00076

Student ID:

Machine No:

Sri Lanka Institute of Information Technology

B.Sc. Honours Degree in Information Technology

Specialized in Information Technology

Final Examination (Computer Based)
Year 2, Semester 1 (2023)
Paper Version D

IT2030 – Object Oriented Programming

Duration: 3 Hours

November 2023

Instructions to Candidates:

- This paper contains four questions. Answer All Questions.
- Marks for each question are given on the paper. Total Marks: 100.
- Create a folder on the Desktop with your student registration number and store all your program files inside that.
- Create a separate Project for each question. The name of the project is provided in the question.
- This paper contains 08 pages including the Cover Page.

Instructions to Candidates when submitting:

- Save all your work.
- Delete all unnecessary files from each project folder (There should be 4 folders for 4 projects named as Question01, Question02, Question03 and Question04 inside your ID folder, and each folder should contain the source code (.java files)) only.
- Zip the Student ID folder (Zipped folder also should be named with Student ID number).
- Upload the zipped folder to the correct link.

Question 1

(20 marks)

This question is based on the **Collection Framework and Generics**.

a) You should implement an array list of Students and Lecturers and use one Generic class called **GenericPerson** to display elements in both array lists. Please refer the **GenericPersonDemo** Test class and its execution output to fine-tune your results.

```
15 public class GenericPersonDemo {
16
17     public static void main(String[] args) {
18         ArrayList<Student> students = new ArrayList<>();
19         students.add(new Student("STD1111", 6));
20         students.add(new Student("STD2222", 7));
21         students.add(new Student("STD3333", 12));
22         students.add(new Student("STD4444", 11));
23         students.add(new Student("STD5555", 10));
24
25         ArrayList<Lecturer> lecturers = new ArrayList<>();
26         lecturers.add(new Lecturer("EMP0000", "IT"));
27         lecturers.add(new Lecturer("EMP1111", "SE"));
28         lecturers.add(new Lecturer("EMP2222", "CSN"));
29         lecturers.add(new Lecturer("EMP3333", "EE"));
30         lecturers.add(new Lecturer("EMP4444", "IS"));
31
32         GenericPerson genericPerson = new GenericPerson();
33         genericPerson.displayElements(students);
34         genericPerson.displayElements(lecturers);
35     }
36 }
```

```
Console
<terminated> GenericPersonDemo [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\jav
Student = STD1111, Grade = 6
Student = STD2222, Grade = 7
Student = STD3333, Grade = 12
Student = STD4444, Grade = 11
Student = STD5555, Grade = 10

Lecturer = EMP0000, Department = IT
Lecturer = EMP1111, Department = SE
Lecturer = EMP2222, Department = CSN
Lecturer = EMP3333, Department = EE
Lecturer = EMP4444, Department = IS
```

i) Create Implement an interface **IPerson** and declare the method displayDetails() should return the output in String type. (02 marks)

ii) Create a class called **Student** and implement the two properties called studetID (String) and grade (int) and values should be assigned through the overloaded constructor. (02 marks)

iii) Implement the **IPerson** interface in the **Student** class and override the method displayDetails() to print the student ID and the grade. (02 marks)

iv) Create a class called **Lecturer** and implement the two properties called employeeID (String) and department (String) and the values should be assigned through the overloaded constructor. (02 marks)

v) Implement the **IPerson** interface in the **Lecturer** class and override the method displayDetails() to print the employeeID and the department. (02 marks)

vi) Now create the generic class called **GenericPerson** and implement the method displayElements should support passing generic array list (either Lecturers array list or Students array list). The displayElements() method should have an iteration and within the iteration, the each element should call the displayDetails() method to print the Lecturer and Student details as per the given output. (05 marks)

Save the project as **Question01A**

b) You should create a class called **AscendingTable** and that should store elements as key, value pairs. Keys should be stored according to the Ascending order. Implement the display() method that should print keys and values according to the ascending order. Refer the **GenericDemo** Test class and console output to adjust your results accordingly.

