تست‌نرم افزار تکلیف یک و دو

علیرضادستمالچی ساعی

شناسه‌دانشجویی:993613026

دانشگاه:دانشگاه اصفهان

دوره:دوره تست نرم افزار

24نوامبر 2023

1.معرفی

دراین گزارش جامع، ما کاوش و ارزیابی یک سیستم رزرو قطار را با استفاده از روش آزمایش با رویکرددوگانه آغاز می کنیم. با استفاده از قدرت JUnit و Gherkin، ما بررسی کاملی از عملکردهای‌سیستم انجام می دهیم و از استحکام، قابلیت اطمینان و پایبندی به الزامات تجاری آن‌اطمینان می دهیم.

.برای‌ارزیابی دقیق جنبه های مختلف سیستم رزرو قطار استفاده می کنیم. این سناریوها شامل‌عملکردهای حیاتی مانند مدیریت شهر و قطار، مدیریت سفر، رزرو و لغو بلیط، مدیریت تاخیر،و مدیریت تبادل (با کرک) می شود JUnit یک چارچوب آزمایشی که به طور گسترده برای برنامه‌های جاوا پذیرفته شده است، پایه محکمی را برای آزمایش مبتنی بر سناریو فراهم می کند.از طریق ایجاد پنج سناریو دقیق، ما از ،JUnit

درتکمیل تست های JUnit خود، از Gherkin، یک قالب زبان-آگنوستیک، برای نوشتن سه سناریواضافی استفاده می کنیم. Gherkin توسعه رفتار محور (BDD) را با اجازه دادن به ما برای‌توصیف رفتار سیستم در قالب زبان طبیعی که برای ذینفعان فنی و غیر فنی به راحتی قابل‌درک است، تسهیل می کند.

این‌رویکرد یکپارچه یک ارزیابی جامع از سیستم رزرو قطار را تضمین می کند که هم تست واحدسطح پایین ارائه شده توسط JUnit و هم تست رفتاری سطح بالا که توسط Gherkin تسهیل‌می شود را پوشش می دهد. با ترکیب این دو روش آزمایش، هدف ما ارائه یک اعتبارسنجی‌قوی و کامل از عملکرد سیستم، اطمینان از آمادگی آن برای استقرار در سناریوهای دنیای‌واقعی است.

2بخش 1: مدیریت شهر و قطار

2.1مورد آزمایشی 1.1: یک شهر جدید اضافه کنید

شرح:بررسی کنید که یک شهر جدید می تواند به سیستم اضافه شود. پیش شرط‌ها:

- ^سیستم در حال اجراست.
- ^کاربر مجاز است یک شهر جدید اضافه کند.
- ^3 شهر جدید با نام های مختلف اضافه کنید (به عنوان مثال لس آنجلس، واشنگتن، تگزاس) از قبل.
- ^شهر جدیدی وجود دارد که باید با نام دیگری اضافه شود (به عنوان مثال کالیفرنیا).

مراحل:

1.اضافه کردن یک شهر جدید به سیستم با*addCity*() روش.

نتایج‌مورد انتظار:شهر را باید به لیست شهرهای سیستم اضافه کرد.

2.2مورد آزمایشی 1.2: یک قطار جدید اضافه کنید

شرح:بررسی کنید که قطار جدیدی می تواند به سیستم اضافه شود. پیش شرط ها:

- ^سیستم در حال اجراست.
- ^کاربر مجاز است یک قطار جدید اضافه کند.
- ^3 قطار جدید با نام های مختلف اضافه کنید (به عنوان مثال قطار گلوله 1، 2، 3)
- ^یک قطار جدید (قطار گلوله 4) برای اضافه شدن وجود دارد.

مراحل:

1.با استفاده از یک قطار جدید به سیستم اضافه کنید.*addTrains*() روش.

نتایج‌مورد انتظار:قطار جدید باید به لیست قطارهای سیستم اضافه شود.

3بخش 2: مدیریت سفر

3.1مورد آزمایشی 2.1: یک سفر جدید ایجاد کنید

شرح:بررسی کنید که با توجه به محدودیت های خاص می توان یک سفر جدید به سیستم اضافه‌کرد.
پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆکاربر مجاز است یک سفر جدید اضافه کند.

ˆالزامات سفر جدید برای ایجاد در دسترس هستند.

ˆایجاد 2 سفر (Trip1، Trip2).

