دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش تکالیف اول و دوم آزمون نرم افزار

Library Management

پدیدآورنده:

محمد امین کیانی

4003613052

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد راهنما: جناب اقای دکتر شعرباف

نیمسال اول تحصیلی 04-1403

فهرست مطالب

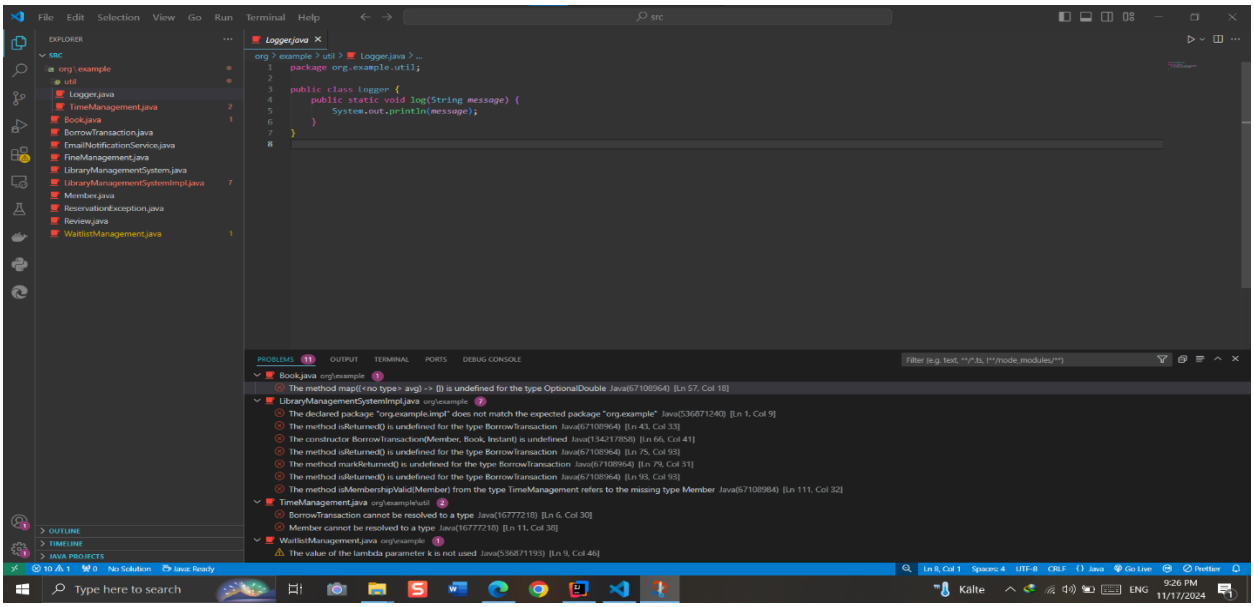
مستندات

لینک ویدیو

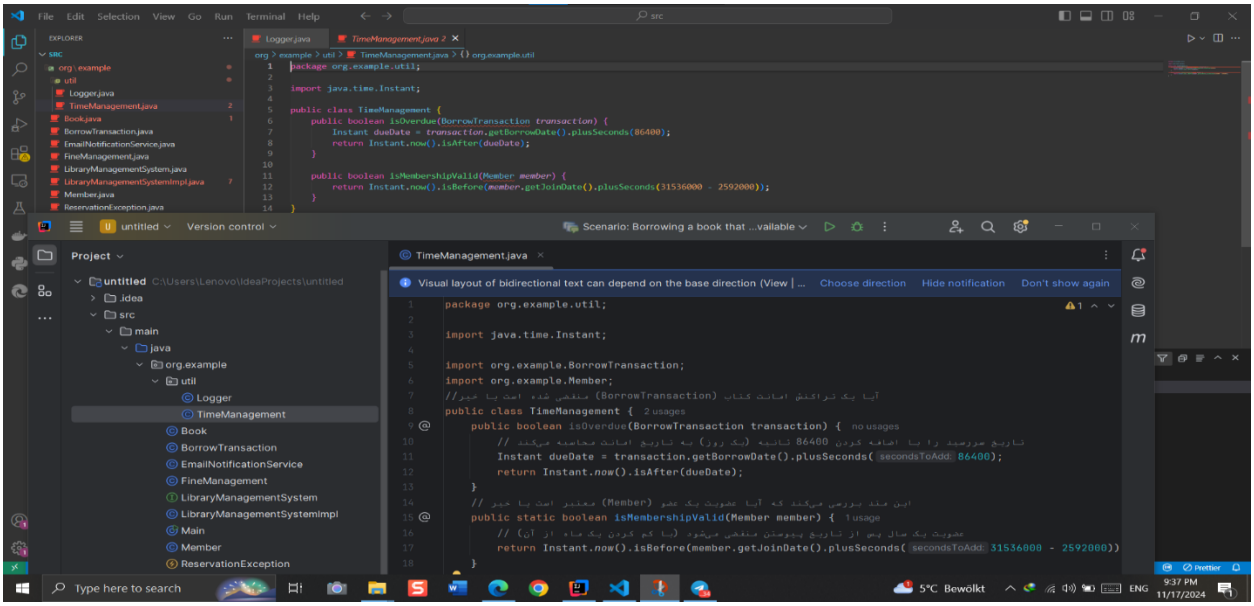
https://drive.google.com/file/d/1QbHoTE4BOdBBsirVGxpAVBqZy9JgXfP6/view?usp=sharing

