CONFIDENTIAL

FACULTY OF COMPUTING
UTM Johor Bahru

UNIVERSITI TEKNOLOGI MALAYSIA

FINAL EXAM SEMESTER II, 2022/2023

SUBJECT CODE	**:**	**SECJ2154**
SUBJECT NAME	**:**	**OBJECT ORIENTED PROGRAMMING**
YEAR/PROGRAM	**:**	**2 (SECB/ SECJ/ SECP/ SECR/ SECV)**
DURATION	**:**	**3 HOURS**
DATE	**:**	
VENUE	**:**	

INSTRUCTIONS:

THIS EXAMINATION BOOK CONSISTS OF 2 SECTIONS:

SECTION A:	Structured Questions	70 Marks
SECTION B:	Programming	30 Marks

ANSWER ALL QUESTIONS IN THE ANSWER BOOKLET PROVIDED
QUESTION BOOKLET MUST BE RETURN WITH THE ANSWER BOOKLET

(Please write your lecturer name and section in your answer booklet)

Name	
IC	
Year/Program	
Section	
Lecturer Name	

This question paper consists of **15** (**FIFTEEN**) printed pages excluding this page.

SECTION A: STRUCTURED QUESTIONS (70 Marks)

Section A consists of 5 structured questions. Answer all questions and write your answer in the answer booklet.

Question 1 (10 Marks)

1.1 What are the major benefits of using Vector instead of a conventional array in Java?

(3 marks)

1.2 Consider the following Java program (Figure 1.1):

```
1  import _____________________;
2  import _____________________;
3
4  public class TestList {
5    public static void main(String[] args) {
6      ArrayList<String> arrayList1 = new ArrayList<>();
7      arrayList1.add("Apple");
8      arrayList1.add("Banana");
9      arrayList1.add("Orange");
10
11     ArrayList<Integer> arrayList2 = new ArrayList<>(2);
12     for (int i = 1; i <= 5; i++) {
13       arrayList2.add(i);
14     }
15
16     Vector<String> vector1 = new Vector<>();
17     vector1.addElement("Cat");
18     vector1.addElement("Dog");
19     vector1.addElement("Elephant");
20
21     Vector<Double> vector2 = new Vector<>(2, 5);
22     vector2.addElement(3.14);
23     vector2.addElement(2.718);
24     vector2.addElement(1.618);
25     System.out.println(arrayList2.size() + " - " +
26     vector2.capacity());
27   }
28 }
```

Figure 1.1: Java program to test `Vector` and `ArrayList` objects

Based on the source code provided in Figure 1.1 above, please answer the following questions:

i) Complete the required statements in lines `1` and `2`.

(2 marks)

ii) Complete the blank parts of the following statements:

a) The `"Orange"` element in `arrayList1` is located at index ____

b) The initial capacity of `arrayList2` is ____

c) Statement to remove `"Banana"` from the list: _____;

d) Statement to replace `"Cat"` with `"Bird"`: _____;

e) The final output of the program is _____

(5 marks)

Question 2 (15 Marks)

2.1 Given the class definition as in Figure 2.1 below:

```
public class Buyer {
  private Drone[] droneList;

  public void buy(Drone d){ . . . }
}

public class Drone {
  private Producer producer;
  public void setProducer(Producer p){. . .}
}

public class Producer{
  private Drone[] droneList;
  public void produce(Drone d){. . .}
}
```

Figure 2.1: Definitions of Buyer, Drone, and Produces classes

i) Draw the class diagram that shows the relationship among **`Buyer`**, **`Drone`** and **`Producer`** classes.

(4 marks)

ii) What type of relationship among **`Buyer`**, **`Drone`** and **`Producer`** classes?

(1 mark)

