BSCCS2005: Sep 2023 OPE1 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.

1 Session 2

1.1 Session 2 Type 1 Copy Constructor

Problem Statement

In a college, `Student s1` chooses a set of courses. `Student s2` also chooses all the courses chosen by `s1` except the second course, in place of which `s2` chooses another course. Write a program that defines two classes `Student` and `Admission`. Define copy constructor to create `s2` from `s1` such that changing the values of instance variables of either `s2` or `s1` does not affect the other one. The code takes `name` of student `s2` and the new course chosen by `s2` as input.

- Class `Student` has/should have the following members.
  - Private instance variables `String name` and `String[] courses` to store name and courses chosen respectively
  - Define required constructor(s)
  - Accessor methods `getName( )` and `getCourses(int)` to get the name of the student and the course at a specific index respectively.
  - Mutator methods `setName(String)` and `setCourses(int,String)` to set the name of the student and the course at a specific index respectively.
- Class `Admission` has method `main` that does the following.
  - Two objects of `Student s1` and `s2` are created. `s2` is created using `s1`
  - `name` of `Student s2` and second course chosen by `s2` are updated by taking the input
  - Finally, name of `s1`, `s2` and second course chosen by `s1` and `s2` are printed

What you have to do

- Define constructor(s) in class `Student`

Template Code

```
import java.util.*;
class Student{
    String name;
    String[] courses;
    //***** Define constructor(s) here
    public void setName(String n) {
        name = n;
    }
    public void setCourses(int indx, String c) {
```

```
        courses[indx] = c;
    }
    public String getName() {
        return name;
    }
    public String getCourses(int indx) {
        return courses[indx];
    }
}
public class Admission {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] courses = {"Maths", "DL","DSA","DC"};
        Student s1 = new Student("Nandu", courses);
        Student s2 = new Student(s1);
        s2.setName(sc.next());
        s2.setCourses(1,sc.next());
        System.out.println(s1.getName() + ": "+ s1.getCourses(1));
        System.out.println(s2.getName() + ": " + s2.getCourses(1));
    }
}
```

Test cases:

Public test case 1:
Input:

```
Suba COA
```

Output:

```
Nandu: DL
Suba: COA
```

Public test case 2:
Input:

```
Pai CV
```

Output:

```
Nandu: DL
Pai: CV
```

Private test case 1:
Input:

```
Neha DS
```

Output:

```
Nandu: DL
Neha: DS
```

Solution:

```
import java.util.*;
class Student{
    String name;
    String[] courses;
    public Student(String n, String[] c) {
        name = n;
        courses=c;
    }
    public Student(Student s) {
        this.name = s.name;
        this.courses = new String[s.courses.length];
        for(int i = 0; i < courses.length; i++) {
            this.courses[i] = s.courses[i];
        }
    }
    public void setName(String n) {
        name = n;
    }
    public void setCourses(int indx, String c) {
        courses[indx] = c;
    }
    public String getName() {
        return name;
    }
    public String getCourses(int indx) {
        return courses[indx];
    }
}
public class Admission {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] courses = {"Maths", "DL","DSA","DC"};
        Student s1 = new Student("Nandu", courses);
        Student s2 = new Student(s1);
        s2.setName(sc.next());
        s2.setCourses(1,sc.next());
```

```
        System.out.println(s1.getName() + ": "+ s1.getCourses(1));
        System.out.println(s2.getName() + ": " + s2.getCourses(1));
    }
}
```

Question:

Write a Java code to print the name of a passenger whose PNR number is given. The name should be printed only for valid PNR numbers. The code should use the concepts of inner classes in Java, and should have the following functionality.

Class PassengerList has/should have the following members:

- A list of valid PNR numbers `pnrList` as an instance variable.
- Constructor `PassengerList()` populates the list `pnrList` as given in the template code.
- Method `public PassengerInfo getPassengerInfo(String)` should take a PNR number as argument, check if the PNR number is valid, and if yes, then return a valid `PassengerInfo` object that has the name of the passenger. For ease of implementation, we assume that the valid PNR numbers are 1 to 3.
- An inner private class `PassengerInfo`
  - Name of a specific passenger as an instance variable.
  - This class implements interface `IPassengerInfo`, which enables its object to be accessible from outside the class `PassengerList`.
  - Constructor `PassengerInfo(String)` assigns the name of the passenger to the instance variable of class `PassengerInfo` whose PNR number is given. If the PNR number is i, then the name should be `Passenger` i, where the valid PNR numbers are for i ranging from 1 to 3.

Class PassengerListTest has the main method.

- It accepts a PNR number as input, and invokes the necessary method to obtain the passenger name for that PNR number.

Test Cases:
Public Test Cases:
Test Case 1:

```
Input:
1
Output:
Passenger 1
```

Test Case 2:

```
Input:
4
Output:
Not a valid passenger
```

Private Test Cases:
Test Case 1:

```
Input:
2
Output:
Passenger 2
```

Test Case 2:

```
Input:
4123
Output:
Not a valid passenger
```

Solution:

```
import java.util.*;

interface IPassengerInfo {
    public void printName();
}

class PassengerList {
    private ArrayList<String> pnrList;

    public PassengerList() {
        pnrList = new ArrayList<String>();
        for (int i = 1; i < 4; i++) {
            pnrList.add("" + i);
        }
    }

    public PassengerInfo getPassengerInfo(String passPnr) {
        PassengerInfo pInfo = null;
        if (pnrList.contains(passPnr))
            pInfo = new PassengerInfo(passPnr);
        return pInfo;
    }

    private class PassengerInfo implements IPassengerInfo {
        String passName;

        public PassengerInfo(String sPnr) {
```

```
            passName = "Passenger " + sPnr;
        }

        public void printName() {
            System.out.println(passName);
        }
    }
}

public class PassengerListTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String passPnr = sc.nextLine();
        PassengerList passList = new PassengerList();
        IPassengerInfo passInfo = passList.getPassengerInfo(passPnr);
        if (passInfo == null) {
            System.out.println("Not a valid passenger");
        } else {
            passInfo.printName();
        }
        sc.close();
    }
}
```

