Laboratory Work

Subject: Java Technologies

Branch: B.Tech. (CE)

Semester: IV

Batch: A1

Student Roll No: CE012

Student Name: Gohel Akshay Rajendrakumar

Department of Computer Engineering,

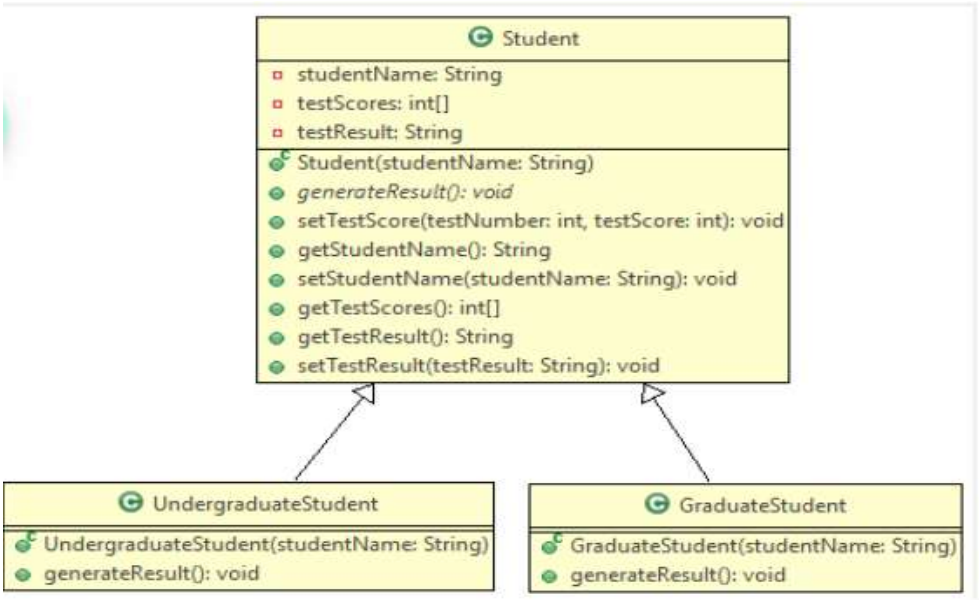
Faculty of Technology,

Dharmsinh Desai University, Nadiad – 387001.

Gujarat, INDIA.

Topics :- Abstract class , Interface , Multithreading

1. Anchor College offers both UnderGraduate 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.



Method Description

Implement the getter and setter methods appropriately.

1) Student(Class)

I. Student(String studentName)

● Initialize the instance variable studentName with the value passed to the constructor and other instance variables to the default values.

II. setTestScore(int testNumber, int testScore)

● Set the value of the testScore in the appropriate position of testScores array based on the testNumber.

2) UndergraduateStudent(Class)

I. UndergraduateStudent(String studentName)

● Initialize the instance variable studentName with the value passed to the constructor and other instance variables to the default values.

II. generateResult()

● Implement the abstract method of Student class by setting the value of testResult based on the below details.

Average Score	Result
>=60	Pass
<60	Fail

Sample Input and Output For UndergraduateStudent

Input:-

Instance variable	Values
Name	Jerry
testScore	{70,69,71,55}

Output:-

Student Name : Jerry

Result : Pass

3) PostGraduateStudent(Class)

I. PostgraduateStudent(String studentName)

● Initialize the instance variable studentName with the value passed to the constructor and other instance variables to the default values.

II. generateResult()

● Implement the abstract method of Student class by setting the value of testResult based on the below details.

Average Score	Result
>=75	Pass
<75	Fail

Sample Input and Output For PostUndergraduateStudent

Input:-

Instance variable	Values
Name	Tom
testScore	{70,75,80,85}

Output:- Student Name : Tom

Result : Pass

CODE:-

Student.java
```
package lab5ce012;
```

```
abstract public class Student {
	private String studentName;
	private int[] testScore=new int[4];
	private String testResult;
	public Student(String studentName) {
		super();
		this.studentName = studentName;
	}
	abstract void generateResult();
	public String getStudentName() {
		return studentName;
	}
	public void setStudentName(String studentName) {
		this.studentName = studentName;
	}
	public void setTestScore(int testNumber,int testScore) {
		this.testScore[testNumber]=testScore;
	}
	public int[] getTestScore() {
		return testScore;
	}
	public String getTestResult() {
		return testResult;
	}
	public void setTestResult(String testResult) {
		this.testResult = testResult;
	}
}
```

UnderGraduate.java

```
package lab5ce012;
import java.util.*;
public class UnderGraduateStudent extends Student{
	public UnderGraduateStudent(String name) {
		super(name);
	}
	public void generateResult(){
		double avg=0;
		for(int a:this.getTestScore()) {
			avg+=(double)a;
		}
		avg=avg/4;
		if(avg>=60) {
			super.setTestResult("Pass");
		}
		else {
			super.setTestResult("Fail");
		}
	}
	public static void main(String[] arg) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter your name");
		String name;
		name=sc.next();
		UnderGraduateStudent s=new UnderGraduateStudent(name);
		System.out.println("Enter Score");
```

```
		int i=0;
		while(i<4) {
			int x=sc.nextInt();
			s.setTestScore(i++, x);
		}
		System.out.println("Student name "+s.getStudentName());
		s.generateResult();
		System.out.println("Result "+ s.getTestResult());
	}
}
```

