Object Oriented Programming and Design (Assignment 2)

Khushi Suri (202217b3187)

Section H

Problem Statement 1:

Journal registration and validation using Exception handling in Java - 5 marks

Solution:

```
import java.util.ArrayList;
import java.util.List;
class CHECK_NAME extends Exception {
   public CHECK_NAME(String m) {
      super(m);
   }
}
class CHECK_JOURNAL_ID extends Exception {
   public CHECK_JOURNAL_ID(String m) {
      super(m);
   }
}
class ISSUENUMBER extends Exception {
   public ISSUENUMBER(String m) {
      super(m);
   }
}
class CHECK_ISSN extends Exception {
   public CHECK_ISSN(String m) {
      super(m);
   }
}
class Journal {
   private String name;
```

```
    private String journalID;
    private String issueNumber;
    private String issn;
    public Journal(String name, String journalID, String issueNumber, String issn) {
        this.name = name;
        this.journalID = journalID;
        this.issueNumber = issueNumber;
        this.issn = issn;
    }
    @Override
    public String toString() {
        return "Journal [Name=" + name + ", JournalID=" + journalID +
                ", IssueNumber=" + issueNumber + ", ISSN=" + issn + "]";
    }
}
class p1 {
    public static void validateName(String nm) throws CHECK_NAME {
        if (nm.length() > 30) {
            throw new CHECK_NAME("Invalid Name: Must be <= 30 characters.");
        }
        for (char ch : nm.toCharArray()) {
            if (!Character.isLetter(ch))
                throw new CHECK_NAME("Invalid Name: Must contain alphabets only.");
        }
    }
    public static void validateJournalID(String id) throws CHECK_JOURNAL_ID {
        for (char ch : id.toCharArray()) {
            if (!Character.isLetterOrDigit(ch))
                throw new CHECK_JOURNAL_ID("Invalid Journal ID: Must be alphanumeric.");
        }
    }
```

```
public static void validateIssueNumber(String in) throws ISSUENUMBER {
    if (in.length() > 20)
        throw new ISSUENUMBER("Invalid Issue Number: Must be <= 20 characters.");
}
public static void validateISSN(String issn) throws CHECK_ISSN {
    if (issn.length() != 9) {
        throw new CHECK_ISSN("Invalid ISSN: Must be exactly 9 characters including one hyphen.");
    }
    int hyphenCount = 0;
    for (char ch : issn.toCharArray()) {
        if (ch == '-') {
            hyphenCount++;
        } else if (!Character.isDigit(ch)) {
            throw new CHECK_ISSN("Invalid ISSN: Must contain only digits and one hyphen.");
        }
    }
    if (hyphenCount != 1) {
        throw new CHECK_ISSN("Invalid ISSN: Must contain exactly one hyphen.");
    }
}
public static void main(String[] args) {
    List<Journal> journals = new ArrayList<>();
    String[][] journalDetails = {
            { "JournalOne", "J12345", "001", "1234-56789" }, // Invalid ISSN
            { "AnotherJournal", "AJ7890", "002", "9876-5432" },
            { "InvalidName123", "ID001", "003", "1111-2222" }, // Invalid Name
            { "ValidJournal", "ID@002", "004", "3333-4444" }, // Invalid ID
            { "LongJournalNameWhichExceedsLimit", "ID003", "005", "5555-6666" } // Invalid Name
    };
    for (String[] details : journalDetails) {
        try {
```

```
            validateName(details[0]);
            validateJournalID(details[1]);
            validateIssueNumber(details[2]);
            validateISSN(details[3]);
            journals.add(new Journal(details[0], details[1], details[2], details[3]));
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
    System.out.println("\nValid Journals:");
    for (Journal journal : journals) {
        System.out.println(journal);
    }
  }
}
```

Output Screenshot:

Problem Statement 2:

Passenger registration using Collections, Comparable, Comparator interfaces – 5 marks

Solution:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
class Passenger implements Comparable<Passenger> {
   String name, ticketClass, startPlace, destination;
   int id, age, seatNumber;
   public Passenger(String name, int id, int age, String ticketClass, String startPlace, String
destination,
         int seatNumber) {
      this.name = name;
      this.id = id;
      this.age = age;
      this.ticketClass = ticketClass;
      this.startPlace = startPlace;
      this.destination = destination;
      this.seatNumber = seatNumber;
   }
   @Override
   public int compareTo(Passenger ob) {
      return this.name.compareTo(ob.name);
   }
   public void displayPassengerData() {
      System.out.printf("%-10s %-10d %-5d %-10s %-15s %-15s %-5d\n", name, id, age, ticketClass,
startPlace,
```

```
            destination, seatNumber);
    }
}
class SortByID implements Comparator<Passenger> {
    public int compare(Passenger p1, Passenger p2) {
        return Integer.compare(p1.id, p2.id);
    }
}
class SortByAge implements Comparator<Passenger> {
    public int compare(Passenger p1, Passenger p2) {
        return Integer.compare(p1.age, p2.age);
    }
}
class SortByTicketClass implements Comparator<Passenger> {
    public int compare(Passenger p1, Passenger p2) {
        return p1.ticketClass.compareTo(p2.ticketClass);
    }
}
class SortByDestination implements Comparator<Passenger> {
    public int compare(Passenger p1, Passenger p2) {
        return p1.destination.compareTo(p2.destination);
    }
}
public class p2 {
    public static void main(String[] args) {
        Scanner SC = new Scanner(System.in);
        List<Passenger> passengers = new ArrayList<>();
        HashSet<Integer> uniqueIDs = new HashSet<>();
        for (int i = 0; i < 5; i++) {
            System.out.println("Enter details for Passenger " + (i + 1) + ":");
            System.out.print("Name: ");
```

```
            String name = SC.next();
            System.out.print("Passenger ID: ");
            int id = SC.nextInt();
            // Ensure id is unique
            while (!uniqueIDs.add(id)) {
                System.out.print("Duplicate ID! Enter a unique Passenger ID: ");
                id = SC.nextInt();
            }
            System.out.print("Age: ");
            int age = SC.nextInt();
            System.out.print("Ticket Class: ");
            String ticketClass = SC.next();
            System.out.print("Start Place: ");
            String startPlace = SC.next();
            System.out.print("Destination: ");
            String destination = SC.next();
            System.out.print("Seat Number: ");
            int seatNumber = SC.nextInt();
            passengers.add(new Passenger(name, id, age, ticketClass, startPlace, destination,
seatNumber));
        }
        // Sorting menu
        System.out.println("\nChoose attribute to sort by:");
        System.out.println("1. Name\n2. Passenger ID\n3. Age\n4. Ticket Class\n5. Destination");
        int choice = SC.nextInt();
        switch (choice) {
            case 1:
                Collections.sort(passengers);
                break;
            case 2:
                passengers.sort(new SortByID());
```

```
            break;
        case 3:
            passengers.sort(new SortByAge());
            break;
        case 4:
            passengers.sort(new SortByTicketClass());
            break;
        case 5:
            passengers.sort(new SortByDestination());
            break;
        default:
            System.out.println("Invalid choice!");
            SC.close();
            return;
      }
      // Display sorted data
      System.out.println("\nSorted Passenger Details:");
      System.out.printf("%-10s %-10s %-5s %-10s %-15s %-15s %-5s\n", "Name", "ID", "Age", "Class",
"Start Place",
            "Destination", "Seat");
      for (Passenger p : passengers) {
        p.displayPassengerData();
      }
      SC.close();
   }
}
```

Output Screenshot: