BSCCS2005: OPE2 Questions
with Test Cases and Solutions

Overview

The exam will be conducted in two sessions. In each session, a student will be presented with 6 questions out of which 4 are graded and 2 are optional (challenging). Out of the 4 graded questions, he/she should answer 3 questions correctly, passing all the private test cases to get a full score. In case they attempt all 4, only the best 3 scores will be considered. We configure 2 copies of each type on the portal, and using a randomization script, every student sees one copy of each type on the exam portal.
The challenging questions are not graded. The purpose of these questions is to engage students who finish the other questions early.

0.1 Exceptions

Problem Statement

Write a Java program that allows the user to create projects, add team members to each project, and ensure that every project has at least one team member. If the project doesn't have any member then add default team member. The program takes project name, number of team members followed by team members names of some projects as input and finally prints the project list with their team members. Complete the program as specified below.

- Class `NoTeamMemberException` extends the `Exception` class and should have the following member:
  - Constructor that call its super class constructor
- Class `Project` has/should have the following members:
  - Private instance variables `String projectName` and `List<String> teamMembers`
  - A constructor to initialize the instance variables
  - Method `addTeamMember` that adds a team member to the project
  - Method `toString` to print in the format shown in the test cases
  - Method `hasTeamMembers` that returns `true` if the project has team members, and `NoTeamMemberException` otherwise
- Class `ProjectManager` has the following members:
  - Method `updateProjectList` that takes a `Project` list as a parameter and checks if each project has at least one team member. If not, it catches the `NoTeamMemberException`, adds a default team member to the project
  - Method `displayProjectList` that takes a `Project` list as a parameter and prints the updated project list
  - Method `main` takes input to create `Project` list and invokes the `updateProjectList` method to ensure that each project has at least one team member and also invokes the `displayProjectList` method to print the updated project list.

What you have to do

- Define class `NoTeamMemberException`
- Define method `updateProjectList` in class `ProjectManager`

Template Code

```
import java.util.*;
    // define class NoTeamMemberException
```

```
class Project {
    private String projectName;
    private List<String> teamMembers;
    public Project(String pN) {
        projectName = pN;
        this.teamMembers = new ArrayList<>();
    }
    public void addTeamMember(String memberName) {
        teamMembers.add(memberName);
    }
    public String toString() {
        return "Project: " + projectName + ", Team Members: " + teamMembers;
    }
    public boolean hasTeamMembers() throws NoTeamMemberException {
        if (teamMembers.size() > 0) {
          return true;
        } else {
          throw new NoTeamMemberException();
        }
    }
}
public class ProjectManager {

       // define method updateProjectList

    public static void displayProjectList(List<Project> projectList) {
        System.out.println("Updated Project List:");
        for (Project project : projectList) {
            System.out.println(project);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Project> projectList = new ArrayList<Project>();
        for (int i = 0; i < 3; i++) {
            String projectName = sc.next();
            Project project = new Project(projectName);
            projectList.add(project);
            int numTeamMembers = sc.nextInt();
            for (int j = 0; j < numTeamMembers; j++) {
                String memberName = sc.next();
                project.addTeamMember(memberName);
            }
        }
        try {
```

```
            for (Project project : projectList){
               project.hasTeamMembers();
            }
        } catch (NoTeamMemberException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
        updateProjectList(projectList);
        displayProjectList(projectList);
        sc.close();
    }
}
```

Public test case 1:
Input:

```
ChatGPT-Enhancement 3 Alice Bob Charlie
OpenAI-Research 2 David Eva
ProjectX 1 Ian
```

Output:

```
Updated Project List:
Project: ChatGPT-Enhancement, Team Members: [Alice, Bob, Charlie]
Project: OpenAI-Research, Team Members: [David, Eva]
Project: ProjectX, Team Members: [Ian]
```

Public test case 2:
Input:

```
Website-Redesign 1 Elena
Machine-Learning-Project 0
Data-Analysis 0
```

Output:

```
Exception caught: null
Updated Project List:
Project: Website-Redesign, Team Members: [Elena]
Project: Machine-Learning-Project, Team Members: [Default Member]
Project: Data-Analysis, Team Members: [Default Member]
```

Private test case 1:
Input:

```
AI-Chatbot-Integration 2 Grace Emma
Project-Management-Tool 0
Customer-Survey-Analysis 1 Ian
```

Output:

```
Exception caught: null
Updated Project List:
Project: AI-Chatbot-Integration, Team Members: [Grace, Emma]
Project: Project-Management-Tool, Team Members: [Default Member]
Project: Customer-Survey-Analysis, Team Members: [Ian]
```