2.2 Given the class diagram (Figure 2.2), with 4 classes (`Hp`, `NumberCode`, `SimCard`, and `HPTest`).

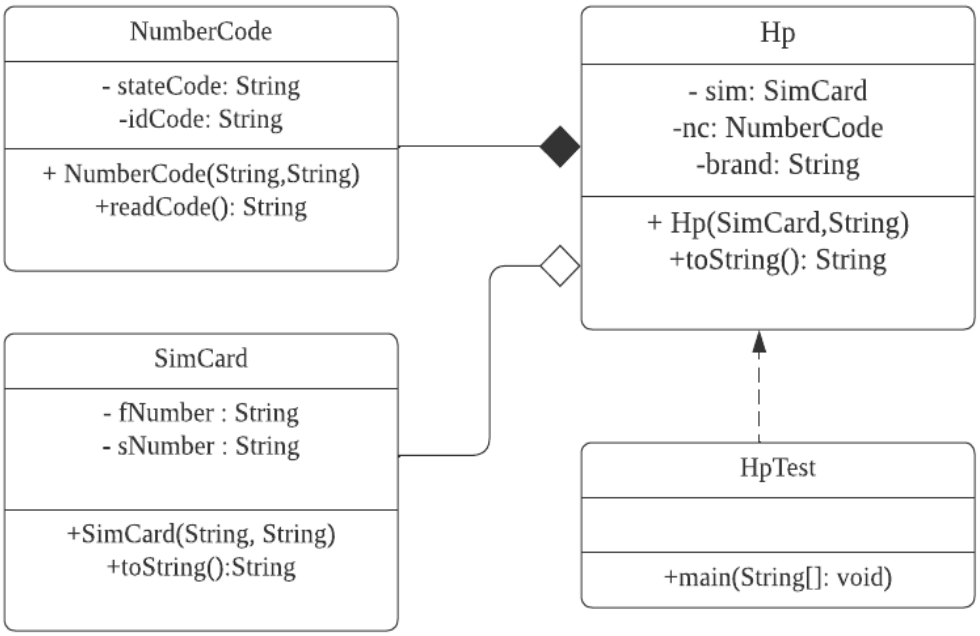

Figure 2.2: A class diagram of `Hp`, `NumberCode`, `SimCard`, and `HPTest` classes.

Figure 2.3 shows the Java program that implements the class diagram depicted in Figure 2.2. The final output of the program is shown in Figure 2.4.

```
public class HpTest {
  public static void main(String[] args) {
    SimCard sim1 = new SimCard("8991", "0313");
    Hp mob = new Hp(sim1, "Samsung");
    System.out.println(mob.toString());
  }
} // End of HpTest class

class SimCard {
  private String fNumber;
  private String sNumber;

  public SimCard(String n1,String n2) {
    fNumber=n1;
    sNumber=n2;
  }

  public String toString() {
    return (fNumber + sNumber);
  }
} // end of SimCard class
```

```
class NumberCode {
  private String stateCode;
  private String idCode;

  public NumberCode (String s, String i) {
    stateCode = s;
    idCode = i;
  }

  public String readCode() {
    return stateCode + "-" + idCode;
  }
} //End of NumberCode class

class Hp {
// i) attributes of the class


// ii) constructor


// iii) toString() method


} // End of Hp class
```

Figure 2.3: Java program to implement class diagram in Figure 2.2

```
Brand: Samsung, Number Code: 1113-CH43, SIM Card number: 89910313
```

Figure 2.4: The output generated by the `main` method of `HpTest` class

Using the information provided in Figures 2.2, 2.3, and 2.4 above, please answer the following questions:

i) Write the code to define the attributes of `Hp` class

(2 marks)

ii) Write the complete code of `HP` class constructor that initializes the `sim`, `nc`, and `brand` instance variables

(5 marks)

iii) Write the `toString` method that will display the output as shown in Figure 2.4

(3 marks)

Question 3 (15 Marks)

3.1 Consider the following Java source code in Figure 3.1.

```
1  class Car {
2    private String brand;
3    private double price;
4
5    public Car(String brand, double price ) {
6
7      // i) assign Car class attribute's values
8      _____________________________;
9      _____________________________;
10   }
11
12   public void display() {
13     System.out.println(brand);
14     System.out.println(price);
15   }
16 }
17
18 class Sedan extends Car {
19   private int numOfDoors;
20
21   public Car() {
22     // ii) assign subclass and superclass attribute's values
23     _____________________________;
24     _____________________________;
25
26   }
27
28   public void display() {
29     // iii) Display brand, price, and numOfDoors
30     _____________________________;
31     _____________________________;
32
33   }
34 }
35
36 public class TestInheritance {
37   public static void main(String[] args) {
38     // iv) Create subclass instance (civic)
39     _____________________________;
40
41     // v) Call the overridden display() method
42     _____________________________;
43   }
44 }
```

Figure 3.1: A Java source code that defines `Car`, `Sedan`, and `TestInheritance` classes

Please answer the following questions based on the classes defined in Figure 3.1 above.

i) Assign values for `Car` superclass attributes

(2 marks)

ii) Assign values to `Sedan` subclass attribute (`numOfDoors=4`) and its superclass inherited attributes (`brand="Honda", price=78800`)

(3 marks)

iii) Display `brand, price,` and `numOfDoors` of `Sedan.`

(2 marks)

iv) Create a `Sedan` object named as `Civic`

(1 mark)

v) Calls the overridden `display()` method in the `Sedan` class

(1 mark)