بررسی خطاهای اولیه

All Errors

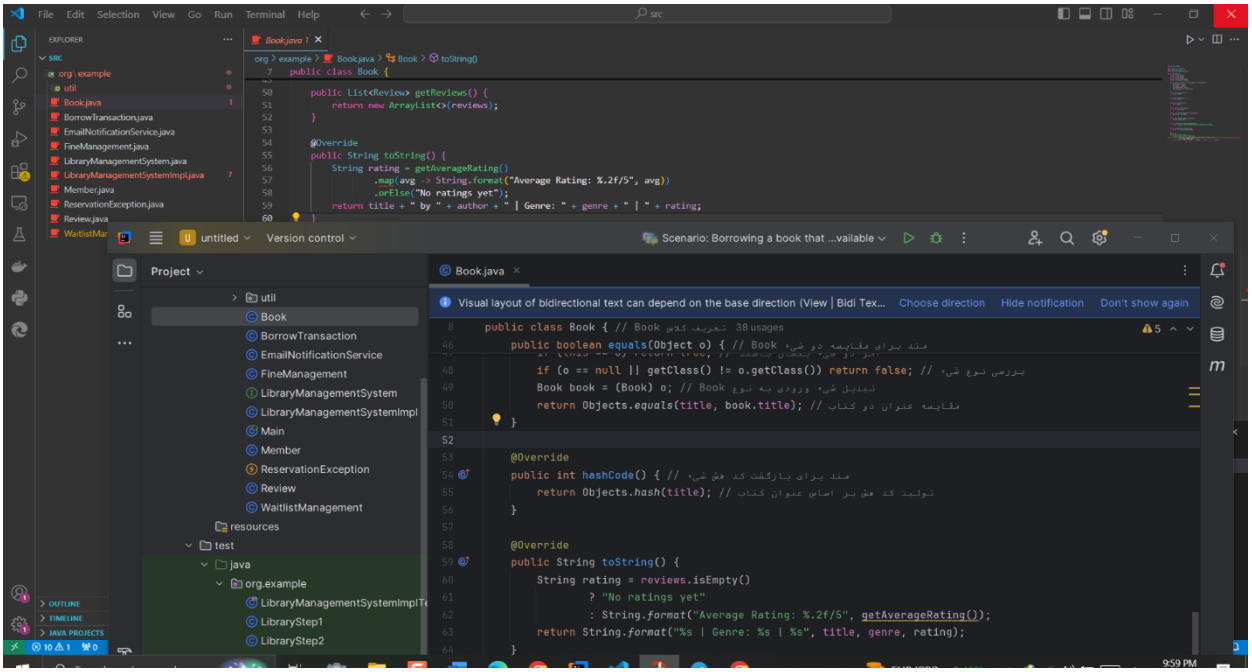


Fix Time



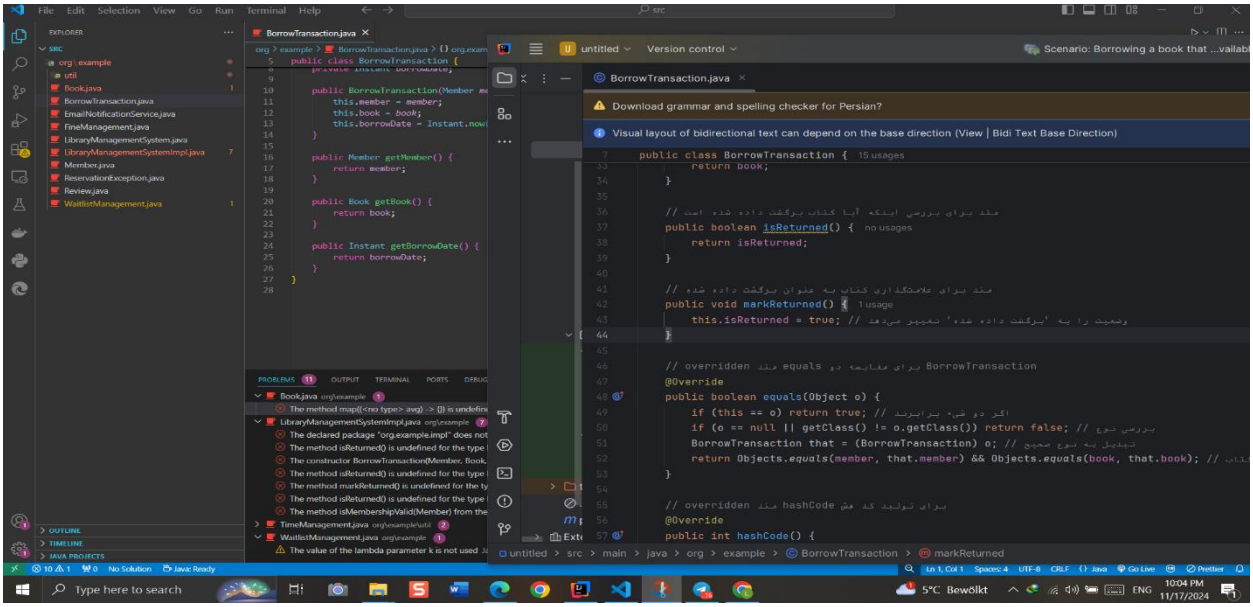
پکیج‌ها به صورت ناقص تعریف شده است و متد isMembershipValid نیز به صورت static باید تعریف شود تا بدون نیاز به ایجاد یک نمونه از کلاس، آن را بتوانیم فراخوانی کنیم.