مراحل:

1.جزئیات معتبر برای سفر جدید ارائه دهید: مبدا

2.جزئیات معتبر برای سفر جدید ارائه دهید: مقصد

3.ارائه جزئیات معتبر برای سفر جدید: قطار

4.جزئیات معتبر برای سفر جدید ارائه دهید: زمان حرکت

5.جزئیات معتبر برای سفر جدید ارائه دهید: زمان ورود

6.ایجاد سفر با استفاده از*createTrip*() روش.

نتایج‌مورد انتظار:سفر جدید باید ایجاد و در سیستم ثبت شود.

3.2مورد آزمایشی 2.2: لغو سفر

شرح:بررسی کنید که یک سفر در سیستم قابل لغو است. پیش شرط ها:

ˆسیستم در حال اجراست.

ˆکاربر مجاز به لغو سفر است.

ˆیک سفر با 2 بلیط رزرو شده در آن وجود دارد.

مراحل:

1.سفری را برای لغو انتخاب کنید*لغو سفر*() روش.

نتایج‌مورد انتظار:سفر انتخابی باید لغو شود و تمامی بلیط های مرتبط نیز باید لغو شوند.

3.3مورد آزمایشی 2.3: سفری با زمان بندی متناقض به قطار اضافه کنید

شرح:بررسی کنید که سفری با زمان های متناقض را نمی توان به فهرست سفرهای قطار اضافه‌کرد.
پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆکاربر مجاز به افزودن یک سفر است.

ˆیک سفر دارای زمان حرکت "10:00 25-11-2023" است و با ورود زمان"14:00 28-11-2023".

ˆسفر دوم دارای زمان حرکت "08:30 26-11-2023" است و با ورود زمان"12:30 01-12-2023".

مراحل:

1.با استفاده از اولین سفر را به آموزش اضافه کنید*createTrip*() روش.

2.یک سفر جدید (دومین) اضافه کنید که دارای زمان رهگیری با سفر قبلی است.

نتایج‌مورد انتظار:دومی نباید به لیست سفرهای قطار اضافه شود و الف*TripException*باید پرتاب‌شود.

3.4مورد آزمایشی 2.4: یک سفر اضافه کنید اما رسیدن قبل از حرکت است

شرح:بررسی کنید که سفری با زمان رسیدن آن قبل از زمان حرکت ایجاد نمی شود

پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆکاربر مجاز به افزودن یک سفر است.

مراحل:

1.یک سفر نامعتبر به قطار اضافه کنید (زمان حرکت ("10:00 25-11-2023") بعد از زمان ورود‌است ("14:00 25-10-2023")).

نتایج‌مورد انتظار:سفر نباید ایجاد شود و یک استثنا باید پرتاب شود (TripException).

4بخش 3: رزرو و کنسل کردن بلیط

4.1مورد آزمایشی 3.1: رزرو بلیط

شرح:بررسی کنید که اگر سفر به حداکثر تعداد مسافران نرسیده باشد، می توان بلیط جدیدی برای‌سفر رزرو کرد.
پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆکاربر مجاز به رزرو بلیط است.

ˆیک سفر در دسترس با بلیط های موجود برای رزرو وجود دارد.

مراحل:

1.سفر موجود ذکر شده در بالا را انتخاب کنید.

2.نام مسافر (به عنوان مثال "علیرضا") را ارائه دهید.

3.بلیط رزرو کنید.

نتایج‌مورد انتظار:یک بلیط جدید باید برای سفر انتخاب شده رزرو شود و بلیط باید در لیست بلیط‌های رزرو شده سفر مرتبط باشد.

4.2مورد آزمایشی 3.2: یک بلیط را لغو کنید

شرح:بررسی کنید که بلیط رزرو شده را می توان لغو کرد. پیش شرط ها:

ˆسیستم در حال اجراست.

ˆکاربر یک بلیط رزرو شده دارد.

ˆیک سفر با بلیط رزرو شده مرتبط با آن وجود دارد.

مراحل:

1.بلیط رزرو شده را از سفر ذکر شده انتخاب کنید.

نتایج‌مورد انتظار:بلیط انتخاب شده باید لغو شود و لیست بلیط های سفر و لیست بلیط های کنسل‌شده باید بر اساس آن به روز شوند.

4.3مورد آزمایشی 3.3: یک بلیط برای یک سفر کامل رزرو کنید

شرح:بررسی کنید که بلیط برای سفری با حداکثر مسافر نمی تواند ایجاد شود.

پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆاین سفر دارای 3 بلیط در دسترس برای رزرو است.

ˆهر 3 بلیط توسط افراد دیگر رزرو شده است.

