JAVA

Aim :Abstract class , Interface , Multithreading

1. Anchor College offers both graduate and postgraduate programs. The college stores the names of the students, their test scores and the final result for each student. Each student has to take 4 tests in total. You need to create an application for the college by implementing the classes based on the class diagram and description given below.

```
package practice;

import java.util.Scanner;

abstract class Student{
   String studentName;
   int[] testScores;
   String testResult;
   Student(){

   }
   Student(String studentName){
         this.studentName = studentName;
   }
   public String getStudentName() {
         return studentName;
   }
   public void setStudentName(String studentName) {
         this.studentName = studentName;
   }
   public int[] getTestScores() {
         return testScores;
   }
   public void setTestScore(int testNumber,int testScore) {
         testScores[testNumber] = testScore;
   }
   public void setTestScores(int[] testScores) {
         this.testScores = testScores;
   }
   public String getTestResult() {
         return testResult;
   }
   public void setTestResult(String testResult) {
```

```
        this.testResult = testResult;
    }
    abstract void generateResult();
}
class UndergraduateStudent extends Student{
    UndergraduateStudent(String studentName){
        super.studentName = studentName;
    }
    public void generateResult() {
        int sum = 0;
        for(int i:testScores) {
            sum+=i;
        }
        if((sum/testScores.length) >= 60) {
            System.out.println("Pass");
        }else {
            System.out.println("Fail");
        }
    }
}
class GraduateStudent extends Student{
    GraduateStudent(String studentName){
        super.studentName = studentName;
    }
    public void generateResult() {
        int sum = 0;
        for(int i:testScores) {
            sum+=i;
        }
        System.out.println("Name : "+getStudentName());
        System.out.println("Result : "+sum/testScores.length);
    }
}
public class program1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("1.Graduate \n 2.Undergraduate");
        int option = sc.nextInt();
        System.out.println("Enter name : ");
        String name = sc.next();
        int[] marks = new int[4];
        System.out.println("Enter Marks");
        for(int i=0;i<5;i++) {
            System.out.print("Marks of subject "+(i+1));
```

```
                marks[i] = sc.nextInt();
        }
        if(option == 1) {
                GraduateStudent g = new GraduateStudent(name);
                g.generateResult();
        }else {
                UndergraduateStudent ug = new UndergraduateStudent(name);
                ug.generateResult();
        }
   }
}
```

Output :

```
1.Graduate
2.Undergraduate
1
Enter name : OM
Enter Marks
Marks of subject 1 : 50
Marks of subject 2 : 60
Marks of subject 3 : 55
Marks of subject 4 : 60
Marks of subject 5 : 70
Name : OM
Result : 59
```

2. Write a Java program as per the given description to demonstrate use of interface.

```
interface RelationInterface {

        public boolean isGreater(Line line1, Line line2);
        public boolean isLess(Line line, Line line2);
        public boolean isEqual(Line line, Line line2);
}

class Line implements RelationInterface {

        double x1,x2,y1,y2;

        public Line(double x1, double x2, double y1, double y2) {
                this.x1 = x1;
```

```
		this.x2 = x2;
		this.y1 = y1;
		this.y2 = y2;
	}

	public double getLength() {
		double length = Math.sqrt((x2 - x1) * (x2 + x1) + (y2 - y1) * (y2 + y1));
		return length;
	}

	@Override
	public boolean isGreater(Line line, Line line2) {

		if(line.getLength() > line2.getLength()) {
			return true;
		} else {
			return false;
		}
	}

	@Override
	public boolean isLess(Line line, Line line2) {

		if(line.getLength() < line2.getLength()) {
			return true;
		} else {
			return false;
		}
	}

	@Override
	public boolean isEqual(Line line, Line line2) {

		if(line.getLength() == line2.getLength()) {
			return true;
		} else {
			return false;
		}
	}

}

class HelloWorld {

	public static void main(String[] args) {

		Line line1 = new Line(10.0, 20.0, 11.3, 13.8);
		Line line2 = new Line(10.0, 20.0, 11.3, 13.7);
```

```
                if(line1.isEqual(line1, line2)) {
                        System.out.println("Both are Equal");
                } else if (line1.isGreater(line1, line2)) {
                        System.out.println("Line 1 is Greater And Line 2 is less");
                } else {
                        System.out.println("Line 2 is Greater And Line 1 is less");
                }
        }
}
```

