**Object Oriented Programming Methodology**

**(Model Solution – All programs can be written using other algorithm too. Given is only one way of writing program)**

**Total Marks : 80**

**(3 Hours)**

**N.B** 1) Question **no. 1 is compulsory.**

2) Attempt **any three from remaining** questions.

**Q. 1 a What is the need for constructor in a class? [10]**

**Develop a class circle with instance variable radius that is initialized using constructor. Create 2 methods in the class to calculate area and perimeter of circle.**

**5M – Need of constructor**

**5M – Program**

1) To initialize your object with default or initial state since default values for primitives may not be what you are looking for.

2) To inform the world about dependencies, a class needs to do its job.Anyone can figure out, what he needs in order to use this class by looking at the constructor.

**Program (One technique to write)**

**class Circle**

```
{
    double radius;
    double area;
    double perimeter;

    Circle(double radius)
    {
      this.radius = radius;
```

```
    CalculateArea()
    {
      area = (22*r*r)/7;
    }

    CalculatePerimeter()
    {
      perimeter = (22*2*r)/7;

    }
}
```

**b Explain static data members and methods in a class** **[5]**

**2.5M – static data members**

**2.5M – static methods**

Static data members, methods, and constants reside with a class and not instances of classes. They can be accessed from within the class defined or another class using the dot operator.

Static Data Members

Static data members have the same features as static methods, plus they are stored in a single location in memory.

They are used when only one copy of a data member is needed across all instances of a class (e.g., a counter).

```
// Declaring a static data member

public class Voter  {

  static int voterCount = 0;

  public Voter() { voterCount++;}

  public static int getVoterCount() {

    return voterCount;

  }

}

...

int numVoters = Voter.voterCount;
```

Static Methods

Static methods have the keyword static in the method declaration.

```
// Declaring a static method
```

```
class Analyzer {

  public static int getVotesByAge( ) {...}

}

// Using the static method

Analyzer.getVotesByAge();
```

Static methods cannot access non-static methods or variables because static methods are associated with a class, not an object.

**c** **Compare method overloading and overriding with an example each.** **[5]**

**3M- Explaination**

**2M - Example**

Overloading gives the ability to have multiple methods with the same name and same or different return type. The important thing about method overloading is that all this methods must have different parameters for the same return type. It is a common practice to leave the implementation for the method with the most parameters and the other methods (with fewer parameters) just redirect to the "big" method

```
public class OverloadingExample {

    static int sumOf(int a, int b) {

        return a+b;
```

```
}

static int sumOf(int a, int b, int c) {
        return a+b+c;
}

static double sumOf(double a, double b) {
        return a+b;
}

static double sumOf(double a, double b, double c) {
        return a+b+c;
}
public static void main(String[] args) {
        System.out.println(sumOf(1,2));
        System.out.println(sumOf(10d,20d,30d));
```

```
    }

}
```

Method overriding is to change the implementation of given method in a subclass. In other words you “override” the implementation of the parent’s class method using the same signature of the method (name, return types, parameters), but implement different functionality inside the overridden method.

```
public class Animal {

    public void makeSound() {

        System.out.println("the animal makes sounds");

    }

}

public class Dog extends Animal{

    @Override
```

```
    public void makeSound() {

        System.out.println("the dog barks");

    }

}
```

**Q. 2 a** Explain different types of relationships among entities. **[10]**

Define the relationships among the objects of given sentences:

1) Customer has Account.
2) CurrentAccount, SavingsAccount is a kind of Account.
3) Customer makes payment

**4M – Explaination**

**6M – Relationships**

Inheritance

It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.

In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. Composition and Aggregation are the two forms of association.

Aggregation is a special form of Association where:

It represents Has-A relationship.

It is a unidirectional association i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.

In Aggregation, both the entries can survive individually which means ending one entity will not effect the other entity

Eg. class Department has list of Students

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

It represents part-of relationship.

In composition, both the entities are dependent on each other.

When there is a composition between two entities, the composed object cannot exist without the other entity.

Eg. class Library contains list of Books

Aggregation relation is “has-a” and composition is “part-of” relation.

Composition is a strong Association whereas Aggregation is a weak Association.