Fix Book

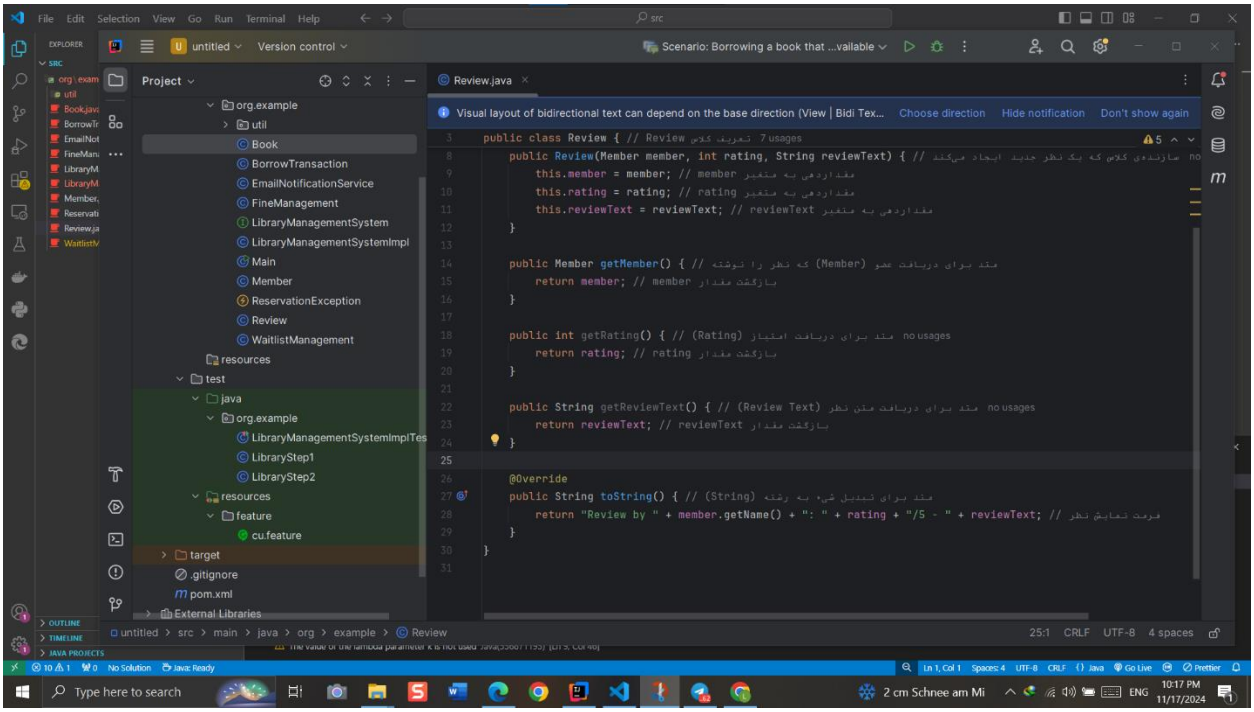


از متد map و orElse برای بررسی وجود امتیاز استفاده شده‌است، اما به درستی مدیریت نمی‌شود. پس باید بررسی شود که آیا امتیاز موجود است؟(تایپ دابل، مپ ندارد.)