(05 marks)

```
18 public class GenericDemo {
19
20     public static void main(String[] args) {
21
22         AscendingTable<Integer, String> myTable = new AscendingTable<>();
23         myTable.add(40, "ddd");
24         myTable.add(10, "aaa");
25         myTable.add(30, "ccc");
26         myTable.add(20, "bbb");
27
28         AscendingTable<Integer, Double> myTableD = new AscendingTable<>();
29         myTableD.add(40, 10.123);
30         myTableD.add(30, 23.456);
31         myTableD.add(20, 34.567);
32         myTableD.add(10, 45.678);
33
34         AscendingTable.display(myTable);
35         AscendingTable.display(myTableD);
36     }
37 }
```

Console | Javadoc | Problems | Declaration | Servers | Data Source Explorer | Debug

<terminated> GenericDemo [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\javaw.exe (Sep 2, 2019, 12:27:05 AM

```
10, aaa
20, bbb
30, ccc
40, ddd
10, 45.678
20, 34.567
30, 23.456
40, 10.123
```

Save the project as **Question01B**

Question 2 (20 marks)

This question is based on the **Threads implementation**.

You are tasked with simulating the Producer Consumer chain in a factory. The factory uses Iron, Wood and Cement to make their products. The raw materials are supplied by different suppliers. Once raw materials are received it should increase the inventory and each time a product is produced, it should reduce 1 Iron, 2 Wood and 1 Cement from the inventory.

If raw material is unavailable the production line should be halted until the supplies arrive. Sample output is given below.

a) Create 3 classes called **SupplierIron**, **SupplierWood**, and **SupplierCement** that handle the supply of raw materials. Each class should supply 1 material every 600 milliseconds. (06 marks)

b) Create a **ProductionLine** class that handles production with a rate of 1 product every 1000 milliseconds. (04 marks)

c) Create **FactorySimulation** Class that maintains the inventory and handles the handles the production based on the available inventory. (10 marks)

Save the project as **Question02**

Sample Output:

```
Supplied 1 Wood. Wood Inventory: 1
Supplied 1 Cement. Cement Inventory: 1
Waiting for supplies...
Supplied 1 Iron. Iron Inventory: 1
Waiting for supplies...
Supplied 1 Iron. Iron Inventory: 2
Waiting for supplies...
Supplied 1 Wood. Wood Inventory: 2
Supplied 1 Cement. Cement Inventory: 2
Product produced. Iron: 1, Wood: 0, Cement: 1
Supplied 1 Cement. Cement Inventory: 2
Supplied 1 Wood. Wood Inventory: 1
Supplied 1 Iron. Iron Inventory: 2
Waiting for supplies...
```

Question 3 (30 marks)

This question is based on the **Design Patterns** implementation.

a) You are going to implement the **Abstract Factory Pattern** based on the scenario of **Vehicle** factory that turns out Cars or Busses for the specified model. Refer the Screenshot of the Console Output and the **VehicleDemo** class. You should implement the rest of the other classes.

i). Implement **VehicleProducer** class that returns either **CarFactory** or **BusFactory** outputs through **getVehicle()** operation. Refer the screenshot in Demo class.

(02 marks)

ii). Design **CarFactory** and **BusFactory** classes according to the **Singleton Pattern** and let **getModel(String vehicle)** select the factories.

(06 marks)

iii). Design 03 classes for Car factory called (**BMW, Benz, and RollsRoys**) according to the **singleton pattern**. Each car should have a **displayVehicle()** method you have to override.

(03 marks x 03 = 09 marks)

iv). Create Design 03 classes for Bus factory called (**Volvo, Fuso, and TATA**) according to the **singleton pattern**. Each bus should have a **displayVehicle()** method you must override.

(03 marks x 03 = 09 marks)

b) Now design three interfaces to **IBus**, **ICar**, and **VehicleFactory** to override common behaviors.

i). Now design **IBus** and **ICar** interfaces and declare the **displayVehicle()** operation in each.

(01-mark x 2 = 02 marks)

ii). Design **VehicleFactory** as an Interface and declare the **getModel** operation to select vehicle option either Bus or Car

(02 marks)

Hint: Refer to the main program and package hierarchy given below

Save the project as **Question03**

Main Program:

```
3  public class VehicleSelectionDemo {
4
5      private static final String CAR = "Car";
6      private static final String BUS = "Bus";
7
8      public static void main(String[] args) {
9
10         ((ICar) VehicleProducer.getVehicle(CAR).getModel("RollsRoys")).displayVehicle();
11         ((ICar) VehicleProducer.getVehicle(CAR).getModel("Benz")).displayVehicle();
12         ((ICar) VehicleProducer.getVehicle(CAR).getModel("BMW")).displayVehicle();
13
14         ((IBus) VehicleProducer.getVehicle(BUS).getModel("Volvo")).displayVehicle();
15         ((IBus) VehicleProducer.getVehicle(BUS).getModel("Fuso")).displayVehicle();
16         ((IBus) VehicleProducer.getVehicle(BUS).getModel("TATA")).displayVehicle();
17     }
18 }
```