**Type of relationship**

1)Composition

2)Inheritance

3)Association

**b What is a thread?Which are the two ways to create a thread? [10]**

**Write a program to show interleaving of actions from 2 thread: t1 and t2 synchronizing on a shared object.**

**t1 print message "ping" and t2 print message "pong".**

**4M - Theory**

**6M – Program**

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms :

1. Extending the Thread class

2. Implementing the Runnable Interface

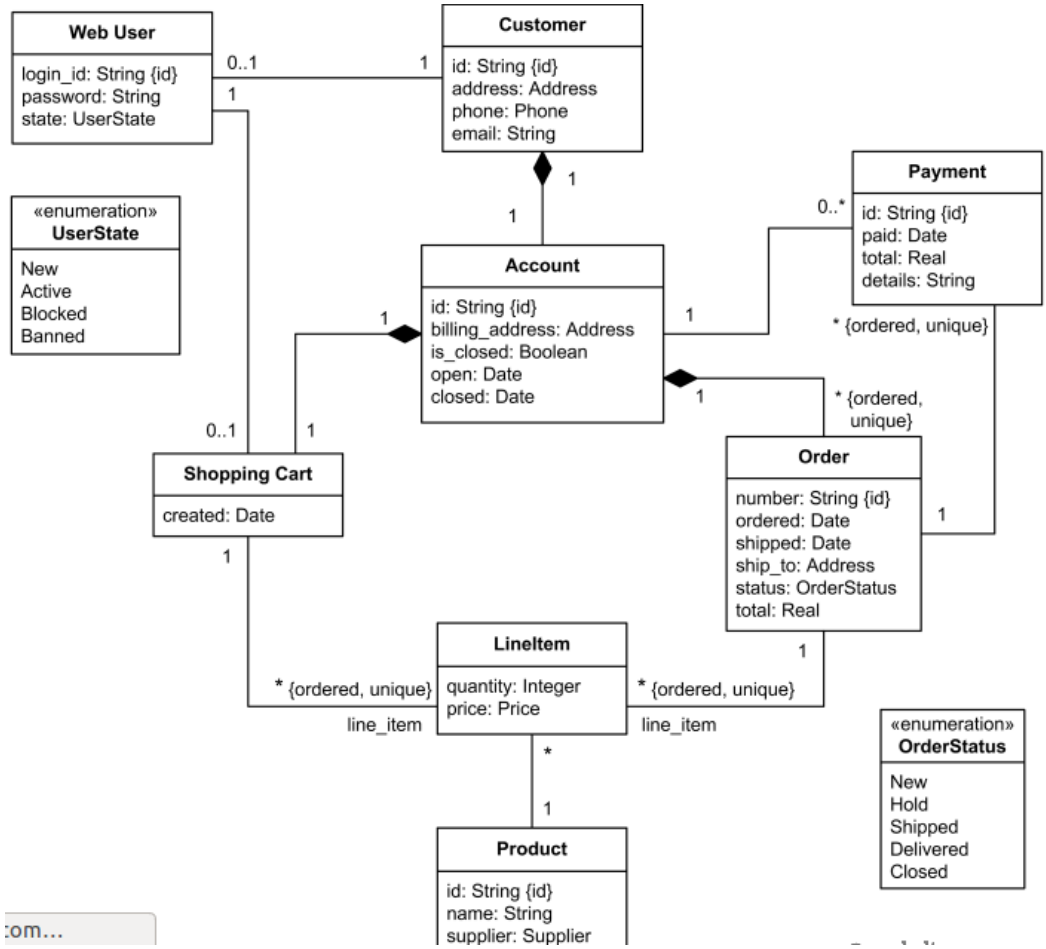
```
import java.io.*;

import java.util.*;
```

```
public class PingPong extends java.lang.Thread {

   private String word; // word to print

   private int delay;



   // constructor:

   public PingPong (String whatToSay, int delayTime) {

         word = whatToSay;

         delay = delayTime;

   }



   public void run() {

         try {

            while(true) {

                  System.out.print(word + " ");

                  Thread.sleep(delay); // class method
```

```
            }

        } catch (java.lang.InterruptedException ex) {return;}

    }


    public static void main (String args[]) {

        PingPong ping = new PingPong("ping", 300);

        PingPong pong = new PingPong("PONG", 1000);

        ping.start();

        pong.start();

    }

}
```