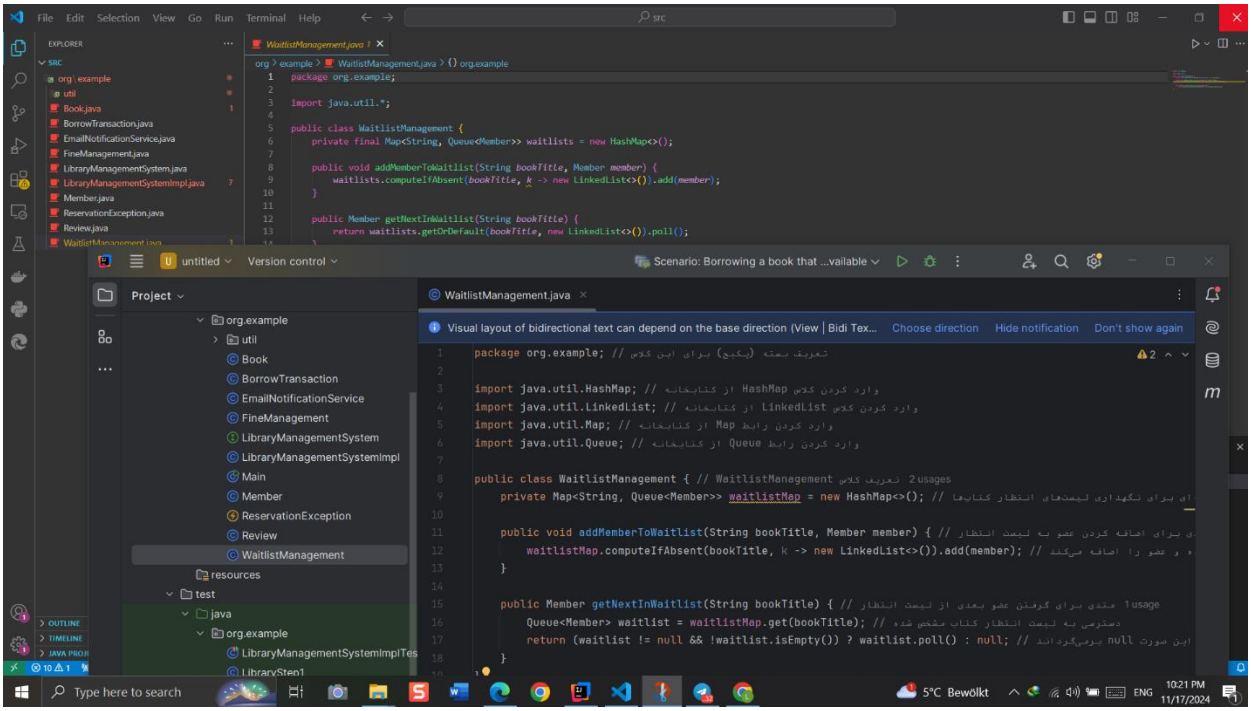
Fix Add Borrow



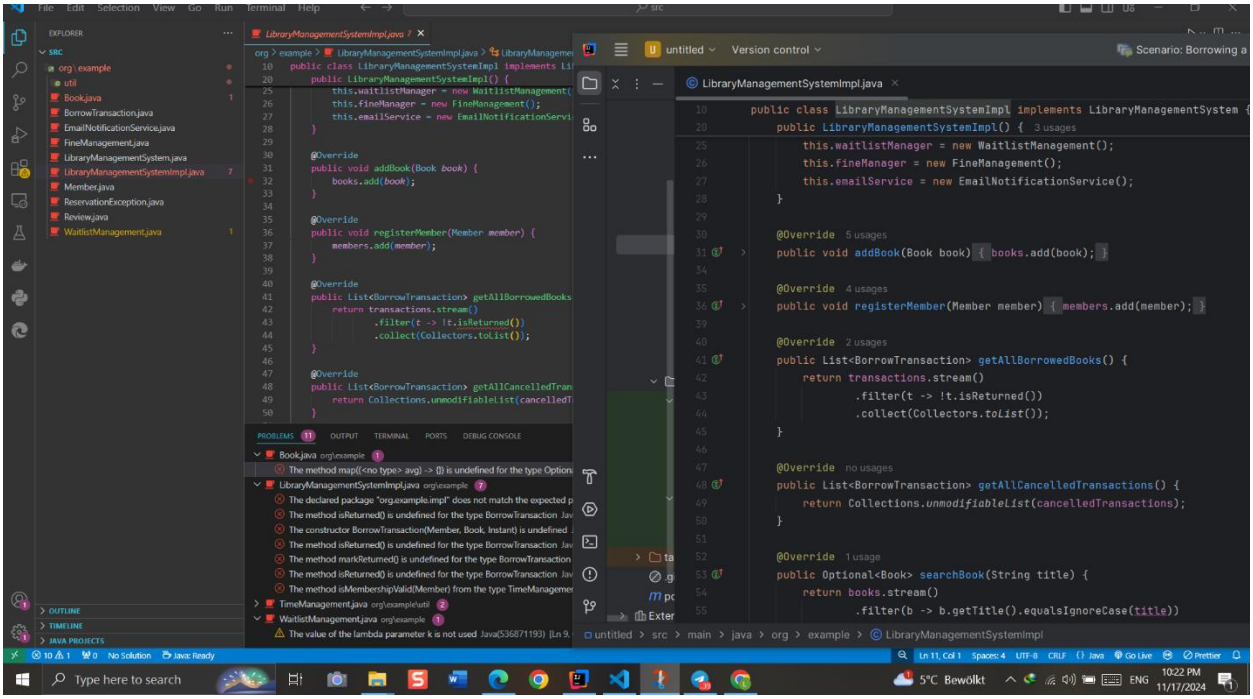
Fix Rate For Book In Review

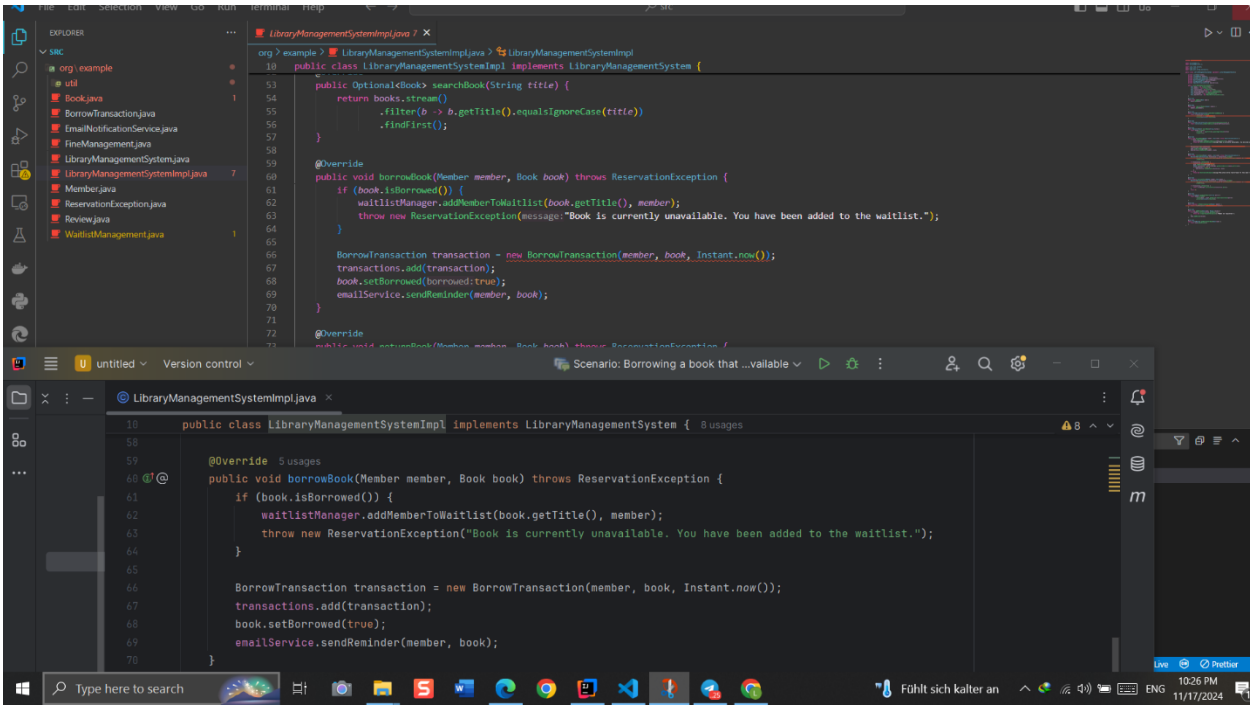


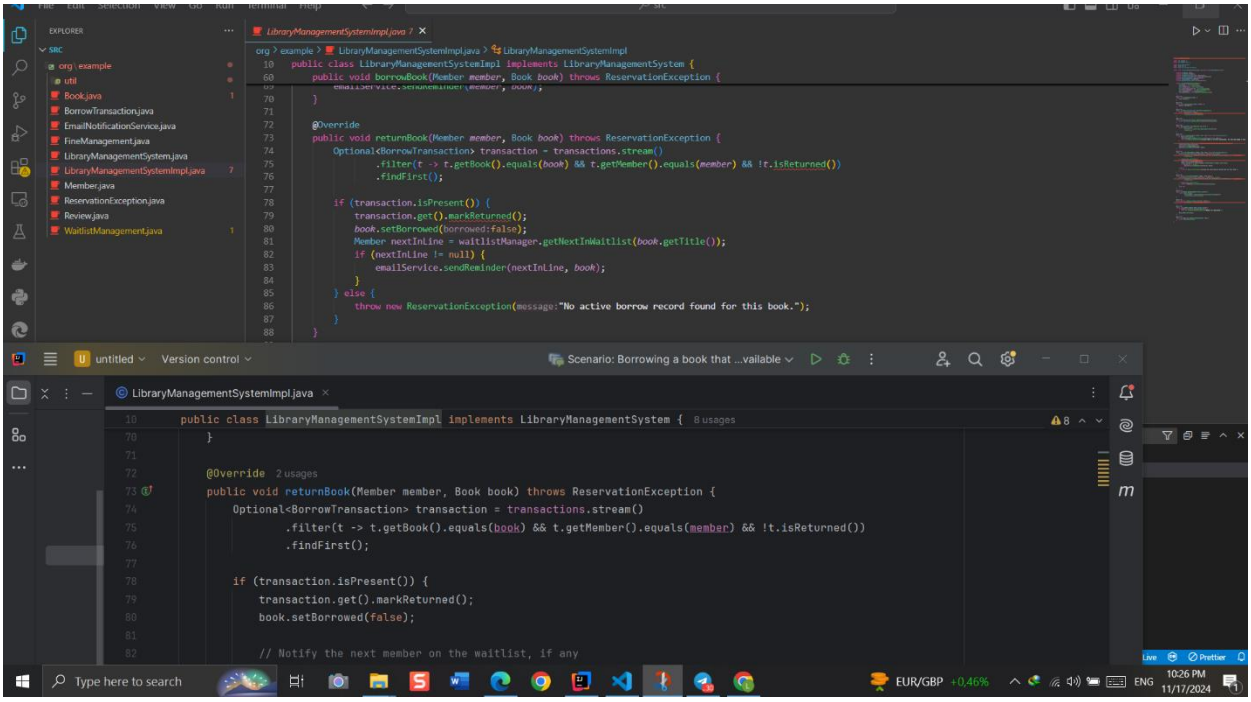
Fix Wait

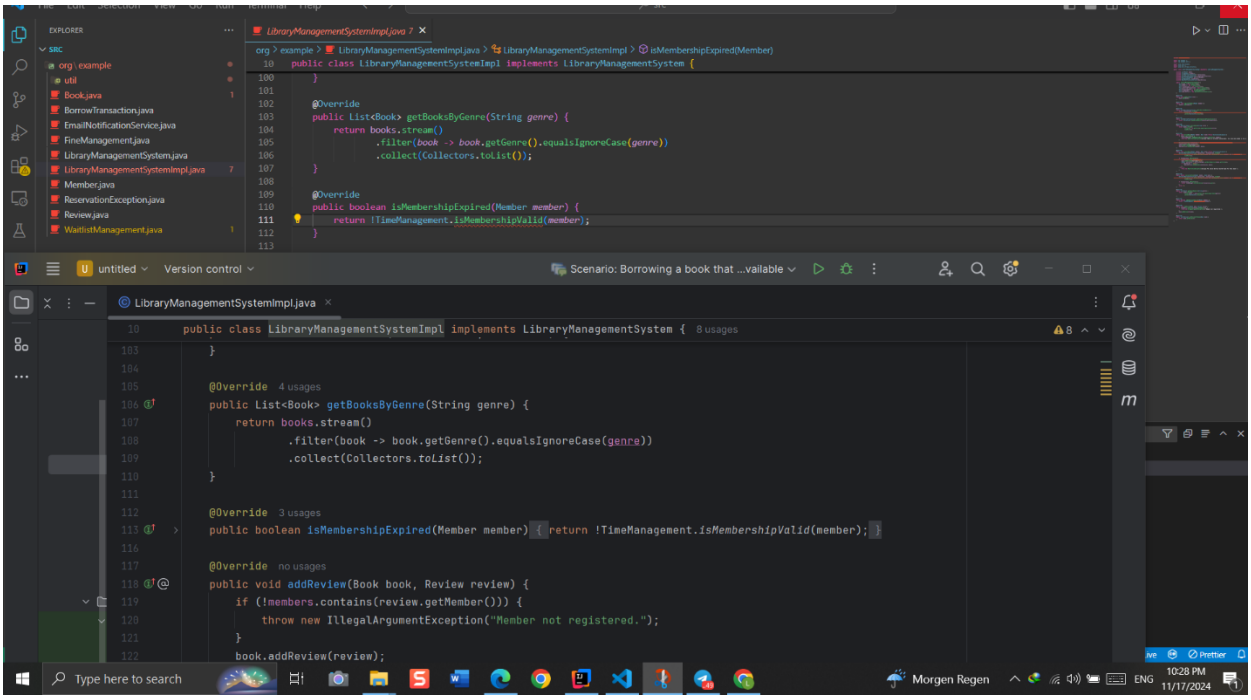


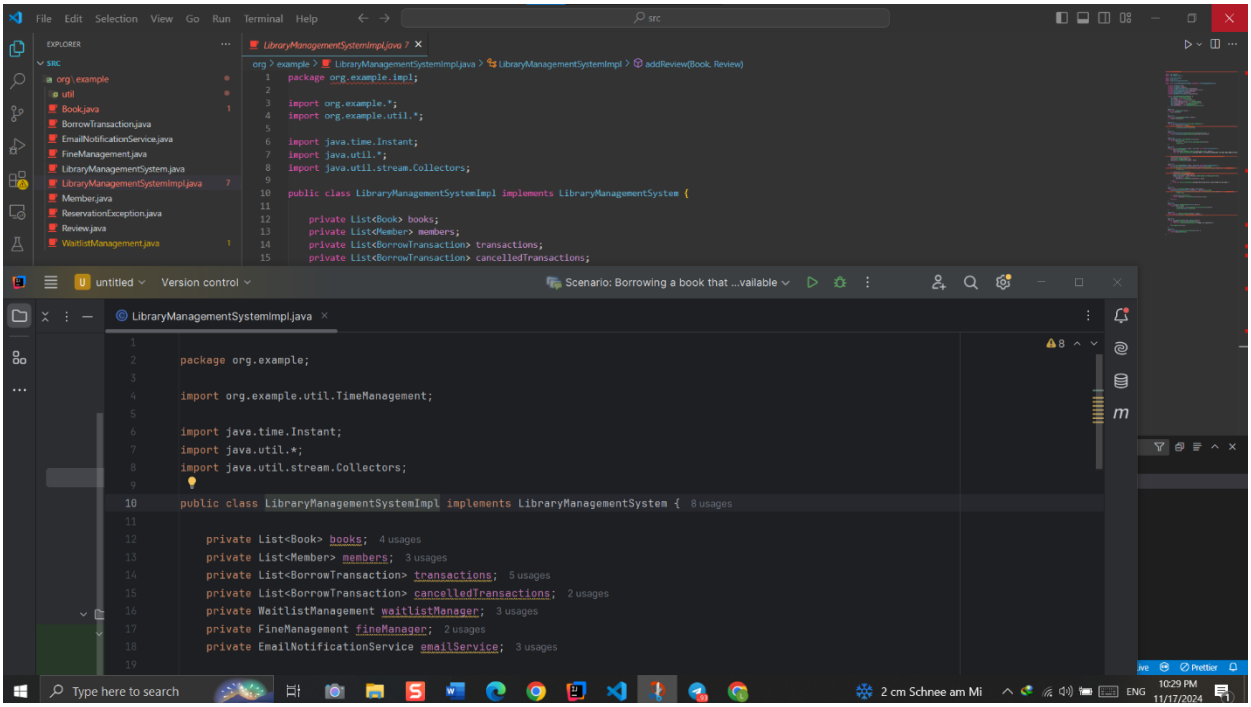
Fix mpl






LibraryManagementSystemImpl.java
public class LibraryManagementSystemImpl implements LibraryManagementSystem {
@Override
public void returnBook(Member member, Book book) throws ReservationException {
Optional<BorrowTransaction> transaction = transactions.stream()
.filter(t -> t.getBook().equals(book) && t.getMember().equals(member) && !t.isReturned())
.findFirst();
if (transaction.isPresent()) {
transaction.get().markReturned();
book.setBorrowed(false);
// Notify the next member on the waitlist, if any


LibraryManagementSystemImpl.java
public class LibraryManagementSystemImpl implements LibraryManagementSystem {
@Override
public List<Book> getBooksByGenre(String genre) {
return books.stream()
.filter(book -> book.getGenre().equalsIgnoreCase(genre))
.collect(Collectors.toList());
}
@Override
public boolean isMembershipExpired(Member member) { return !TimeManagement.isMembershipValid(member); }
@Override
public void addReview(Book book, Review review) {
if (!members.contains(review.getMember())) {
throw new IllegalArgumentException("Member not registered.");
}
book.addReview(review);



اولین پکیج به example.org تغییر کرد تا با ساختار دیگر فایل‌ها سازگاری داشته باشد.

