Movie theater

Theo Gabrial

420-SF2-RE DATA STRUCTURES AND OBJECT ORIENTED PROGRAMMING

Description

This project simulates a movie theater booking system where customers can book seats for regular or vip movies. The application handles seat reservations, pricing and maintains booking records.

Hierarchy User Class

Customer class is used to get and print the name of the user

```java
public class Customer extends User {  1 usage

    public Customer(String name) {  1 usage
        super(name);
    }

    /**
     * prints customer
     */
    @Override  1 usage
    public void showUserType() { System.out.println("Customer: " + name); }
}
```

Hierarchy Movie class

RegularMovie Class is used to get the price of a regular movie

```
public class RegularMovie extends Movie {  5 usages

    public RegularMovie(String title, int duration) { super(title, duration); }

    @Override  1 usage
    public double getPrice() { return 13.0; }
}
```

VipMovie Class is used to get the price of a VIP movie

```
public class VipMovie extends Movie {  2 usages
    public VipMovie(String title, int duration) { super(title, duration); }

    @Override  1 usage
    public double getPrice() { return 25.0; }
}
```

Bookable interface

```
public interface Bookable {  1 usage  3 implementations
    void bookSeat(int seatNumber);  3 usages  1 implementation
}
```

Runtime-Polymorphism

```
public abstract double getPrice();  1 usage  2 implementations
```

Regular Movie

```
@Override  1 usage
public double getPrice() { return 13.0; }
```

Vip Movie

```
    @Override  1 usage
    public double getPrice() { return 25.0; }
}
```

Test Movie

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

public class MovieTest {  no usages

    @Test  no usages
    public void testBookingSeat() {
        Movie movie = new RegularMovie( title: "Test Movie",  duration: 120);
        movie.bookSeat( seatNumber: 5);
        assertTrue(movie.getBookedSeats().contains(5));
    }

    @Test  no usages
    public void testPrice() {
        Movie imax = new VipMovie( title: "VIP Movie",  duration: 140);
        assertEquals(25.0, imax.getPrice(), 0.01);
    }

    @Test  no usages
    public void testComparable() {
        Movie m1 = new RegularMovie( title: "Short",  duration: 90);
        Movie m2 = new RegularMovie( title: "Long",  duration: 150);
        assertTrue(m1.compareTo(m2) < 0);
    }
}
```

TextIO

```java
/**
 * saves booked seats to file
 * @param movies movies
 * @param filename file
 */
public static void saveBookings(List<Movie> movies, String filename) {  1 usage
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
        for (Movie movie : movies) {
            for (int seat : movie.getBookedSeats()) {
                writer.write( str: movie.getTitle() + "," + seat);
                writer.newLine();
            }
        }
    } catch (IOException e) {
        System.err.println("Error writing file: " + e.getMessage());
    }
}

/**
 * puts moves in file into a stringList
 * @param filename file
 * @return stringList of movies
 */
public static List<String> loadBookings(String filename) {  1 usage
    List<String> bookings = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
        String line;
        while ((line = reader.readLine()) != null) {
            bookings.add(line);
        }
    } catch (IOException e) {
        System.err.println("Error reading file: " + e.getMessage());
    }
    return bookings;
```

Comparable Movie Class

```
    @Override
    public int compareTo(Movie other) { return Integer.compare(this.duration, other.duration); }
}
```

Comparator

```
public class MovieTitleComparator implements Comparator<Movie> {  1 usage

    @Override
    public int compare(Movie m1, Movie m2) { return m1.getTitle().compareTo(m2.getTitle()); }
}
```

Challenges and Learning

The biggest challenge was the transition between Deliverable 2 and Deliverable 3. I learn by trial and error so when I realized that my plan for deliverable 2 wouldn't work whatsoever, it was difficult for me to start completely from scratch. I learned to take it one step at a time. Instead of trying to come up with something that fills all the requirements I focus on one requirement at a time.

Another more specific challenge for me was figuring out how to do the testing and the textIO. Those were the two things that made the least sense to me. I used my previous tests, the documents on lea and the internet to figure out how to make it work. Even though I knew this already it reassured me that I can get anything done thanks to the resources that I've been provided with.