**Q. 3 a** **An online shopping application requires a customer to have an account. Each customer has unique id and is linked to exactly one account. Account owns shopping cart and orders. Customer has to register as a web user and can make only online purchases. Every user has a login name which is unique. User could have multiple states, new, active, temporary blocked or banned and is linked to shopping cart. Shopping cart belongs to account. Customer add products to shopping cart and then create order. Each order has order status. Both order and shopping cart have line items linked to a specific product. There is payment associated with every order.** **[10]**

**Draw class diagram for the given scenareo. Show the class attributes and methods and class relationships.**



**b Explain different types of coupling and cohesion [10]**

**5M – types of coupling**

**5M – types of cohesion**

1.Coincidentally cohesive: The subsystem in which the set of tasks are related with each other loosely then such subsystems are called coincidentally cohesive.

2. Logically cohesive: A subsystem that performs the tasks that are logically related with each other is called logically cohesive.

3. Temporal cohesive: The subsystem in which the tasks need to be executed in some specific time span is called temporal cohesive.

4. Procedural cohesive: When processing elements of a subsystem are related with one another and must be executed in some specific order, such subsystems is called Procedural cohesive.

5. Communication cohesion: when the processing elements of a subsystem share the data then such subsystem is called communication cohesive.

6. Sequential cohesion: when the output of 1 subsystem is given as input for other subsystem is called Sequential cohesion.

Coupling:

Coupling effectively represents how the subsystems can be connected with other subsystem or with the outside world.

Coupling is a measure of interconnection among subsystems in a program structure.

1. Data coupling: The data coupling is possible by parameter passing or data interaction.

2. Control coupling: The modules share related control data in control coupling.

3. Common coupling: In common coupling common data or global data is shared among the modules.

4. Content coupling: Content coupling occurs when one module makes use of data or control informatio

**Q. 4 a How does do-while construction differ from that of while loop? [10]**

**Write a program that has 2 methods. The first method reads a list of numbers terminated by -999 into an ArrayList. The second method displays the second largest value in the list.**

**4M - Difference**

**6M – Program**

| BASIS FOR COMPARISON | WHILE | DO-WHILE |
|---|---|---|
| General Form | while ( condition) {<br>statements; //body of loop<br>} | do{<br>.<br>statements; // body of loop.<br>.<br>} while( Condition ); |
| Controlling Condition | In 'while' loop the controlling condition appears at the start of the loop. | In 'do-while' loop the controlling condition appears at the end of the loop. |
| Iterations | The iterations do not occur if, the condition at the first iteration, appears false. | The iteration occurs at least once even if the condition is false at the first iteration. |

```
import java.util.ArrayList;

import java.util.Scanner;



public class Find

{

  ArrayList<Double> numbers = new ArrayList<Double>();




  private static readInput()

  {

    Scanner in = new Scanner(System.in);

    System.out.println("Please enter a list of numbers: ");



    while(true)

    {

    System.out.println("Next number");
```

```
    Double val = in.nextDouble();

    if(val == -999)

        break

    else

    numbers.add(val);



    }

 }



 private static void secondLargest()

{



    if (numbers.size() == 0)

    {

       System.out.println("List is empty.");
```

```
}



else

{

  double first, second;

     first = second = Integer.MIN_VALUE;

  for (double element : numbers)

  {

    total = total + element;

       if (element > first)

    {

       second = first;

       first = element;

    }

    else if (element > second && element != first)

    {

       second = element;
```

```
        }


    }


        System.out.println("The second largest value is: " + second);


    }
  }
}
```

**b** **What is checked and unchecked exception in Java? Explain the use of following in exception handling.** **[5]**

**Try-Catch, Finally, Throw, Throws**

**1M + 4M**

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle

the exception or it must specify the exception using throws keyword.

Unchecked are the exceptions that are not checked at compiled time.

Try-Catch

The try block contains set of statements where an exception can occur. A try block is always followed by a catch block,which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.

A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.