مراحل:

1.یک بلیط برای سفر با حداکثر مسافر برای یک فرد جدید ایجاد کنید.

نتایج‌مورد انتظار:بلیط نباید رزرو شود و باید خطا بدهد (*ReservationException*Throw).

5بخش 4: مدیریت تاخیر

5.1مورد تست 4.1: اضافه کردن تاخیر خروج به یک سفر

شرح:بررسی کنید که تاخیر خروج می تواند به یک سفر اضافه شود و زمان واقعی حرکت را به روزمی کند.
پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆیک سفر برای تاخیر در دسترس است.

ˆقرار است 1 روز تاخیر 2 ساعت به سفر اضافه شود.

مراحل:

1.سفر ایجاد شده را انتخاب کنید.

2.تاخیر خروج را با میزان تاخیر ذکر شده اضافه کنید.

نتایج‌مورد انتظار:تأخیر حرکت باید به سفر اضافه شود و زمان واقعی حرکت باید بر اساس آن به‌روز شود.

5.2مورد تست 4.2: اضافه کردن تاخیر رسیدن به یک سفر

شرح:بررسی کنید که تاخیر ورود می تواند به یک سفر اضافه شود و زمان واقعی ورود را به روز می‌کند.
پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆیک سفر برای تاخیر در دسترس است.

ˆقرار است 1 روز تاخیر 2 ساعت به سفر اضافه شود.

مراحل:

1.سفر ایجاد شده را انتخاب کنید.

2.تاخیر ورود را با میزان تاخیر ذکر شده اضافه کنید.

نتایج‌مورد انتظار:تأخیر ورود باید به سفر اضافه شود و زمان واقعی ورود باید متناسب با آن به روزشود.

5.3مورد آزمایشی 4.3: اضافه کردن تاخیر خروج بیش از مدت به تاریخ ورود

شرح:بررسی کنید که تاریخ حرکت نمی تواند از زمان رسیدن پس از تأخیر عبور کند.

پیش‌شرط ها:

ˆسیستم در حال اجراست.

ˆیک سفر برای تاخیر در دسترس است.

ˆاین سفر دارای ساعت حرکت "10:00 25-11-2023" و زمان رسیدن است. از"10:00 27-11-2023".

ˆ4 روز تاخیر داریم.

مراحل:

1.یک سفر را انتخاب کنید.

2.اضافه کردن تاخیر خروج (4 روز) بیشتر از مدت زمان (2 روز).

نتایج‌مورد انتظار:تأخیر خروج نباید به سفر اضافه شود، یا زمان ورود باید به روز شود، یا باید استثناءدر نظر گرفته شود.

6 بخش5: مدیریت مبادلات (خیار)

6.1 Gherkin 5.1: یافتن‌تمام بلیط های قابل تعویض

شخصی‌می خواهد بلیط خود را تعویض کند، بلیط های موجود برای مبادلات باید نشان داده شودتا او بتواند از بین آنها انتخاب کند.
داده‌شده→ ما 2 سفر داریم (سفر 1 و سفر 2) و 1 بلیط از trip1 رزرو می کنیم. چه زمانی→ درخواست‌برای دیدن بلیط های موجود برای تعویض
سپس→ لیستی از تمام بلیط های موجود باید نشان داده شود، در این مورد یکی است (طول لیست‌باید 1 باشد).

6.2 Gherkin 5.2: سفرقبلی و بعدی قطار را دریافت کنید

شخصی‌سفری با قطار دارد و می خواهد سفر قبلی و بعدی آن قطار را ببیند

داده‌شده→ قطار بیش از سه سفر دارد و یک سفر قبلی و بعدی دارد چه زمانی→ درخواست برای‌دیدن سفرهای قبلی و بعدی قطار (با فرض اینکه اطلاعات سفر میانی داریم)

سپس→ سفرهای جانشین و سلف صحیح در صورت درخواست نشان داده می شوند

6.3 Gherkin 5.3: یک‌بلیط را با موفقیت تعویض کنید

کسی‌می خواهد بلیط خود را تعویض کند، باید روند تعویض صحیح باشد

داده‌شده→ شرایط مبادله برآورده شده است (یک سفر با 2 بلیط رزرو شده وجود دارد که قابل تعویض‌است)
چه‌زمانی→ درخواست تعویض بلیط از سفر مذکور
سپس→ تعویض باید برای یک سفر انجام شود و بلیط تعویض شده باید لغو شود و به لیست بلیط‌های لغو شده منتقل شود.