Solution:

```
import java.util.*;
class NoTeamMemberException extends Exception {
    public NoTeamMemberException() {
        super();
    }
}
class Project {
    private String projectName;
    private List<String> teamMembers;
    public Project(String pN) {
        this.projectName = pN;
        this.teamMembers = new ArrayList<>();
    }
    public void addTeamMember(String memberName) {
        teamMembers.add(memberName);
    }
    public String toString() {
        return "Project: " +projectName + ", Team Members: " +teamMembers;
    }
    public boolean hasTeamMembers() throws NoTeamMemberException {
        if (teamMembers.size() > 0) {
          return true;
        } else {
          throw new NoTeamMemberException();
        }
    }
}
public class ProjectManager {
    public static void updateProjectList(List<Project> projectList) {
        for (Project project : projectList) {
          try {
            project.hasTeamMembers();
          } catch (NoTeamMemberException e) {
```

```
            project.addTeamMember("Default Member");
          }
        }
    }
    public static void displayProjectList(List<Project> projectList) {
        System.out.println("Updated Project List:");
        for (Project project : projectList) {
            System.out.println(project);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Project> projectList = new ArrayList<Project>();
        for (int i = 0; i < 3; i++) {
            String projectName = sc.next();
            Project project = new Project(projectName);
            projectList.add(project);
            int numTeamMembers = sc.nextInt();
            for (int j = 0; j < numTeamMembers; j++) {
                String memberName = sc.next();
                project.addTeamMember(memberName);
            }
        }
        try {
            for (Project project : projectList){
               project.hasTeamMembers();
            }
        } catch (NoTeamMemberException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
        updateProjectList(projectList);
        displayProjectList(projectList);
        sc.close();
    }
}
```

0.2 Session 1 Type 1 Copy Streams

Problem Statement

Write a Java program that, given as input name, price and stock quantity of some products, prints the filtered stream of products that have price greater than 5000.00 and stock quantity more than 10. Complete the program as specified below.

- Class `Product` has/should have the following members:
  - Private instance variables `String name`, `double price` and `int stockQuantity`
  - A constructor to initialize instance variables
  - Method `toString` to print in the format shown in the test cases
  - Accessor method for `price` and `stockQuantity`
  - Method `productProcessor` should take an `ArrayList` of `Product` objects as input and returns a filtered stream of premium products with sufficient stock. The criteria for filtering are:
    - The price of the product is greater than 5000.00
    - The stock quantity of the product is greater than 10
- Class `StreamTest` has the following members:
  - Method `main` creates an `ArrayList` of `Product` objects by taking input in the order `name`, `price`, `stockQuantity` then invokes the method `productProcessor` to filter premium products with sufficient stock and then display those products

What you have to do

- Define method `productProcessor` in class `Product`

Template Code

```
import java.util.*;
import java.util.stream.*;
class Product {
    private String name;
    private double price;
    private int stockQuantity;
    public Product(String n, double p, int sq) {
        name = n;
        price = p;
        stockQuantity = sq;
    }
    public String toString() {
        return name + " - " + price + " - In Stock: " + stockQuantity;
    }
```

```
    public double getPrice() {
        return price;
    }
    public int getStockQuantity() {
        return stockQuantity;
    }
    // define method productProcessor

}
public class StreamTest {
    public static void main(String[] args) {
        ArrayList<Product> products = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < 4; i++) {
            Product product = new Product(sc.next(), sc.nextDouble(), sc.nextInt());
            products.add(product);
        }
        Stream<Product> filteredStream = Product.productProcessor(products);
        System.out.println("Premium Products with Sufficient Stock:");
        filteredStream.forEach(System.out::println);
        sc.close();
    }
}
```

Public test case 1:
Input:

```
Laptop 32000.0 15
Smartphone 8000.0 20
Smartwatch 300.0 12
TV 100000.0 8
```

Output:

```
Premium Products with Sufficient Stock:
Laptop - 32000.0 - In Stock: 15
Smartphone - 8000.0 - In Stock: 20
```

Public test case 2:
Input:

```
Blender 800.0 5
Headphones 600.0 8
Microwave 15000.0 20
Coffee_Maker 4000.0 15
```

Output:

```
Premium Products with Sufficient Stock:
Microwave - 15000.0 - In Stock: 20
```

Private test case 1:
Input:

```
Camera 6000.0 25
Smartphone 12000.0 25
Smartwatch 3000.0 15
AirConditioner 35000.0 20
```