3.2 Consider the implementations of `Vacation`, `Beach`, and `Cuti` classes in Figure 3.2 below:

```
 1  class Vacation {
 2    private String destination;
 3    int duration;
 4
 5    public Vacation() {
 6      System.out.println("Welcome To");
 7    }
 8
 9    public Vacation(String destination, int duration ) {
10      this.destination = destination;
11      this.duration=duration;
12      System.out.println("Enjoy your trip");
13    }
14  }
15
16  class Beach extends Vacation {
17    private String travelAirline;
18
19    public Beach() {
20      this("AirAsia");
21      System.out.println("Have a wonderful trip");
22    }
23
24    public Beach(String travelAirline, String destination) {
25      this(travelAirline);
26      System.out.println(destination);
27    }
28
29    public Beach(String travelAirline) {
30      this.travelAirline = travelAirline;
31      System.out.println(travelAirline);
32    }
33  }
34
35  public class Cuti {
36    public static void main(String[] args) {
37      Beach myVacation = new Beach("MAS", "Maldives");
38    }
39  }
```

Figure 3.2: Implementation of `Vacation`, `Beach`, and `Cuti` classes in a Java program

Based on the class implementations provided in Figure 3.2 above, answer the following questions:

i) What is the output of the program?

(3 marks)

ii) What will be the output displayed if the instruction in line 37 is changed to:

```
Vacation myVacation = new Beach();
```

(3 marks)

Question 4 (15 Marks)

4.1 The Java program in Figure 4.1 below demonstrates the abstract class concept through the definition of `Shape`, `Circle`, `Rectangle`, and `TestShapeAbstract` classes.

```
1  abstract class Shape {
2    protected String color;
3
4    public Shape(String color) {
5      this.color = color;
6    }
7
8    public double abstract calculateArea();
9
10   public void display() {
11     System.out.println("This is a " + color + " shape.");
12   }
13 }
14
15 class Circle implements Shape {
16   private double radius;
17
18   public Circle(String color, double radius) {
19     this.color=color;
20     this.radius = radius;
21   }
22
23   public double calculateArea() {
24     return Math.PI * radius * radius;
25   }
26 }
27
28 class Rectangle extends Circle {
29   private double length;
30   private double width;
31
32   public Rectangle(String color, double length, double width) {
33     super.color = color;
34     this.length = length;
35     this.width = width;
36   }
37
38    public double calculateArea() {
39     return length * width;
40   }
41 }
42
43 public class TestShapeAbstract {
44   public static void main(String[] args) {
45     Shape circle = new Circle("Red", 5.0);
46     Shape rectangle = new Rectangle("Blue", 4.0, 6.0);
47
```

```
48      circle.display();
49      System.out.println("Area of the circle: " +
50      circle.calculateArea());
51
52      rectangle.display();
53      System.out.println("Area of the rectangle: " +
54      rectangle.calculateArea());
55    }
56  }
```

Figure 4.1: Example of abstract class concept implementation in Java

Answer the following questions based on the Java program presented in Figure 4.1:

i) The Java program in Figure 4.1 contains 5 errors:1 error in the `Shape` class, 2 errors in the `Circle` class, and 2 errors in the `Rectangle` class. Complete the table below by identifying the line numbers of the errors and providing the statements to correct them.

(10 marks)

Line Number	**Statement for Correction**

ii) Modify the `TestShapeAbstract` class starting from line 45 by:

a. Declare and create an array of `Shape` objects named `shapeList` to store the `Rectangle` and `Circle` objects (do apply the same shape properties as shown in lines 45 and 46 in Figure 4.1)

b. Use enhanced for-loop statement to iterate the `shapeList` array and call the `display()` method

(5 marks)

Question 5 (15 Marks)

5.1 Given the class `Square` in Figure 5.1 below:

```
 1  public class Square {
 2    public static void main(String[] args) {
 3      Scanner scanner = new Scanner(System.in);
 4      int number = 0, square = 0;
 5
 6      System.out.print("Enter a number: ");
 7      number = scanner.nextInt();
 8      square = number * number;
 9
10      System.out.printf("The square of %d is %d", number, square);
11    }
12  }
```

Figure 5.1: Java program of class `Square`

What would be the output if the user enters a non-numeric value, such as a string or a character, for the statement in line 7 of the Java program depicted in Figure 5.1 above?