7توضیح و گزارش کد

به‌طور کلی 15 آزمون نوشته شده است (12 با JUnit و 3 با Gherkin). ما هر یک از آنها را مرور خواهیم‌کرد و در صورت نیاز به تفصیل توضیح خواهیم داد.

7.1تنظیم تست ها

راه‌اندازی JUnit و Cucumber در پروژه Maven شامل افزودن وابستگی های لازم به فایل pom.xml پروژه است. JUnit یک چارچوب آزمایشی پرکاربرد برای جاوا است، در حالی که Cucumber توسعه رفتار محور را با استفاده از نحو Gherkin تسهیل می کند. در زیر یک متن نمونه‌وجود دارد که نحوه تنظیم این وابستگی ها را در پروژه Maven توضیح می دهد:

بایدوابستگی های زیر را اضافه کرد:

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.8.1</version>
        <scope>test</scope>
    </dependency>


    <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>5.1.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>7.14.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-core</artifactId>
        <version>7.14.0</version>
    </dependency>
```

شکل1: وابستگی های مورد نیاز

7.2پیش شرط ها

درزمینه تست JUnit، پیش شرط ها به شرایط ضروری اشاره می کنند که قبل از اجرای هر آزمونی‌باید رعایت شوند. در مورد ما، هر تست JUnit نیاز به یک نمونه اولیه درست شده از TicketReservationSystem برای کنترل و ارزیابی رفتار کد اساسی دارد.

برای‌اجرای یک تنظیم ثابت قبل از هر آزمون، حاشیه نویسی @BeforeEach در چارچوب تستJUnit استفاده می شود. این حاشیه نویسی نشان می دهد که یک روش تنظیم تعیین شده‌باید قبل از شروع هر آزمایش جداگانه اجرا شود. در سناریوی ما، این روش راه اندازی مسئول‌ایجاد و پیکربندی نمونه اساسی TicketReservationSystem است.

```
@BeforeEach
public void setUp() {
    // Set up the default ZoneId and create a TicketReservationSystem
    zoneId = ZoneId.systemDefault();
    trs = new TicketReservationSystemImpl(zoneId);
    origin = new CityImpl( name: "City A");
    destination = new CityImpl( name: "City B");
}
```

شکل2: روش با قبل از هر

سایرپیش شرط ها در بین موارد آزمون منحصر به فرد نیستند، برخی از پیش شرط ها در هرروش آزمون به صورت جداگانه ایجاد می شوند.

7.3بخش 1: مدیریت شهر و قطار

7.3.1مورد آزمایشی 1.1: یک شهر جدید اضافه کنید

کدبرای آزمایش اینکه آیا یک شهر می تواند به درستی به لیست شهر اضافه شود در زیر آمده است:
ابتدالیستی از شهرها با اندازه 3 ایجاد می کنیم. سپس یک شهر جدید با نام "California" ایجاد می‌شود و به لیست شهر اضافه می شود. با متغیر*شهر وجود دارد* و با تکرار از طریق لیست تمام‌شهرهای موجود، می توانیم بررسی کنیم که آیا شهر در لیست است یا خیر.

```java
@Test
public void addNewCity(){
    // Add a city to the system before executing the test
    City existingCity1 = new CityImpl( name: "Los Angeles");
    trs.addCity(existingCity1);
    City existingCity2 = new CityImpl( name: "Washington");
    trs.addCity(existingCity2);
    City existingCity3 = new CityImpl( name: "Texas");
    trs.addCity(existingCity3);

    City city = new CityImpl( name: "California");
    trs.addCity(city);
    List<City> cityList = trs.getCities();

    boolean cityExists = false;
    for (City c : cityList) {
        if (c.getName().equals(city.getName())) {
            cityExists = true;
            break;
        }
    }
    assertTrue(cityExists);
}
```

شکل3: مورد آزمون 1.1

7.3.2مورد آزمایشی 1.2: یک قطار جدید اضافه کنید

این‌روش بررسی می کند که آیا می توان یک سیستم جدید را به درستی در TrainReservationSystem ما اضافه کرد یا خیر. قطعه کد در زیر قابل مشاهده است:

```java
@Test
public void AddNewTrain(){
    Train train1 = new TrainImpl( name: "Bullet Train",  maxPassengers: 100);
    trs.addTrain(train1);
    Train train2 = new TrainImpl( name: "Bullet Train",  maxPassengers: 100);
    trs.addTrain(train2);
    Train train3 = new TrainImpl( name: "Bullet Train",  maxPassengers: 100);
    trs.addTrain(train3);