Output:

```
Premium Products with Sufficient Stock:
Camera - 6000.0 - In Stock: 25
Smartphone - 12000.0 - In Stock: 25
AirConditioner - 35000.0 - In Stock: 20
```

Solution:

```
import java.util.*;
import java.util.stream.*;
class Product {
    private String name;
    private double price;
    private int stockQuantity;
    public Product(String n, double p, int sq) {
        name = n;
        price = p;
        stockQuantity = sq;
    }
    public String toString() {
        return name + " - " + price + " - In Stock: " + stockQuantity;
    }
    public double getPrice() {
        return price;
    }
    public int getStockQuantity() {
        return stockQuantity;
    }
    public static Stream<Product> productProcessor
                        (ArrayList<Product> products) {

        return products.stream()
```

```
                .filter(product -> product.getPrice() > 5000.00
                && product.getStockQuantity() > 10);
    }
}
public class StreamTest {
    public static void main(String[] args) {
        ArrayList<Product> products = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < 4; i++) {
            Product product = new Product(sc.next(),
                                sc.nextDouble(), sc.nextInt());
            products.add(product);
        }
        Stream<Product> filteredStream = Product.productProcessor(products);
        System.out.println("Premium Products with Sufficient Stock:");
        filteredStream.forEach(System.out::println);
        sc.close();
    }
}
```

0.3 Session 1 Type 2 Cloning

Problem Statement

Write a Java program that, given as input, details of an original book, represented by `Book b1`. The details include the book's `title`, author's `name`, and author's `age`. Subsequently, the program should clone the original book to create a new book, denoted as `Book b2`. The cloning process should ensure that modifications to the details of either `b1` or `b2` do not affect the other.Finally, the program displays the details of both the original and cloned books. Complete the program as specified below.

- Class `Author` implements `Cloneable` interface and has/should have the following members:
  - Private instance variables `String name` and `int age`
  - A constructor to initialize the instance variables
  - Accessor and Mutator methods for the `name` and `age`
  - Method `toString` to print in the format shown in the test cases
  - Implement the `clone()` method
- Class `Book` implements `Cloneable` interface and has/should have the following members:
  - Private instance variables `String title` and an instance of the `Author` class.
  - A constructor to initialize instance variables,taking the book's `title` and an `Author` object.
  - Method `toString` to print in the format shown in the test cases
  - Mutator method to set new `author`
  - Implement the `clone()` method
- Class `CloningTest` contains the `main` method that takes the inputs in the order of book's `title`, author's `name` and `age` and invokes appropriate methods to achieve the functionality

What you have to do

- Define method `clone` in class `Book`
- Define method `clone` in class `Author`

Template Code

```
class Author implements Cloneable {
    private String name;
    private int age;
```

```
    public Author(String n, int a) {
        name = n;
        age = a;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "Author: " + name + " (Age: " + age + ")";
    }
    // define method clone


}
class Book implements Cloneable {
    private String title;
    private Author author;
    public Book(String t, Author a) {
        title = t;
        author = a;
    }
    public void setAuthor(Author author) {
        this.author.setName(author.getName());
        this.author.setAge(author.getAge());
    }
    public String toString() {
        return "Book: " + title + "\n" + author;
    }
    // define method clone

}
public class CloningTest {
    public static void main(String[] args) throws CloneNotSupportedException {
        Scanner sc = new Scanner(System.in);
        Book b1 = new Book(sc.next(), new Author(sc.next(), sc.nextInt()));
        Book b2 = b1.clone();
```

```
        b2.setAuthor(new Author(sc.next(), sc.nextInt()));
        System.out.println("Original Book:\n" + b1);
        System.out.println("Cloned Book:\n" + b2);
        sc.close();
    }
}
```

Public test case 1:
Input:

```
Programming_Java
Alice 25
Bob 28
```

Output:

```
Original Book:
Book: Programming_Java
Author: Alice (Age: 25)
Cloned Book:
Book: Programming_Java
Author: Bob (Age: 28)
```

Public test case 2:
Input:

```
History_of_Science
Isaac 35
Elena 29
```

Output:

```
Original Book:
Book: History_of_Science
Author: Isaac (Age: 35)
Cloned Book:
Book: History_of_Science
Author: Elena (Age: 29)
```

Private test case 1:
Input:

```
Ancient_History
John 30
Erick 32
```

Output:

```
Original Book:
Book: Ancient_History
Author: John (Age: 30)
Cloned Book:
Book: Ancient_History
Author: Erick (Age: 32)
```