تکلیف اول

بخش اول:

سناریو 1: افزودن کتاب جدید به سیستم

شرح سناریو:

بررسی عملکرد متد AddBook برای اضافه کردن یک کتاب جدید به سیستم.

پیش‌شرط‌ها:

1. سیستم باید آماده به کار باشد (بدون هیچ خطای اولیه).
2. لیست کتاب‌ها باید موجود و در دسترس باشد.
3. داده‌های کتاب باید کامل و معتبر باشند (شامل عنوان، ژانر).
4. در صورت نیاز به بررسی تکراری بودن، عناوین کتاب‌های فعلی باید به‌درستی ذخیره شده باشند.

خروجی مورد انتظار:

1. کتاب به لیست کتاب‌ها اضافه شود.
2. تعداد کتاب‌های موجود در سیستم افزایش یابد.
3. اطلاعات کتاب اضافه شده صحیح باشد. .

مقادیر تست:

1. کتاب:

- عنوان: "تفکر سریع و کند"
- ژانر: "روانشناسی"

سناریو 2: ثبت یک عضو جدید در سیستم

شرح سناریو:

بررسی عملکرد متد RegisterMember برای ثبت اطلاعات یک عضو جدید در سیستم.

پیش‌شرط‌ها:

1. سیستم باید آماده به کار باشد (بدون هیچ خطای اولیه).
2. لیست اعضای ثبت‌شده باید موجود و در دسترس باشد.
3. اطلاعات عضو جدید باید کامل و معتبر باشند (شامل نام، نقش).
4. عضویت عضو جدید در لحظه ثبت معتبر باشد (تاریخ انقضا در آینده باشد).

خروجی مورد انتظار:

1. عضو جدید با موفقیت به سیستم اضافه شود.
2. تعداد اعضا در سیستم افزایش یابد.
3. عضویت جدید معتبر باشد.

مقادیر تست:

1. عضو:

- شماره عضویت: 1111
- نام: "امین کیانی"
- نقش: "aminkiani82@gmail.com"

سناریو 3: جستجوی کتاب در سیستم بر اساس عنوان

شرح سناریو:

بررسی عملکرد متد SearchBook برای پیدا کردن یک کتاب بر اساس عنوان وارد شده.

پیش‌شرط‌ها:

1. حداقل یک کتاب با عنوان مشخص باید از قبل در سیستم ثبت شده باشد.
2. عنوان کتاب‌های ثبت‌شده باید بدون خطا و به‌درستی ذخیره شده باشند.
3. سیستم باید قابلیت جستجوی کتاب‌ها را داشته باشد.

خروجی مورد انتظار:

1. اگر کتاب با عنوان وارد شده موجود باشد، باید به‌درستی بازگردانده شود.

مقادیر تست:

1. عنوان کتاب: "Test Book"

سناریو 4: امانت گرفتن کتاب وقتی کتاب در دسترس است.

شرح سناریو:

بررسی عملکرد متد BorrowBook برای امانت گرفتن کتاب وقتی کتاب مورد نظر در دسترس است.

پیش‌شرط‌ها:

1. کتاب باید در سیستم ثبت شده و وضعیت آن "در دسترس" باشد.
2. عضو درخواست‌دهنده باید قبلاً در سیستم ثبت شده باشد.
3. تاریخ امانت باید به‌درستی ذخیره شود.

خروجی مورد انتظار:

1. وضعیت کتاب به "امانت گرفته‌شده" تغییر کند.
2. تراکنش جدید در لیست امانت‌ها ثبت شود.
3. تعداد امانت‌ها به‌درستی افزایش یابد. .

مقادیر تست:

1. عضو: testMember
2. کتاب: testBook

سناریو 5: تلاش برای امانت گرفتن کتاب وقتی کتاب در دسترس نیست.

شرح سناریو:

بررسی عملکرد متد BorrowBook وقتی تلاش می‌شود کتابی که قبلاً امانت گرفته‌شده است، دوباره امانت گرفته شود.

پیش‌شرط‌ها:

1. کتاب باید از قبل به‌عنوان "امانت گرفته‌شده" علامت‌گذاری شده باشد.
2. عضو درخواست‌دهنده باید قبلاً در سیستم ثبت شده باشد.

خروجی مورد انتظار:

1. سیستم باید از امانت دوباره کتاب جلوگیری کند.
2. استثنای ReservationException پرتاب شود.

مقادیر تست:

1. کتاب: testBook (در وضعیت "امانت گرفته‌شده")
2. عضو: testMember

بخش دوم:

هدف آزمون پذیرش به سبک Gherkin: بررسی عملکرد کلی سیستم از دیدگاه کاربر.

- **پوشش کامل :**سناریوهای آزمون باید تمام جنبه‌های سیستم را پوشش دهند.
- **سادگی :**سناریوها باید به زبان ساده و قابل فهم نوشته شوند.
- **مستقل :**هر سناریو باید به صورت مستقل قابل اجرا باشد.
- **قابل تکرار :**سناریوها باید به صورت خودکار قابل اجرا و تکرار باشند.

اجرای آزمون:

- **JUnit:** برای نوشتن آزمون‌های واحد در جاوا
- **Cucumber:** برای اجرای سناریوهای آزمون بر اساس Gherkin

سناریو 1: افزودن یک کتاب جدید به کتابخانه

Gherkin

```
Feature: مدیریت کتاب‌ها
    Scenario: افزودن یک کتاب جدید به کتابخانه
        Given یک سیستم کتابخانه که لیستی از کتاب‌ها ندارد.
```

```
        When  کتابدار کتابی با عنوان "1984" و ژانر "دیستوپیایی" را اضافه
                                                                می‌کند.
        Then  سیستم باید یک کتاب با عنوان "1984" و ژانر "دیستوپیایی" داشته
                                                                باشد.
        And .کتاب باید برای امانت گرفتن در دسترس باشد
```

سناریو 2: امانت گرفتن کتابی که در دسترس است

Gherkin

```
Feature:  امانت گرفتن کتاب‌ها
    Scenario: .امانت گرفتن کتابی که در دسترس است
        Given  یک سیستم کتابخانه با کتابی با عنوان "کشتن مرغ مقلد" و ژانر
                                                                "داستانی"
        And "123" یک عضو ثبت‌شده با شناسه
        When  عضوی با شناسه "123" کتابی با عنوان "کشتن مرغ مقلد" را امانت
                                                                می‌گیرد.

        Then  سیستم باید کتاب "کشتن مرغ مقلد" را به وضعیت "امانت گرفته‌شده"
                                                                علامت بزند.
```