    Train train = new TrainImpl( name: "Bullet Train",  maxPassengers: 100);
    trs.addTrain(train);

    List<Train> trains = trs.getAllTrains();

    boolean trainExists = false;
    for (Train t : trains) {
        if (t.getName().equals(train.getName())) {
            trainExists = true;
            break;
        }
    }
    assertTrue(trainExists);
}
```

شکل4: مورد آزمون 1.2

همانطورکه مشاهده می شود، در این روش تست، یک لیست اولیه از 3 قطار ایجاد می شود.سپس، یک نمونه جدید از قطار با نام*BulletTrain*ایجاد شده و به لیست قطارها اضافه شده‌است. حلقه foreach پس از اضافه کردن قطار، اصلاح روند اضافه کردن قطارهای جدید را بررسی‌می کند. ارزش*trainExists*پس از تکرار باید درست باشد*قطارها*فهرست

7.4بخش 2: مدیریت سفر

7.4.1مورد آزمایشی 2.1: یک سفر جدید ایجاد کنید

این‌تست واحد برای آزمایش است*createTrip*روش در کلاس "TrainReservationSystem"
.در کد، 2 سفر ایجاد شده و طول لیست مورد انتظار سفرها 2 در نظر گرفته شده است.

```
@Test
public void createNewValidTrip() throws TripException {
    City origin1 = new CityImpl( name: "City A");
    City destination1 = new CityImpl( name: "City B");
    Train train1 = new TrainImpl( name: "Express Train",  maxPassengers: 200);
    Instant departureTime1 = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime1 = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip1 = trs.createTrip(origin1, destination1, train1, departureTime1, arrivalTime1);


    City origin2 = new CityImpl( name: "City C");
    City destination2 = new CityImpl( name: "City D");
    Train train2 = new TrainImpl( name: "Local Train",  maxPassengers: 100);
    Instant departureTime2 = TimeManagement.createInstant( dateTimeString: "2023-12-02 08:00", zoneId);
    Instant arrivalTime2 = TimeManagement.createInstant( dateTimeString: "2023-12-02 10:00", zoneId);

    Trip trip2 = trs.createTrip(origin2, destination2, train2, departureTime2, arrivalTime2);

    List<Trip> trips = trs.getAllTrips();
    assertEquals(trips.toArray().length,  actual: 2);
}
```

شکل5: مورد آزمایشی 2.1

7.4.2مورد آزمایشی 2.2: لغو سفر

این‌مورد آزمایشی برای بررسی این است که آیا یک سفر معتبر می تواند به درستی لغو شود یا خیر.

```
@Test
public void cancelAnExistingTrip() throws TripException {
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 200);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);

    trs.cancelTrip(trip);
    List<Trip> trips = trs.getAllTrips();
    assertEquals(trips.toArray().length,  actual: 0);
}
```

شکل6: مورد آزمایشی 2.2

درکد بالا، ابتدا یک نمونه از اشیاء مورد نیاز برای ایجاد یک سفر ایجاد می کنیم، پس از ایجادسفر، همان سفر ایجاد شده قبلی را لغو می کنیم و سپس بررسی می کنیم که طول لیست های‌سفر 1 کاهش یابد (در این حالت: 0).

7.4.3مورد آزمایشی 2.3: سفری با زمان بندی متناقض به قطار اضافه کنید

دراین مورد آزمایشی، ما سعی می کنیم دو سفر برای قطار ایجاد کنیم که زمان بندی متناقض داشته‌باشند و یکی را نتوان ایجاد کرد.

```
@Test
public void createATripWithConflict() throws TripException{
    City origin1 = new CityImpl( name: "City A");
    City destination1 = new CityImpl( name: "City B");
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 200);
    Instant departureTime1 = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime1 = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip1 = trs.createTrip(origin1, destination1, train, departureTime1, arrivalTime1);


    City origin2 = new CityImpl( name: "City C");
    City destination2 = new CityImpl( name: "City D");
    Instant departureTime2 = TimeManagement.createInstant( dateTimeString: "2023-11-26 08:30", zoneId);
    Instant arrivalTime2 = TimeManagement.createInstant( dateTimeString: "2023-12-01 12:30", zoneId);

    assertThrows(TripException.class, () -> {
        Trip trip2 = trs.createTrip(origin2, destination2, train, departureTime2, arrivalTime2);
    });
}
```

شکل7: مورد آزمایشی 2.3

7.4.4مورد آزمایشی 2.4: یک سفر اضافه کنید اما رسیدن قبل از حرکت است

دراین مورد آزمایشی، می خواهیم بررسی کنیم که آیا می توان سفری را با زمان رسیدن آن قبل اززمان حرکت ایجاد کرد. این عمل باید یک استثنا ایجاد کند. همانطور که در تصویر مشخص استیک سفر ایجاد شده و زمان بندی نامعتبر به آن داده شده است. سپس، کد انتظار دریافت TripException را دارد.

