasPassedPre\_EmptyPrerequisites:

در این تست، درس پیش‌نیاز ندارد.

تابع باید true بازگرداند زیرا هیچ پیش‌نیازی وجود ندارد.

testHasPassedPre\_AllPreFailed:

در این تست، هر دو درس پیش‌نیاز ۱۰۱ و ۱۰۲ با نمرات زیر ۱۰ (۸ و ۷) گذرانده شده‌اند.

تابع باید false بازگرداند زیرا هیچ یک از پیش‌نیازها پاس نشده‌اند.

testHasPassedPre\_MultipleTerms:

در این تست، درس‌های پیش‌نیاز ۱۰۱ و ۱۰۲ در ترم‌های مختلف با نمرات ۱۰ و ۱۲ گذرانده شده‌اند.

تابع باید true بازگرداند زیرا تمام پیش‌نیازها پاس شده‌اند.

testHasPassedPre\_SinglePrePassed:

در این تست، تنها یک درس پیش‌نیاز (۱۰۱) با نمره ۱۰ گذرانده شده است.

تابع باید true بازگرداند زیرا پیش‌نیاز پاس شده است.

testHasPassedPre\_SinglePreFailed:

در این تست، تنها یک درس پیش‌نیاز (۱۰۱) با نمره ۸ گذرانده شده است.

تابع باید false بازگرداند زیرا پیش‌نیاز پاس نشده است.

testHasPassedPre\_ExactBoundaryNonMehman:

در این تست، درس پیش‌نیاز (۱۰۱) با نمره دقیقاً ۱۰ گذرانده شده است و دانشجو مه‌مان نیست.

تابع باید true بازگرداند زیرا نمره دقیقاً برابر با حداقل نمره قبولی است.

testHasPassedPre\_ExactBoundaryMehman:

در این تست، درس پیش‌نیاز (۱۰۱) با نمره ۱۲ گذرانده شده است و دانشجو مه‌مان است.

تابع باید true بازگرداند زیرا نمره دقیقاً برابر با حداقل نمره قبولی برای دانشجوی مه‌مان است.

testHasPassedPre\_BelowBoundaryMehman:

در این تست، درس پیش‌نیاز (۱۰۱) با نمره ۱۱ گذرانده شده است و دانشجو مه‌مان است.

تابع باید false بازگرداند زیرا نمره کمتر از حداقل نمره قبولی برای دانشجوی مه‌مان است.

testHasPassedPre\_MultipleRecordsSameCourse:

در این تست، درس پیش‌نیاز (۱۰۱) در دو ترم مختلف با نمرات ۸ و ۱۰ گذرانده شده است.

تابع باید true بازگرداند زیرا نمره بالاتر (۱۰) پاس شده است و دانشجو درس را در نهایت پاس کرده است.

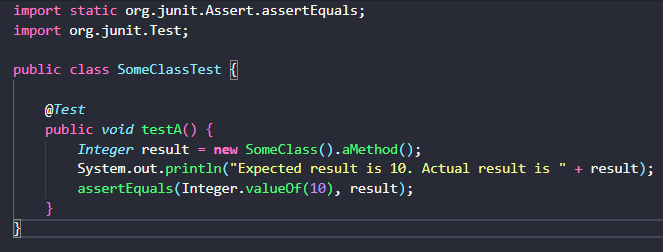
سوال سوم )

الف)

عدم وجود دستور import برای کلاس مورد نظر: اگر SomeClass و متد aMethod در پکیج های جدا باشند باید آن‌ها را import کنیم.

عدم وجود assertion برای بررسی نتیجه: در یک تست واحد، باید از assertions برای بررسی نتیجه استفاده شود تا مطمئن شویم که نتیجه مورد انتظار حاصل شده است.

کد اصلاح شده :



ب)

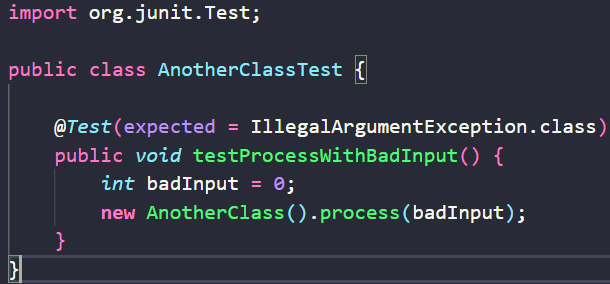
استفاده نادرست از annotation @Test:

برای اعلام اینکه یک تست انتظار وقوع یک استثنا را دارد، باید از ویژگی expected در annotation @Test استفاده شود. در کد فعلی، expects Exception یک ویژگی معتبر برای annotation @Test نیست.

نامشخص بودن نوع استثنا:

باید مشخص کنیم که کدام نوع استثنا انتظار می‌رود. به عنوان مثال، اگر انتظار داریم IllegalArgumentException رخ دهد، باید این نوع را مشخص کنیم.

کد اصلاح شده:



**ج)**

استفاده از متدهای initialize در تست‌ها:

فراخوانی متدهای initialize در هر تست به طور مجزا می‌تواند منجر به مشکلاتی شود زیرا وضعیت سیستم ممکن است ناپایدار یا ناهمگام شود.