(2 marks)

5.2 Figure 5.2 is the proposed improved version of the `main` method of the Java program in Figure 5.1.

```
1   public static void main(String[] args) {
2      Scanner scanner = new Scanner(System.in);
3      int number = 0, square = 0;
4
5      try {
6         _______________________________; // 1 mark
7         _______________________________; // 1 mark
8
9         // check and throw exception (IllegalArgumentException)
10        // if the input is out of range (1 - 100)
11        if (_______________________________) {  // 2 marks
12           ____________________________________; // 2 marks
13        }
14
15        _______________________________; // 1 mark
16        _______________________________; // 1 mark
17
18     } ____________________________________ { // 1.5 marks
19
```

```
20          // out of range error message
21          ____________________________________; // 1 mark
22
23      } ____________________________________ { // 1.5 marks
24
25          // non-numeric input error message
26          ____________________________________; // 1 mark
27      }
28  }
```

Figure 5.2: An improved version of the `Square` class `main` method

Complete the Java code in Figure 5.2 (lines 5–27) to incorporate input error handling as demonstrated in Figure 5.3.

(13 marks)

```
C:\java Square
Enter a number (1 - 100): 5
The square of 5 is 25

C:\java Square
Enter a number (1 - 100): five
Error: non-numeric input

C:\java Square
Enter a number (1 - 100): 0
Error: out of range

C:\java Square
Enter a number (1 - 100): 101
Error: out of range
```

Figure 5.3: The list of input/output generated by the improved version of the `Square` class

SECTION B: PROGRAMMING (30 Marks)

Figure B1 depicts the class diagram of a Java program named **TripApp.java**. The expected output of this program is shown in **Figure B2**.

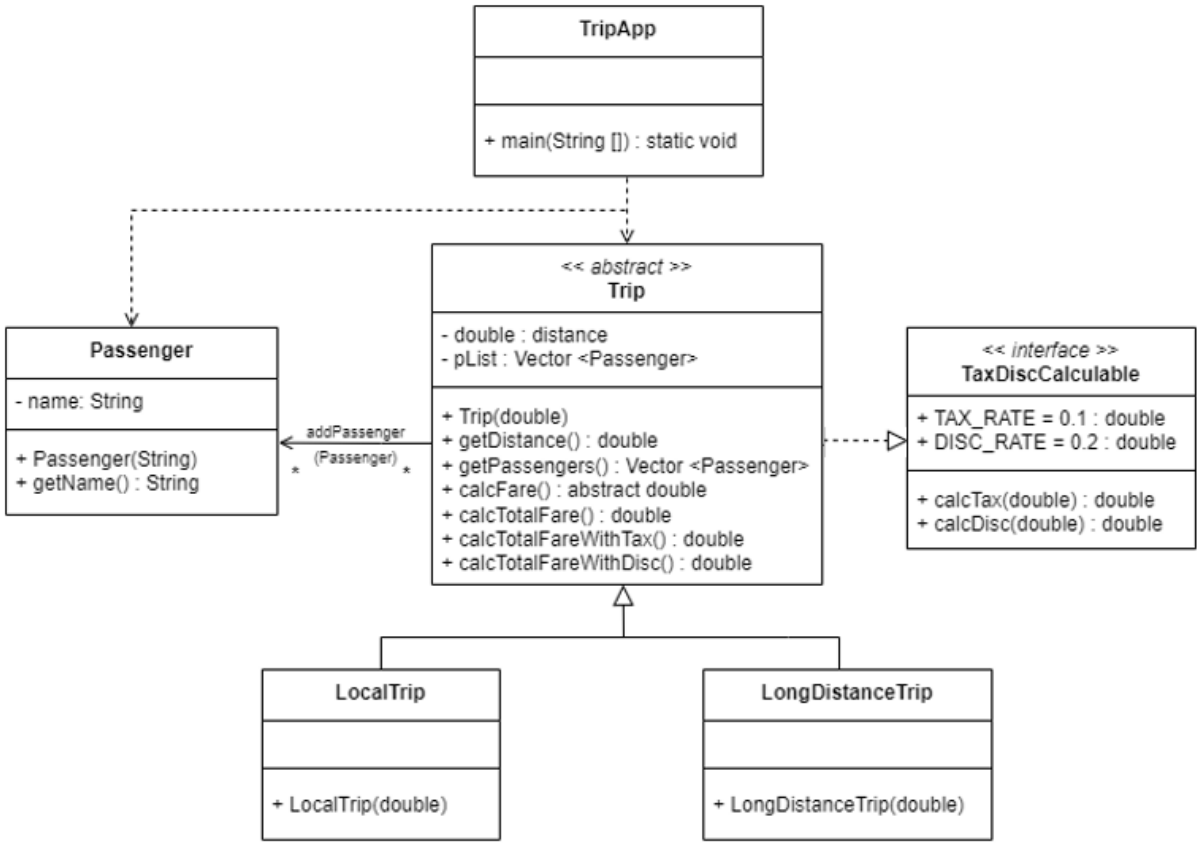

Figure B1: UML Class Diagram of **TripApp.java** program

```
Information for Local Trip
List of passengers:
1)  Alwi Ahmad
2)  Ahmad Shariff
3)  Rabiah Hakim

Total Fare: RM63.00
Total Fare with Tax: RM69.30
Total Fare with Discount: RM50.40

Information for Long-Distance Trip
List of passengers:
1)  Halimah Abu
2)  Rafidah Harun

Total Fare: RM150.00
Total Fare with Tax: RM165.00
Total Fare with Discount: RM120.00
```