```
@Test
public void createATripButArrivalIsBeforeDeparture(){
    City origin = new CityImpl( name: "City A");
    City destination = new CityImpl( name: "City B");
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 200);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-10-25 14:00", zoneId);

    assertThrows(TripException.class, ()->{
        Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    });
}
```

شکل8: مورد آزمایشی 2.4

7.5بخش 3: رزرو و کنسل کردن بلیط

7.5.1مورد آزمایشی 3.1: رزرو بلیط

دراین مورد آزمایشی، ما توانایی سیستم را برای رزرو موفقیت آمیز بلیط برای یک سفر مشخص‌بررسی می کنیم. این آزمون شامل ایجاد یک سفر، تلاش برای رزرو بلیط و اطمینان از ثبت‌صحیح بلیط رزرو شده در سیستم است.

```java
@Test
public void bookingATicket() throws TripException, ReservationException {
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 200);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    trip.bookTicket( passengerName: "Alireza");

    List<Ticket> tickets = trs.getAllBookedTickets();
    List<Ticket> actual_tickets = trip.getBookedTickets();

    assertEquals(tickets, actual_tickets);
}
```

شکل9: مورد آزمایشی 3.1

7.5.2مورد آزمایشی 3.2: لغو یک بلیط

این‌مورد آزمایشی عملکرد لغو بلیط قبلا رزرو شده را ارزیابی می کند. پس از رزرو بلیط در یک سفرمشخص، سیستم برای تأیید اینکه فرآیند لغو به درستی بلیط را از لیست بلیط های رزرو شده‌حذف می کند، آزمایش می شود.

```java
@Test
public void cancelATicket() throws TripException, ReservationException{
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 200);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    trip.bookTicket( passengerName: "Alireza");

    List<Ticket> tickets = trs.getAllBookedTickets();
    trip.cancelTicket(tickets.get(0));

    boolean ticketIsCancelled = trip.getBookedTickets().isEmpty();

    List<Ticket> canceled = trs.getAllCancelledTickets();

    assertTrue(ticketIsCancelled);
    assertFalse(canceled.isEmpty());
}
```

شکل10: مورد آزمایشی 3.2

7.5.3مورد آزمایشی 3.3: رزرو بلیط برای یک سفر کامل

این‌تست رفتار سیستم را هنگام اقدام به رزرو بلیط برای سفری که به ظرفیت کامل خود رسیده استبررسی می کند. بررسی می کند که آیا سیستم به طور مناسب سناریوهایی را که در آن صندلی های‌موجود در قطار قبلاً به طور کامل رزرو شده اند انجام می دهد یا خیر.

```java
@Test
public void bookAnInvalidTicket() throws TripException, ReservationException{
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 3);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);


    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    trip.bookTicket( passengerName: "Alireza Saei");
    trip.bookTicket( passengerName: "David Goggins");
    trip.bookTicket( passengerName: "Donald Trump");

    assertThrows(ReservationException.class, () -> {
        trip.bookTicket( passengerName: "Andrew Tate");
    });
}
```

شکل11: مورد آزمایشی 3.3

7.6بخش 4: مدیریت تاخیر

7.6.1مورد آزمایشی 4.1: اضافه کردن تاخیر خروج به یک سفر

این‌مورد آزمایشی توانایی سیستم را برای کنترل و انعکاس صحیح تاخیر خروج برای یک سفر مشخص‌تأیید می کند. این آزمایش شامل ایجاد یک سفر، اضافه کردن تاخیر خروج، و بررسی اینکه‌آیا سیستم به طور دقیق این تاخیر را ثبت کرده و در برنامه سفر اعمال می کند، می باشد.