سناریو 3: ثبت‌نام یک عضو جدید

Gherkin

```
Feature:  مدیریت عضویت
    Scenario:  ثبت‌نام یک عضو جدید
        Given  یک سیستم کتابخانه که هیچ عضوی ثبت‌نام نکرده است
        When "aminkiani82@gmail.com" یک کاربر با نام "امین کیانی" و ایمیل
                                                                ثبت‌نام می‌کند.
        Then .سیستم باید یک عضو ثبت‌شده با نام "امین کیانی" داشته باشد

        And .عضویت عضو نباید منقضی شده باشد
```

تکلیف دوم

بخش اول:

JUnit 5

```java
package org.example;
import org.junit.jupiter.api.*;

import java.time.Instant;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
```

```
class LibraryManagementSystemImplTest {

    private LibraryManagementSystemImpl library;
    private Member testMember;
    private Book testBook;

    /**
     * می‌شود اجرا تست‌ها همه از قبل یکبار فقط که سیستم اولیه تنظیم.
     * می‌شود استفاده سنگین و کلی تنظیمات انجام برای متد این.
     */
    @BeforeAll
    static void beforeAll() {
        System.out.println("Initializing resources before all tests...");
        // انجام اینجا باشد، محیط یا سیستم برای دیگری تنظیمات به نیاز اگر
می‌شود.
    }
    /**
     * تست هر اجرای از قبل داده‌ها اولیه تنظیم.
     * می‌کند اضافه سیستم به آزمایشی کتاب یک و عضو یک متد این.
     */
    @BeforeEach
    void setUp() {
        library = new LibraryManagementSystemImpl();
        testMember = new Member(111,"amin kiani", "aminkiani82@gmail.com");
        testBook = new Book("Test Book", "Fiction");

        library.registerMember(testMember);
        library.addBook(testBook);
    }

    /**
     * می‌شود انجام تست هر از پس که پاکسازی عملیات.
     * می‌شود استفاده تست هر با مرتبط منابع آزادسازی برای.
     */
    @AfterEach
    void tearDown() {
        System.out.println("Executing AfterEach: Cleaning up after test...");
        library = null; // مرجع آزادسازی
        testMember = null;
        testBook = null;
    }

    /**
     * می‌شود انجام تست‌ها تمام اتمام از پس یکبار فقط که پاکسازی عملیات.
     * می‌شود استفاده سیستم وضعیت مجدد تنظیم یا کلی منابع آزادسازی برای.
     */
    @AfterAll
    static void afterAll() {
        System.out.println("Executing AfterAll: Cleaning up global
resources...");
        // کلی منابع پاکسازی یا دیتابیس به اتصال بستن :مثال
    }

    /**
     * سیستم به جدید کتاب کردن اضافه تست.
     * شود اضافه سیستم کتاب‌های لیست به جدید کتاب که می‌کند بررسی.
     */
```

```
    @Test
    void testAddBook() {
        Book newBook = new Book("New Book", "Science");
        library.addBook(newBook);
        List<Book> books = library.getBooksByGenre("Science");
        assertEquals(1, books.size()); // مشخص ژانر کتاب‌های تعداد بررسی
        assertEquals("New Book", books.get(0).getTitle()); // عنوان بررسی
کتاب
    }

    /**
     * سیستم در جدید عضو یک ثبت تست.
     * شود اضافه سیستم به جدید عضو که می‌کند بررسی.
     */
    @Test
    void testRegisterMember() {
        Member newMember = new Member(1111,"amin kiani",
"aminkiani82@gmail.com");
        library.registerMember(newMember);  //
        assertTrue(library.isMembershipExpired(newMember)); // است این بر فرض
که است معتبر جدید عضویت .
    }

    /**
     * عنوان اساس بر سیستم در کتاب جستجوی تست.
     * شود پیدا نظر مورد کتاب که می‌کند بررسی.
     */
    @Test
    void testSearchBook() {
        Optional<Book> foundBook = library.searchBook("Test Book");
        assertTrue(foundBook.isPresent()); // است شده پیدا کتاب اینکه بررسی
        assertEquals(testBook, foundBook.get()); // شده پیدا کتاب تطابق بررسی
    }

    /**
     * است دسترس در کتاب وقتی کتاب گرفتن امانت تست.
     * کند تغییر "گرفته‌شده امانت" به کتاب وضعیت که می‌کند بررسی.
     */
    @Test
    void testBorrowBookSuccessfully() throws ReservationException {
        library.borrowBook(testMember, testBook);
        assertTrue(testBook.isBorrowed()); // کتاب وضعیت بررسی
        assertEquals(1, library.getAllBorrowedBooks().size()); // تعداد بررسی
امانت‌ها
    }

    /**
     * نیست دسترس در کتاب وقتی کتاب گرفتن امانت برای تلاش تست.
     * شود پرتاب ReservationException استثنای که می‌کند بررسی.
     */
    @Test
    void testBorrowBookWhenUnavailable() {
        testBook.setBorrowed(true); // به کتاب وضعیت تغییر "گرفته‌شده امانت"
        assertThrows(ReservationException.class, () ->
library.borrowBook(testMember, testBook));
    }
```

```
    /**
     * کتابخانه به کتاب بازگرداندن تست.
     * کند تغییر "دسترس در" به کتاب وضعیت که می‌کند بررسی.
     */
    @Test
    void testReturnBookSuccessfully() throws ReservationException {
        library.borrowBook(testMember, testBook); // کتاب گرفتن امانت
        library.returnBook(testMember, testBook); // کتاب بازگرداندن
        assertFalse(testBook.isBorrowed()); // کتاب وضعیت بررسی
        assertTrue(library.getAllBorrowedBooks().isEmpty()); // اینکه بررسی
است خالی امانت‌ها لیست
    }

    /**
     * است نشده گرفته امانت که کتابی بازگرداندن برای تلاش تست.
     * شود پرتاب ReservationException استثنای که می‌کند بررسی.
     */
    @Test
    void testReturnBookWithoutBorrowing() {
        assertThrows(ReservationException.class, () ->
library.returnBook(testMember, testBook));
    }

//    @Test
//    void testAddReview() {
//        Review review = new Review(testMember,5 , "Great book!");
//        library.addReview(testBook, review);
//        assertEquals(1, testBook.getReviews().size());
//        assertEquals("Great book!", library.getReviewsForBook(testBook));
//    }

    /**
     * ندارد وجود جریمه‌ای وقتی جریمه محاسبه تست.
     * تأخیر دلیل به نباشد صفر جریمه مقدار که می‌کند بررسی.
     */
    @Test
    void testCalculateFineWithDelay() throws ReservationException {
        library.borrowBook(testMember, testBook); // کتاب گرفتن امانت

        // تأخیر کردن اضافه
        try {
            Thread.sleep(2000); // 2 تأخیر ثانیه
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            fail("Test interrupted unexpectedly");
        }

        double fine = library.calculateFine(testMember, testBook);
        assertEquals(0, fine, "you fined!"); // باشد صفر نباید جریمه مقدار.
    }


    /**
     * مشخص ژانر اساس بر کتاب‌ها لیست دریافت تست.
     * شوند بازگردانده لیست در صحیح کتاب‌های که می‌کند بررسی.
     */
    @Test
```

```
    void testGetBooksByGenre() {
        Book secondBook = new Book("Another Book", "Fiction");
        library.addBook(secondBook);
        List<Book> books = library.getBooksByGenre("Fiction");
        assertEquals(2, books.size()); // مشخص ژانر کتاب‌های تعداد بررسی
    }

    /**
     * کتاب یک برای نظرات دریافت و کردن اضافه تست.
     * شوند بازگردانده و اضافه کتاب به درستی به نظرات که می‌کند بررسی.
     */
//     @Test
//     void testAddAndGetReviews() {
//         Review review = new Review(testMember, "Great book!", 5);
//         library.addReview(testBook, review); // نظر کردن اضافه
//         List<Review> reviews = library.getReviewsForBook(testBook);
//         assertEquals(1, reviews.size()); // نظرات تعداد بررسی
//         assertEquals("Great book!", reviews.get(0).getComment()); // بررسی
نظر محتوای
//     }
    /**
     * عضو یک عضویت انقضای وضعیت بررسی تست.
     * دهد تشخیص را عضویت انقضای درستی به سیستم که می‌کند بررسی.
     */
//     @Test
//     void testMembershipExpiration() {
//         // است معتبر پیش‌فرض به‌صورت تست عضویت می‌کنیم فرض
//         assertFalse(library.isMembershipExpired(testMember)); // بررسی
معتبر عضویت
//
//         // است شده منقضی عضویت می‌کنیم فرض حالا
//
testMember.setMembershipExpirationDate(Instant.now().minusSeconds(60 * 60 *
24)); // دیروز : انقضا تاریخ
//         assertTrue(library.isMembershipExpired(testMember)); // عضویت بررسی
شده منقضی
//     }
    @Test
    void testIsMembershipExpired() {
        Member newMember = new Member(1,"Expired Member",
"expired@example.com");
        library.registerMember(newMember);
        assertFalse(library.isMembershipExpired(newMember));

        newMember.setMembershipExpirationDate(Instant.now().minusSeconds(60 *
60 * 24)); // one day : Set expiration date to past
        assertTrue(library.isMembershipExpired(newMember));  // after one day
must expire but still here
    }
}
```