Question:

Complete the Java program to demonstrate the use of abstract classes and interfaces. You have to complete the definition of classes JuniorRS and SeniorRS to obtain the output as given in the public test cases.

- Interface `IResearchScholar` has two methods: `public void teaches(String str)` and `public void studies(String str)`.
- Define classes `JuniorRS` and `SeniorRS` such that `JuniorRS` implements `IResearchScholar` and `SeniorRS` extends `JuniorRS`.
- Class `InterAbstrTest` extends `SeniorRS`, and has the main method. An object of `JuniorRS` invokes the method `studies`, and an object of `SeniorRS` invokes methods `studies` and `teaches`.

Public Test Cases:
Test Case 1:

```
Input 1:
Python
Java
Output 1:
TA studies Python
TA studies Java
TA teaches Java
```

Test Case 2:

```
Input 2:
Cloud computing
Data Mining
Output 2:
TA studies Cloud computing
TA studies Data Mining
TA teaches Data Mining
```

Private Test Cases:
Test Case 1:

```
Input 1:
Machine Learning
Machine Learning
Output 1:
TA studies Machine Learning
TA studies Machine Learning
TA teaches Machine Learning
```

Test Case 2:

```
Input 2:
Cloud computing
Data Mining
Output 2:
TA studies Cloud computing
TA studies Data Mining
TA teaches Data Mining
```

Solution:

```
import java.util.Scanner;

interface IResearchScholar {
    public void teaches(String str);
    public void studies(String str);
}

abstract class JuniorRS implements IResearchScholar {
    public void studies(String str1) {
        System.out.println("TA studies " + str1);
    }
}

class SeniorRS extends JuniorRS {
    public void teaches(String str) {
        System.out.println("TA teaches " + str);
    }
}

public class InterAbstrTest extends SeniorRS {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str1 = sc.nextLine();
        String str2 = sc.nextLine();
        JuniorRS jrs = new InterAbstrTest();
        SeniorRS srs = new InterAbstrTest();
        jrs.studies(str1);
        srs.studies(str2);
        srs.teaches(str2);
        sc.close();
    }
}
```

Dynamic Dispatch

Problem Statement

Complete the Java code that uses the concept of inheritance to demonstrate dynamic method dispatching.

- Create a class `Vehicle` with the following members:
  - Private instance variable `name`.
  - Constructor to initialize `name`.
  - Accessor method for `name`.
  - Method `display` to display the text: "This is a generic vehicle."

  Classes `Car` and `Bicycle` should be defined in such a way that any object of `Car` or `Bycycle` can be assigned to a reference variable of type `Vehicle`. See the `main` method to understand the context.
  - For `Car`, the method `display` should print: "This is a car named name ."
  - For `Bicycle`, the method `display` should print: "This is a bicycle named name ."

In the `main` method of the `DispatchExample` class, create an array of `Vehicle` objects with size 3.

- Initialize the first element with a generic vehicle (you can use an empty string for its name).
- Initialize the second and third elements with a `Car` and a `Bicycle`, respectively, by taking the vehicle's name as input from the user.

Iterate over the array and call the `display` method for each vehicle.

Sample Input/Output

Input:

```
BMW
Giant
```

Output:

```
This is a generic vehicle.
This is a car named BMW.
This is a bicycle named Giant.
```

Template Code

```
import java.util.Scanner;

class Vehicle {
    private String name;

    public Vehicle(String n) {
        name = n;
    }
    // Define method display
    // Define an accessor method

}

//Define class Car
//Define class Bicycle

public class DispatchExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Vehicle[] vehicles = new Vehicle[3];

        vehicles[0] = new Vehicle("");
        vehicles[1] = new Car(sc.nextLine());
        vehicles[2] = new Bicycle(sc.nextLine());

        for (Vehicle v : vehicles) {
            v.display();
        }

        sc.close();
    }
}
```

Solution

```
import java.util.Scanner;

class Vehicle {
    private String name;

    public Vehicle(String n) {
        name = n;
```

```
    }

    public String getName() {
        return name;
    }

    public void display() {
        System.out.println("This is a generic vehicle.");
    }
}

class Car extends Vehicle {
    public Car(String n) {
        super(n);
    }

    public void display() {
        System.out.println("This is a car named " + getName() + ".");
    }
}

class Bicycle extends Vehicle {
    public Bicycle(String n) {
        super(n);
    }

    public void display() {
        System.out.println("This is a bicycle named " + getName() + ".");
    }
}

public class DispatchExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Vehicle[] vehicles = new Vehicle[3];

        vehicles[0] = new Vehicle("");
        vehicles[1] = new Car(sc.nextLine());
        vehicles[2] = new Bicycle(sc.nextLine());

        for (Vehicle v : vehicles) {
            v.display();
        }
```

```
        sc.close();
    }
}
```