Solution:

```
import java.util.Scanner;
class Author implements Cloneable {
    private String name;
    private int age;
    public Author(String n, int a) {
        name = n;
        age = a;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "Author: " + name + " (Age: " + age + ")";
    }
    public Author clone() throws CloneNotSupportedException {
        return (Author) super.clone();
    }
}
class Book implements Cloneable {
    private String title;
    private Author author;
    public Book(String t, Author a) {
        title = t;
        author = a;
    }
```

```
    public void setAuthor(Author author) {
        this.author.setName(author.getName());
        this.author.setAge(author.getAge());
    }
    public String toString() {
        return "Book: " + title + "\n" + author;
    }
    public Book clone() throws CloneNotSupportedException {
        Book clonedBook = (Book) super.clone();
        clonedBook.author = this.author.clone();
        return clonedBook;
    }
}
public class CloningTest {
    public static void main(String[] args) throws CloneNotSupportedException {
        Scanner sc = new Scanner(System.in);
        Book b1 = new Book(sc.next(), new Author(sc.next(), sc.nextInt()));
        Book b2 = b1.clone();
        b2.setAuthor(new Author(sc.next(), sc.nextInt()));
        System.out.println("Original Book:\n" + b1);
        System.out.println("Cloned Book:\n" + b2);
        sc.close();
    }
}
```

0.4 Session 1 Type 3 Copy 2 Exceptions

Problem Statement

Complete the Java code to raise an exception when the input age is either lower than the age to vote or is higher than the age to appear for UPSC exams or both.

Class `InvalidAgeException` is a user-defined exception.

Class `AgeExceptionTest` has the following members:

- Static method `isEligibletoVote(int age)` that takes an age (as a int value) and returns true if `age >= 18`.
- Static method `isEligibletoWriteUPSC(int age)` that takes an age (as a int value) and returns true if `age <= 32`.

  If `age < 18` then the method should throw `InvalidAgeException` printing the message: `Invalid age to vote`. Else, if `age > 32`, then the method should throw `InvalidAgeException` printing the message `Invalid age to write UPSC`.
- `main` method that takes as input an age, invokes methods to check if a person of that age is eligible to vote, and invokes another method to see if the person is eligible to write the UPSC exams. If both are satisified, then it prints `Eligible to vote and to write UPSC`.

What you have to do:

- Define a user-defined exception: `InvalidAgeException`
- Define method `isEligibletoVote` inside class `InputExceptionTest`
- Define method `isEligibletoWriteUPSC` inside class `InputExceptionTest`

Test Cases

Public test case 1 (Input):

```
10
```

Output:

```
Invalid age to vote
```

Test Cases

Public test case 2 (Input):

```
33
```

Output:

```
Invalid age to write UPSC
```

Private test case 1 (Input):

```
18
```

Output:

```
Eligible to vote and to write UPSC
```

Private test case 2 (Input):

```
43
```

Output:

```
Invalid age to write UPSC
```

Template Code

```
import java.util.Scanner;

//DEFINE a user-defined exception: InvalidAgeException

public class AgeExceptionTest {

    //DEFINE method isEligibletoVote
    //DEFINE method isEligibletoWriteUPSC

    public static void main(String[] args) {
        int age;

        Scanner sc = new Scanner(System.in);
        age = sc.nextInt();

        try{
            isEligibletoVote(age);
            isEligibletoWriteUPSC(age);
            System.out.println("Eligible to vote and to write UPSC");
        }
        catch(InvalidAgeException ie){
```

```
            System.out.println(ie.getMessage());
        }
        sc.close();
    }
}
```

Solution:

```
import java.util.Scanner;
class InvalidAgeException extends Exception{
    public InvalidAgeException(String msg){
        super(msg);
    }
}
public class AgeExceptionTest {
    public static boolean isEligibletoVote(int age)
                                    throws InvalidAgeException{
        if (age < 18)
            throw new InvalidAgeException("Invalid age to vote");
        return true;
    }
    public static boolean isEligibletoWriteUPSC(int age)
                                    throws InvalidAgeException{
        if (age > 32)
            throw new InvalidAgeException("Invalid age to write UPSC");
        return true;
    }
    public static void main(String[] args) {
        int age;

        Scanner sc = new Scanner(System.in);
        age = sc.nextInt();

        try{
            isEligibletoVote(age);
            isEligibletoWriteUPSC(age);
            System.out.println("Eligible to vote and to write UPSC");
        }
        catch(InvalidAgeException ie){
            System.out.println(ie.getMessage());
        }
        sc.close();
```

```
    }
}
```