```java
@Test
public void addDepartureDelayToATrip() throws TripException{
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 3);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    Duration dur_day = Duration.ofDays(1);
    Duration dur_hour = Duration.ofHours(2);
    Duration dur_merged = dur_day.plus(dur_hour);
    trip.addDepartureDelay(dur_merged);

    assertTrue(trs.getAllTrips().get(0).isDelayed());
    assertEquals(trip.getDepartureDelay(), dur_merged);
    assertEquals(trip.findRealDepartureTime(), TimeManagement.createInstant( dateTimeString: "2023-11-26 12:00", zoneId));
    assertEquals(trip.getPlannedDepartureTime(), departureTime);
}
```

شکل12: مورد آزمایشی 4.1

7.6.2مورد تست 4.2: اضافه کردن تاخیر ورود به یک سفر

دراین مورد آزمایشی، پاسخ سیستم به اضافه کردن تاخیر رسیدن به سفر ارزیابی می شود. پس ازمعرفی تاخیر ورود، تست بررسی می کند که آیا سیستم به طور مناسب زمان رسیدن سفر را باتوجه به تاخیر به روزرسانی می کند یا خیر.

```java
@Test
public void addArrivalDelayToATrip() throws TripException{
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 10);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-28 14:00", zoneId);

    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    Duration dur_day = Duration.ofDays(1);
    Duration dur_hour = Duration.ofHours(2);
    Duration dur_merged = dur_day.plus(dur_hour);
    trip.addArrivalDelay(dur_merged);

    // There is an error in isDelayed() for checking delay for arrivalTime
    assertTrue(trs.getAllTrips().get(0).isDelayed());
    assertEquals(trip.getArrivalDelay(), dur_merged);
    assertEquals(trip.findRealArrivalTime(), TimeManagement.createInstant( dateTimeString: "2023-11-29 16:00", zoneId));
    assertEquals(trip.getPlannedArrivalTime(), arrivalTime);
}
```

شکل13: مورد آزمایشی 4.2

7.6.3مورد آزمایشی 4.3: اضافه کردن تاخیر خروج بیش از مدت زمان به تاریخ ورود

این‌تست رفتار سیستم را هنگام تلاش برای اضافه کردن تاخیر خروجی که از مدت زمان لازم برای‌رسیدن به تاریخ رسیدن سفر فراتر می رود، بررسی می کند. هدف این است که اطمینان حاصل‌شود که سیستم چنین مواردی را با ظرافت مدیریت می کند و از تاخیرهای غیرواقعی که بیش‌از تاریخ ورود است جلوگیری می کند.

```java
@Test
public void addDepartureDelayToATripMoreThanDuration() throws TripException{
    Train train = new TrainImpl( name: "Express Train",  maxPassengers: 10);
    Instant departureTime = TimeManagement.createInstant( dateTimeString: "2023-11-25 10:00", zoneId);
    Instant arrivalTime = TimeManagement.createInstant( dateTimeString: "2023-11-27 10:00", zoneId);

    Trip trip = trs.createTrip(origin, destination, train, departureTime, arrivalTime);
    Duration dur = Duration.ofDays(4);
    trip.addDepartureDelay(dur);

    assertTrue(trs.getAllTrips().get(0).isDelayed());
    assertEquals(trip.getDepartureDelay(), dur);
    assertEquals(trip.findRealDepartureTime(), TimeManagement.createInstant( dateTimeString: "2023-11-29 10:00", zoneId));
    assertEquals(trip.getPlannedDepartureTime(), departureTime);
    // Now departure time is after arrival time that is not true
    assertTrue(trip.findRealArrivalTime().isAfter(trip.findRealDepartureTime()));
}
```

شکل14: مورد آزمایشی 4.2

7.7بخش 5: مدیریت بورس

7.7.1 Gherkin 5.1: یافتن‌همه بلیط های قابل تعویض

این‌سناریوی Gherkin مراحل یافتن تمام بلیط هایی را که برای تعویض در سیستم موجود است،تشریح می کند. احتمالاً شامل شناسایی معیارهای واجد شرایط بودن مبادله و ارائه لیستی‌از بلیط هایی است که این معیارها را برآورده می کنند.

```
@Given("Exchange requirements are met")
public void exchange_requirements_are_met() throws TripException{
    trs = new TicketReservationSystemImpl(zoneId);

    City origin1 = new CityImpl( name: "City A");
    City destination1 = new CityImpl( name: "City B");
    Train train1 = new TrainImpl( name: "Express Train",  maxPassengers: 20);
    Instant departureTime1 = Instant.parse( text: "2023-11-25T10:00:00Z");
    Instant arrivalTime1 = Instant.parse( text: "2023-11-28T14:00:00Z");
    trip1 = trs.createTrip(origin1, destination1, train1, departureTime1, arrivalTime1);