Output :

```
Line 1 is Greater And Line 2 is less
|
```

3. In the producer–consumer problem, the producer and the consumer share a common, fixed-size buffer used as a queue. (Take buffer size as 1). The produces's job is to generate data, put it into the buffer. At the same time, the consumer is consuming the data (i.e. removing it from the buffer).The problem is to make sure that the producer wont try to add data into the buffer if it;s full and that the consumer won't try to remove data from an empty buffer. Write a Java application consisting of all necessary classes to achieve this.

```
class Queue {

        int n;
        boolean valueSet = false;

        synchronized int get() {
        while (!valueSet)
        try {
                wait();
        } catch (InterruptedException e) {
                System.out.println("InterruptedException caught");
        }
        System.out.println("Got: " + n);
        valueSet = false;
        notify();
        return n;
        }

        synchronized void put(int n) {
        while (valueSet)
        try {
                wait();
        } catch (InterruptedException e) {
                System.out.println("InterruptedException caught");
```

```
        }
        this.n = n;
        valueSet = true;
        System.out.println("Put: " + n);
        notify();
        }
}

class Producer implements Runnable {

        Queue q;

        Producer(Queue q) {
        this.q = q;
        new Thread(this, "Producer").start();
        }

        public void run() {
        int i = 0;
        while (true) {
        q.put(i++);
        }
        }
}

class Consumer implements Runnable {

        Queue q;

        Consumer(Queue q) {
        this.q = q;
        new Thread(this, "Consumer").start();
        }

        public void run() {
        while (true) {
        q.get();
        }
        }
}

public class transmissionOfData {
        public static void main(String args[]) {
        Queue q = new Queue();
```

```
		new Producer(q);
		new Consumer(q);
		System.out.println("Terminate the process to stop.");
		}
}


class Queue {

		int n;
		boolean valueSet = false;

		synchronized int get() {
		while (!valueSet)
		try {
			wait();
		} catch (InterruptedException e) {
			System.out.println("InterruptedException caught");
		}
		System.out.println("Got: " + n);
		valueSet = false;
		notify();
		return n;
		}

		synchronized void put(int n) {
		while (valueSet)
		try {
			wait();
		} catch (InterruptedException e) {
			System.out.println("InterruptedException caught");
		}
		this.n = n;
		valueSet = true;
		System.out.println("Put: " + n);
		notify();
		}
}

class Producer implements Runnable {

		Queue q;

		Producer(Queue q) {
```

```
        this.q = q;
        new Thread(this, "Producer").start();
        }

        public void run() {
        int i = 0;
        while (true) {
        q.put(i++);
        }
        }
}

class Consumer implements Runnable {

        Queue q;

        Consumer(Queue q) {
        this.q = q;
        new Thread(this, "Consumer").start();
        }

        public void run() {
        while (true) {
        q.get();
        }
        }
}

public class transmissionOfData {
        public static void main(String args[]) {
        Queue q = new Queue();
        new Producer(q);
        new Consumer(q);
        }
}
```

Output :

```
Put: 0
Got: 0
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5
Got: 5
Put: 6
Got: 6
Put: 7
Got: 7
Put: 8
Got: 8
Put: 9
Got: 9
Put: 10
Got: 10
Put: 11
```

4. Write a multithreaded Java application to produce a deadlock condition.

```
package threads.interthreadcommunication;
class ThreadA {

    synchronized void raja(ThreadB b) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered ThreadA.raja");
```

```
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("ThreadA Interrupted");
        }
        System.out.println(name + " trying to call ThreadB.informer()");
        b.informer();
    }

    synchronized void informer() {
        System.out.println("Inside ThreadA.informer");
    }
}

class ThreadB {

    synchronized void baja(ThreadA a) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered ThreadB.baja");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("ThreadB Interrupted");
        }
        System.out.println(name + " trying to call ThreadA.informer()");
        a.informer();
    }

    synchronized void informer() {
        System.out.println("Inside ThreadB.informer");
    }
}

public class Deadlock implements Runnable {

    ThreadA a = new ThreadA();
    ThreadB b = new ThreadB();

    Deadlock() {
        Thread.currentThread().setName("OriginalThread");
```

```
        Thread t = new Thread(this, "SecondThread");
        t.start();
        a.raja(b); // get lock on a in this thread.
        System.out.println("Back in main thread");
    }

    @Override
    public void run() {
        b.baja(a); // get lock on b in other thread.
        System.out.println("Back in other thread");
    }

    public static void main(String args[]) {

        System.out.println("\nCLICK ON TERIMATE BUTTON OTHERWISE BOTH
PROCESS WILL CONTINUE INFINITE");
        new Deadlock();
    }
}
```

Output :

```
CLICK ON TERIMATE BUTTON OTHERWISE BOTH PROCESS WILL CONTINUE INFINITE
OriginalThread entered ThreadA.raja
SecondThread entered ThreadB.baja
SecondThread trying to call ThreadA.informer()
OriginalThread trying to call ThreadB.informer()
```