OUTPUT:-

```
Enter your name
Niraj
Enter Score
90
78
68
89
Student name Niraj
Result Pass
```

PostGraduate.java

```
package lab5ce012;
import java.util.*;
public class PostGraduateStudent extends Student {
	public PostGraduateStudent(String name) {
		super(name);
	}
	public void generateResult(){
		double avg=0;
		for(int a:this.getTestScore()) {
			avg+=(double)a;
		}
		avg=avg/4;
		if(avg>=65) {
			super.setTestResult("Pass");
		}
		else {
			super.setTestResult("Fail");
		}
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter your name");
		String name;
		name=sc.next();
		PostGraduateStudent s=new PostGraduateStudent(name);
		System.out.println("Enter Score");
		int i=0;
		while(i<4) {
			int x=sc.nextInt();
			s.setTestScore(i++, x);
```

```
            }
            System.out.println("Student name "+s.getStudentName());
            s.generateResult();

            System.out.println("Result "+ s.getTestResult());

            sc.close();

      }

}
```

OUTPUT:-

```
Enter your name
niraj
Enter Score
98
75
95
45
Student name niraj
Result Pass
```

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

I. Define an interface RelationInterface. Write three abstract methods: isGreater, isLess and isEqual. All methods have a return type of boolean and take an argument of type Line with which the caller object will be compared.

CODE:-

```
package lab5ce012;

public interface RelationInterface {
      boolean isGreater(Line l);
      boolean isEqual(Line l);
      boolean isLess(Line l);
}
```

II. Define the Line class implements the RelationInterface interface.

- It has 4 double variables for the x and y coordinates of the line.
- Define a constructor in Line class that initializes these 4 variables.
- Define a method getLength() that computes length of the line.

  [double length = Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))].

- Implement the methods of interface in Line class

CODE:-

```
package lab5ce012;
import java.lang.Math;
public class Line implements RelationInterface {
```

```
	private int x1,x2,y1,y2;

	public Line(int x1, int x2, int y1, int y2) {
		super();
		this.x1 = x1;
		this.x2 = x2;
		this.y1 = y1;
		this.y2 = y2;
	}
	public double getLength() {
		return Math.sqrt(Math.pow(x2-x1, 2)+Math.pow(y2-y1,2));
	}
	public boolean isEqual(Line l) {
		return l.getLength()==this.getLength();
	}
	public boolean isLess(Line l) {
		return l.getLength()<this.getLength();
	}
	public boolean isGreater(Line l) {
		return l.getLength()>this.getLength();
	}
}
```

III. In class CompareLines.Java, create two objects of Line class, call the three methods to compare the lengths of the lines.

CODE:-

```
package lab5ce012;

public class CompareLines {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Line l1=new Line(2,4,5,6);
		Line l2=new Line(1,3,4,5);
		System.out.println(l1.isEqual(l2));
		System.out.println(l1.isLess(l2));
		System.out.println(l1.isGreater(l2));
	}

}
```

OUTPUT:-

```
true
false
false
```

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 producer'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 won't 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.

Code:-

```
package lab5ce012;


//A correct implementation of a producer and consumer.
class Q1 {

 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 Producer1 implements Runnable {

 Q1 q;

 Producer1(Q1 q) {
     this.q = q;
     new Thread(this, "Producer").start();
 }

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

```
class Consumer1 implements Runnable {

 Q1 q;

 Consumer1(Q1 q) {
     this.q = q;
     new Thread(this, "Consumer").start();
 }

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

public class ProducerConsumer {

 public static void main(String args[]) {
     Q1 q = new Q1();
     new Producer1(q);
     new Consumer1(q);
     System.out.println("Terminate the process to stop.");
 }
}
```

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
.

.

.
```

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

CODE:-

```
package lab5ce012;

class A {
```

```
    synchronized void foo(B b) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered A.foo");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("A Interrupted");
        }
        System.out.println(name + " trying to call B.last()");
        b.last();
    }

    synchronized void last() {
        System.out.println("Inside A.last");
    }
}

class B {

    synchronized void bar(A a) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered B.bar");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("B Interrupted");
        }
        System.out.println(name + " trying to call A.last()");
        a.last();
    }

    synchronized void last() {
        System.out.println("Inside B.last");
    }
}

public class DeadLock implements Runnable {

    A a = new A();
    B b = new B();

    DeadLock() {
        Thread.currentThread().setName("MainThread");
        Thread t = new Thread(this, "RacingThread");
        t.start();
        a.foo(b); // get lock on a in this thread.
        System.out.println("Back in main thread");
    }

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

    public static void main(String args[]) {
        new DeadLock();
        System.out.println("Thread completed");
```

```
    }
}
```

OUTPUT:-

```
RacingThread entered B.bar
MainThread entered A.foo
RacingThread trying to call A.last()
MainThread trying to call B.last()
```