Console × Problems Javadoc Declaration

<terminated> VehicleSelectionDemo [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Sep 29, 2023, 1:26:40 PM)

```
Factory turns out RollsRoys Car
Factory turns out Benz Car
Factory turns out BMW Car
Factory turns out Volvo Bus
Factory turns out Fuso Bus
Factory turns out TATA Bus
```

Package Hierarchy:

- oop.exam.q1
  - Benz.java
  - BMW.java
  - BusFactory.java
  - CarFactory.java
  - Fuso.java
  - IBus.java
  - ICar.java
  - RollsRoys.java
  - TATA.java
  - VehicleFactory.java
  - VehicleProducer.java
  - VehicleSelectionDemo.java
  - Volvo.java

Question 4 (30 marks)

This question is based on the **OOP concepts**.

You are going to control two types of satellites called Drone Satellite and Navigational Satellite from one location called Satellite Center.

a) You can refer to the output given in **SatelliteDemo** class and adjust your code accordingly.

```
3  public class SatelliteDemo {
4
5      public static void main(String[] args) {
6
7          ISatellite navigationalSatellite = new NavigationSatellite("Ravana-01");
8          IGeoLocation locationTracker1 = new SatelliteLocation("Sri Lanka");
9          ISatellite droneSatellite = new DroneSatellite("Ravana-02");
10         IGeoLocation locationTracker2 = new SatelliteLocation("Russia");
11
12         ISatellite [] satelliteArray = new ISatellite[]{navigationalSatellite, droneSatellite};
13         IGeoLocation [] trackerArray = new IGeoLocation[]{locationTracker1, locationTracker2};
14
15         SatelliteCenter satelliteCenter = new SatelliteCenter(0, satelliteArray,trackerArray);
16         satelliteCenter.startService();
17         satelliteCenter.stopService();
18         satelliteCenter.locationService();
19
20         SatelliteCenter remoteController2 = new SatelliteCenter(1, satelliteArray,trackerArray);
21         remoteController2.startService();
22         remoteController2.stopService();
23         remoteController2.locationService();
24
25 }
```

Console

```
<terminated> SatelliteDemo [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\javaw.exe (Sep 2, 2019, 9:06:47 PM)
Ravana-01 navigational satellite activate
Ravana-01 navigational satellite deactivate
Satellite Location is = Sri Lanka

Ravana-02 drone satellite activate
Ravana-02 drone satellite deactivate
Satellite Location is = Russia
```

i). First implement the **ISatellite** interface and declare activate() and deactivate() methods. (03 marks)

ii). Then implement the **IGeoLocation** interface and declare the method called displayLocation() (02 marks)

iii). Create two classes called **DroneSatellite** and **NavigationSatellite** and implement the ISatellite interface in each class and override necessary methods in each. You should overload the constructor to pass the name of the satellite in both classes. (4 X 2 = 08 marks)

iv). Similarly create a class called **SatelliteLocation** and implement the **IGeoLocation** interface within the class and override the displayLocation() method. Then overload the constructor to pass the location of the satellite. (03 marks)

b) The satellite center maintains multiple satellites and multiple Geo Location trackers. To activate each satellite and the tracker the option can be used as a switch.

i). Create the **SatelliteCenter** class and implement the properties option(int), and array of ISatellite (ISatellite [ ]) and the array of IGeoLocation (IGeoLocation [ ]) tracker. (02 marks)

ii). Overload the constructor of the same class and initialize the above properties. (03 marks)

iii). Implement the method called startService() and you should invoke the activate() method of the satellite class by using the option as switch. [E.g.: - if option = 0 activate Navigation Satellite if option = 1 activate drone satellite] (02 marks)

iv). Implement the method called stopService() and you should invoke the deactivate() method. (02 marks)

v). Then develop the locationService() method and based on the given option tracker should invoke the displayLocation() method. (02 marks)

vi). Add another DroneSatellite and the tracker in the **SatelliteDemo** and display your modified output again in the console. (03 marks)

Save the project as Question04