    Train train2 = new TrainImpl( name: "Bullet Train",  maxPassengers: 20);
    Instant departureTime2 = Instant.parse( text: "2023-11-27T10:00:00Z");
    Instant arrivalTime2 = Instant.parse( text: "2023-11-29T14:00:00Z");
    trip2 = trs.createTrip(origin1, destination1, train2, departureTime2, arrivalTime2);
}

 Alireza Saei
@When("Request to see available tickets for exchange")
public void request_to_see_available_tickets_for_exchange() throws ReservationException{
    Ticket t1 = trip1.bookTicket( passengerName: "Alireza");
    tripsList = trs.findPossibleExchanges(t1);
}

 Alireza Saei
@Then("A list of all available tickets are shown")
public void a_list_of_all_available_tickets_are_shown() { assertEquals(tripsList.toArray().length,  actual: 1); }
```

شکل15: قرقره 5.1

7.7.2 Gherkin 5.2: سفرقبلی و بعدی قطار را دریافت کنید

این‌سناریوی Gherkin بر بازیابی اطلاعات مربوط به سفرهای قبلی و بعدی قطار تمرکز دارد. این‌شامل تعامل با سیستم برای به دست آوردن جزئیات در مورد ترتیب زمانی سفرهای قطار است‌و به کاربران کمک می کند تا سفر خود را به طور موثر برنامه ریزی و مدیریت کنند.

```java
@Given("Train has more that three trips and has a previous and next trip")
public void train_has_more_that_three_trips_and_has_a_previous_and_next_trip() throws TripException {
    trs = new TicketReservationSystemImpl(zoneId);
    train = new TrainImpl( name: "Bullet Train",  maxPassengers: 20);

    City origin1 = new CityImpl( name: "City A");
    City destination1 = new CityImpl( name: "City B");
    Instant departureTime1 = Instant.parse( text: "2023-12-10T10:00:00Z");
    Instant arrivalTime1 = Instant.parse( text: "2023-12-15T14:00:00Z");
    trip1 = trs.createTrip(origin1, destination1, train, departureTime1, arrivalTime1);

    // Created 2 more (Deleted for screenshot)
}

 Alireza Saei
@When("Request to see available earlier and later trips of a train")
public void request_to_see_available_earlier_and_later_trips_of_a_train() throws TripException{
    previous_trip = trs.findPreviousTripOfTrain(train, trip2);
    next_trip = trs.findNextTripOfTrain(train, trip2);
}

 Alireza Saei
@Then("The correct successor and predecessor trips are shown")
public void the_correct_successor_and_predecessor_trips_are_shown() {
    assertTrue(previous_trip.isPresent());
    assertTrue(next_trip.isPresent());

    assertEquals(previous_trip.get(), trip1);
    assertEquals(next_trip.get(), trip3);
}
```

شکل16: قرقره 5.2

یک‌بلیط را با موفقیت تعویض کنید :7.7.3 Gherkin 5.3

این‌سناریوی Gherkin مراحل تعویض موفقیت آمیز بلیط را شرح می دهد. این شامل فرآیند انتخاب‌بلیط برای مبادله، تأیید واجد شرایط بودن آن، و اجرای عملیات مبادله، حصول اطمینان ازمدیریت یکپارچه مبادله توسط سیستم است.

```
@Given("Exchanging requirements are met")
public void exchanging_requirements_are_met() throws TripException, ReservationException{
    trs = new TicketReservationSystemImpl(zoneId);

    City origin1 = new CityImpl( name: "City A");
    City destination1 = new CityImpl( name: "City B");
    Train train1 = new TrainImpl( name: "Express Train",  maxPassengers: 20);
    Instant departureTime1 = Instant.parse( text: "2023-11-25T10:00:00Z");
    Instant arrivalTime1 = Instant.parse( text: "2023-11-28T14:00:00Z");
    trip = trs.createTrip(origin1, destination1, train1, departureTime1, arrivalTime1);

    t = trip.bookTicket( passengerName: "Alireza Saei");
    trip.bookTicket( passengerName: "Jeff Bezos");
    trip.bookTicket( passengerName: "Arthur Morgan");
}

Alireza Saei
@When("Request to exchange a ticket from a trip")
public void request_to_exchange_a_ticket_from_a_trip() throws ReservationException {
    exchanged_ticket = t.exchangeTicket(trip);
}

Alireza Saei
@Then("Exchange must be done for a trip")
public void exchange_must_be_done_for_a_trip() { assertTrue(t.isCancelled()); }
```

شکل17: قرقره 5.3

8توضیح لینک ویدیو

برای‌مشاهده ویدیوی ضبط شده برای این تکلیف اینجا را کلیک کنید.