Finally

A finally block contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

Throw

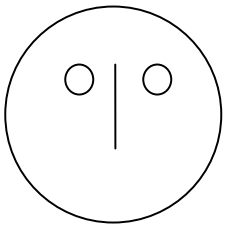
We can define our own set of conditions or rules and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException when we divide number by 5, or any other numbers, what we need to do is just set the condition and throw any exception using throw keyword. Throw keyword can also be used for throwing custom exceptions.

Throws

Suppose there are several methods that can cause exceptions, in that case it would be tedious to write try-catch for each method. The code will become unnecessary long and will be less-readable.

One way to overcome this problem is by using throws : declare the exceptions in the method signature using throws and handle the exceptions where you are calling this method by using try-catch.

**c Write an applet program to display** **[5]**



```
import java.applet.*;

import java.awt.*;
```

```
import java.awt.event.*;

/*<applet code="smile.class" height=200 width=200></applet>*/;

public class smile extends Applet

{

public void paint(Graphics g)

{

g.drawOval(250,250,300,300);

g.drawOval(310,320,30,30);

g.drawOval(440,320,30,30);

g.drawLine(390,350,390,390);

}

}
```

**Q. 5 a Explain creation of user defined package with an example. [10]**

**5M – Theory**

**5M – Program**

A package is a mechanism to group the similar type of classes, interfaces and sub-packages and provide access control. It organizes classes into single unit.

Vehicle.java

```
package vehicles;
```

```
interface Vehicle

{

    public void run();

    public void speed();

}
```

Car.java

```
package vehicles;

public class Car implements Vehicle

{

    public void run()

    {

        System.out.println("Car is running.");

    }

    public void speed()

    {

        System.out.println("Speed of Car: 50 Km/h");
```

```
}

public static void main(String args[])

{

    Car Car = new Car();

    Car.run();

    Car.speed();

    System.out.println("Hello World!");

}
```

**b** **Implement a class AnotherRectangle that extends Rectangle class and overrides the equals(...) method inherited from Object. Implement equals(...) so that 2 objects belonging to AnotherRectangle are equal if they agree in both length and width.** **[10]**

**Set length and width of rectangle using constructor.**

```
class Rectangle

{

  private Double length;

  private Double width;
```

```
  public Rectangle(Double length, Double breadth)
  {
    this.length = length;
    this.breadth = width;
  }
}

class AnotherRectangle extends Rectangle
{
//Overriding equals
  public boolean equals(Object obj)
  {
    if (this == obj) return true;
    if (obj == null) return false;
    if (this.getClass() != obj.getClass()) return false;
    AnotherRectangle that = (AnotherRectangle) obj;
```

```
    if (this.lenghth != that.length) return false;

    if (this.width != that.width) return false;

        return true;

  }

}
```

**Q. 6 a Differentiate between interface and abstract class.** **[10]**

| Interface | Abstract class |
|---|---|
| Interface support multiple inheritance | Abstract class does not support multiple inheritance |
| Interface does'n Contains Data Member | Abstract class contains Data Member |
| Interface does'n contains Cunstructors | Abstract class contains Cunstructors |
| An interface Contains only incomplete member (signature of member) | An abstract class Contains both incomplete (abstract) and complete member |
| An interface cannot have access modifiers by default everything is assumed as public | An abstract class can contain access modifiers for the subs, functions, properties |
| Member of interface can not be Static | Only Complete Member of abstract class can be Static |

**b** Write short note on access specifiers. **[5]**

| Access Modifiers | Default | private | protected | public |
|---|---|---|---|---|
| Accessible inside the class | yes | yes | yes | yes |
| Accessible within the subclass inside the same package | yes | no | yes | yes |
| Accessible outside the package | no | no | no | yes |
| Accessible within the subclass outside the package | no | no | yes | yes |

**c Explain "write once and run anywhere" nature of Java.** **[5]**

Write Once Run Anywhere is the feature applicable to those programs which hold the capability to execute itself on any operating systems or on any machine. According to this concept, the same code must run on any machine and hence the source code needs to be portable. So Java allows run Java bytecode on any machine irrespective of the machine or the hardware, using JVM (Java Virtual Machine). The bytecode generated by the compiler is not platform-specific and hence takes help of JVM to run on a wide range of machines. So we can call Java programs as a write once and run on any machine residing anywhere.