Figure B2: Expected output of the program

You are required to complete the implementation of all the classes (excluding the **Passenger** and **TripApp** classes) with the specified instance variables (attributes) and methods as outlined in the class diagram presented in **Figure B1**. Please refer to the complete definitions of the **Passenger** and **TripApp** classes in **Figure B3.** The purpose of each class method is indicated by its name, and some of them are further explained below. Write the program to perform the following tasks:

a) Define an <<abstract>> class named **Trip**: **(19 marks)**

- The class should provide a constructor with an argument that initializes the member attribute to the value passed as an argument. It should also create a member object of the **Vector** class.
- The **calcTax** method should calculate and return the tax amount by multiplying the passed amount with the tax rate.
- The **calcDisc** method should calculate and return the discount amount by multiplying the passed amount with the discount rate.
- The **calcTotalFare** method should calculate and return the total fare for all passengers.
- The **calcTotalFareWithTax** method should calculate and return the total fare including tax for all passengers.
- The **calcTotalFareWithDisc** method should calculate and return the total fare with discount for all passengers.
- The **addPassenger** method should add a passenger to the trip.

b) Define an <<interface>> named **TaxDiscCalculable**. The interface should have two final static variables and two abstract methods. **(3 marks)**

c) Define a class named **LocalTrip**: **(4 marks)**

- The class should provide a constructor with an argument that initializes the superclass's attribute.
- The **calcFare** method should calculate and return the fare for a local trip by multiplying the distance with the fare per kilometer, which is RM2.00 for local trips.

d) Define a class named **LongDistanceTrip**: **(4 marks)**

- The class should provide a constructor with an argument that initializes the superclass's attribute.
- The **calcFare** method should calculate and return the fare for a long-distance trip by multiplying the distance with the fare per kilometer, which is RM1.50 for long-distance trips.

```
1   //TripApp.java
2   import java.util.*;
3
4   //Fully given
5   class Passenger {
6     private String name;
7
8     public Passenger(String name) {
9       this.name = name;
10    }
11
12    public String getName() {
13      return name;
14    }
15  }
16
17  public class TripApp {
18    public static void main(String[] args) {
19      Vector <Passenger> list = new Vector <Passenger>();
20      Trip trip1 = new LocalTrip(10.5);
21      Trip trip2 = new LongDistanceTrip(50.0);
22
23      trip1.addPassenger(new Passenger("Alwi Ahmad"));
24      trip1.addPassenger(new Passenger("Ahmad Shariff"));
25      trip1.addPassenger(new Passenger("Rabiah Hakim"));
26
27      System.out.println("Information for Local Trip");
28      System.out.println("List of passengers:");
29      list = trip1.getPassengers();
30      for (int i = 0; i < list.size(); i++)
31        System.out.println((i+1) + ") " + list.get(i).getName());
32
33      System.out.printf("\nTotal Fare: RM%.2f\n", trip1.calcTotalFare());
34      System.out.printf("Total Fare with Tax: RM%.2f\n",
35        trip1.calcTotalFareWithTax());
36      System.out.printf("Total Fare with Discount: RM%.2f\n",
37        trip1.calcTotalFareWithDisc());
38
39      trip2.addPassenger(new Passenger("Halimah Abu"));
40      trip2.addPassenger(new Passenger("Rafidah Harun"));
41
```

```
42
43      System.out.println("\n\nInformation for Long-Distance Trip");
44      System.out.println("List of passengers:");
45      list = trip2.getPassengers();
46      for (int i = 0; i < list.size(); i++)
47        System.out.println((i+1) + ") " + list.get(i).getName());
48
49      System.out.printf("\nTotal Fare: RM%.2f\n", trip2.calcTotalFare());
50      System.out.printf("Total Fare with Tax: RM%.2f\n",
51        trip2.calcTotalFareWithTax());
52      System.out.printf("Total Fare with Discount: RM%.2f\n",
53        trip2.calcTotalFareWithDisc());
54    }
55  }
```

Figure B3: TripApp and **Passenger** class definitions