بخش دوم:

Feature: Book Management #Step definition 1 - Cucumber

Scenario: Adding a new book to the library
  Given a library system with no books
  When the librarian adds a book titled "1984" in the genre "Dystopian"
  Then the system should have a book titled "1984" in the genre "Dystopian"
  And the book should be available for borrowing

```
package org.example;

import io.cucumber.java.en.*;
import org.example.Book;
import org.example.LibraryManagementSystemImpl;

import static org.junit.jupiter.api.Assertions.*;

public class LibraryStep1 {

    private LibraryManagementSystemImpl librarySystem;
    private Book addedBook;

    @Given("a library system with no books")
    public void a_library_system_with_no_books() {
        librarySystem = new LibraryManagementSystemImpl();
        assertTrue(librarySystem.getBooksByGenre("Fiction").isEmpty(), "The
library should initially have no books.");
    }

    @When("the librarian adds a book titled {string} in the genre {string}")
    public void the_librarian_adds_a_book_titled_in_the_genre(String title,
String genre) {
        addedBook = new Book(title, genre);
        librarySystem.addBook(addedBook);
    }

    @Then("the system should have a book titled {string} in the genre
{string}")
    public void the_system_should_have_a_book_titled_in_the_genre(String
title, String genre) {
        var books = librarySystem.getBooksByGenre(genre);
        assertEquals(1, books.size(), "There should be exactly one book in
the genre.");
        assertEquals(title, books.get(0).getTitle(), "The title of the book
should match.");
    }

    @Then("the book should be available for borrowing")
    public void the_book_should_be_available_for_borrowing() {
        assertFalse(addedBook.isBorrowed(), "The book should be available for
borrowing.");
    }
}
```

Feature: Borrowing Books #Step definition 2 - Cucumber

Scenario: Borrowing a book that is available

Given a library system with a book titled "To Kill a Mockingbird" in the genre "Fiction"

And a registered member with ID "123"

When the member with ID "123" borrows the book titled "To Kill a Mockingbird"

Then the system should mark the book titled "To Kill a Mockingbird" as borrowed

```
package org.example;

import io.cucumber.java.en.*;
import org.example.LibraryManagementSystemImpl;
import org.example.Book;
import org.example.Member;
import org.example.ReservationException;

import static org.junit.jupiter.api.Assertions.*;

public class LibraryStep2 {

    private LibraryManagementSystemImpl librarySystem;
    private Book book;
    private Member member;

    @Given("a library system with a book titled {string} in the genre
{string}")
    public void a_library_system_with_a_book_titled_in_the_genre(String
title, String genre) {
        librarySystem = new LibraryManagementSystemImpl();
        book = new Book(title, genre);
        librarySystem.addBook(book); // می‌کنیم اضافه کتابخانه به را کتاب
    }

    @Given("a registered member with ID {string}")
    public void a_registered_member_with_id(String memberId) {
        member = new Member(Integer.parseInt(memberId), "amin kiani",
"aminkiani82@gmail.com");
        librarySystem.registerMember(member); // می‌کنیم ثبت را عضو
    }

    @When("the member with ID {string} borrows the book titled {string}")
    public void the_member_borrows_the_book(String memberId, String title)
throws ReservationException {
        // می‌گیرد امانت را کتاب مشخص‌شده عضو
        librarySystem.borrowBook(member, book);
    }

    @Then("the system should mark the book titled {string} as borrowed")
    public void the_system_should_mark_the_book_as_borrowed(String title) {
        assertTrue(book.isBorrowed(), "The book should be marked as
borrowed.");
    }

}
```