عدم تضمین ترتیب اجرای تست‌ها:

ترتیب اجرای تست‌ها در JUnit تضمین نمی‌شود. بنابراین ممکن است testResourceAvailability قبل از testInitialization اجرا شود و در نتیجه منابع مورد نیاز آماده نباشند.

عدم بررسی نتیجه‌ی متد initialize:

هیچ بررسی‌ای برای اطمینان از موفقیت‌آمیز بودن اجرای متد initialize وجود ندارد.

برای رفع این مشکلات می‌توانیم از annotation های @Before یا @BeforeClass برای اطمینان از مقداردهی اولیه‌ی منابع مورد نیاز قبل از اجرای تست‌ها استفاده کنیم. همچنین بهتر است بررسی کنیم که متدهای initialize به درستی اجرا شده‌اند.

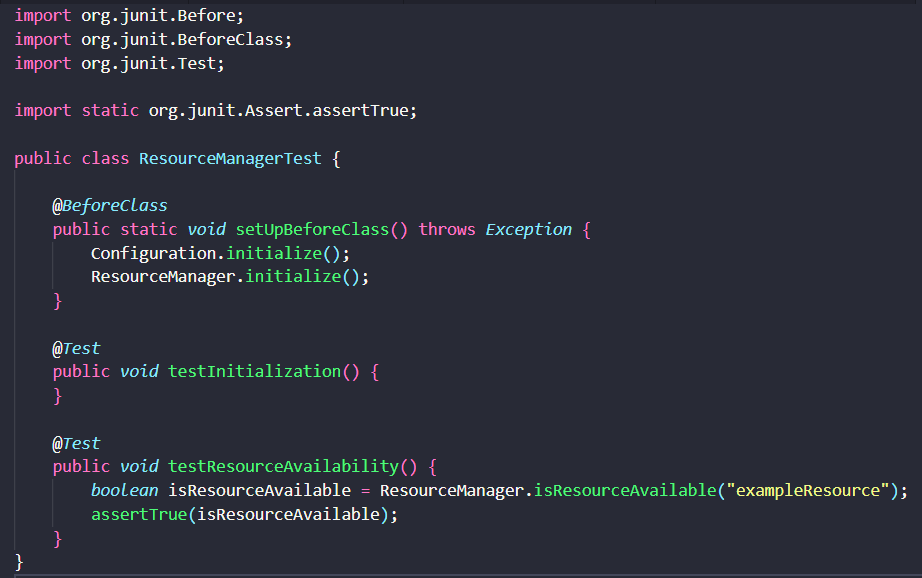
توضیحات:

@BeforeClass برای اطمینان از اجرای متدهای initialize یک بار قبل از اجرای همه‌ی تست‌ها استفاده شده است. این تضمین می‌کند که منابع مورد نیاز برای همه تست‌ها آماده هستند.

متد testInitialization دیگر نیازی به فراخوانی متدهای initialize ندارد زیرا این کار در setUpBeforeClass انجام میشود.

متد testResourceAvailability همانطور که بود باقی مانده است اما اطمینان حاصل می‌شود که منابع قبل از اجرای تست آماده هستند.

کد اصلاح شده:



سوال چهارم )

آزمون واحد (Unit Testing) یکی از روش‌های اصلی برای اطمینان از درستی عملکرد کد در برنامه‌نویسی است، اما وقتی صحبت از برنامه‌های چند ریسمانی (Multithreaded) می‌شود، موضوع پیچیده‌تر میشود. Race Conditions زمانی رخ می‌دهد که چندین ترد به طور همزمان به یک منبع مشترک دسترسی داشته باشند و ترتیب دسترسی به این منبع می‌تواند نتیجه را تغییر دهد. شناسایی و تست این شرایط با آزمون واحد بسیار دشوار است.

Thread Scheduling توسط سیستم‌عامل کنترل می‌شود و می‌تواند غیرقابل پیش‌بینی باشد. این موضوع باعث می‌شود که بازتولید دقیق مشکلات مربوط به زمان‌بندی در محیط تست واحد بسیار سخت باشد. قابلیت تکرارپذیری تست‌های واحد نیز چالشی دیگر است، به این معنا که با اجرای چندباره یک تست واحد باید نتایج مشابهی به دست آید، اما در برنامه‌های چند ریسمانی، عوامل خارجی مانند زمان‌بندی ریسمان‌ها و زمان‌های تاخیر می‌توانند باعث شوند که نتایج هر بار متفاوت باشد.

برای اطمینان از درستی برنامه‌های چند ریسمانی، علاوه بر آزمون واحد، می‌توان از آزمون‌های همزمانی (Concurrency Testing) استفاده کرد که به طور خاص برای شناسایی مشکلات مربوط به همزمانی طراحی شده‌اند و می‌توانند شامل تست‌های استرس (Stress Tests) و آزمون‌های بار (Load Tests) باشند.

استفاده از تکنیک‌های طراحی مناسب مانند عدم اشتراک‌گذاری داده‌های mutable بین ریسمان‌ها می‌تواند به کاهش احتمال بروز مشکلات کمک کند.