

Laboratory Work

Subject: Java Technologies

Branch: B.Tech. (CE)

Semester: IV

Batch: A1

Student Roll No: CE001

Student Name: NISARG KALPESHBHAI AMLANI

Department of Computer
Faculty of Technology,
Dharmsinh Desai University,



Gujarat, INDIA.

Engineering,
Nadiad – 387001.

Lab 05

Question 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

Input:-

Instance Variable Values

name Jerry

testScores {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

testScores {70,75,80,85}

Output:- Student Name : Tom

Result : Pass

Solution)

– > Student Class

```
import java.util.Arrays;

public abstract class student {
    String studentName ;
    int [] testScores = new int[4];
    String testResult ;

    student(String studentName)
    {
        this.studentName = studentName;
    }

    public String getTestResult() {
        return testResult;
    }

    public void setTestResult(String testResult) {
        this.testResult = testResult;
    }

    abstract void generateResult();

    public void setTestScores(int testScore, int testnumber) {
        this.testScores[testnumber] = testScore;
    }

    public int[] getTestScores() {
        return testScores;
    }
}
```

```

    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }

    @Override
    public String toString() {
        return
            "studentName=" + studentName + '\n' +
            "testResult=" + testResult
            ;
    }
}

```

– > Undergraduate class

```

import java.util.Scanner;
public class undergraduate extends student
{
    public undergraduate(String studentName) {
        super(studentName);
    }

    @Override
    void generateResult() {
        String s ;
    }
}

```

```

for(int it : testScores)
{
    if( it < 60)
    {
        s = "Fail";
        super.setTestResult(s);
        return ;
    }
    s = "Pass";
    super.setTestResult(s);
}
}

```

```

class Main{
    public static void main(String[] args) {
        Scanner obj = new Scanner(System.in);
        System.out.print("Enter student name :- ");
        undergraduate s1 = new undergraduate(obj.nextLine());
        for (int i = 0;i<4;i++)
        {
            System.out.print("Enter result of test " + (i+1) + " :- " );
            s1.setTestScores(obj.nextInt(),i);
        }
        s1.generateResult();
        System.out.println(s1);
    }
}

```

- > Screenshot for Undergraduate class

```
/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/ja
.local/share/JetBrains/Toolbox/apps/intellij
Enter student name :- Nisarg
Enter result of test 1 :- 56
Enter result of test 2 :- 78
Enter result of test 3 :- 89
Enter result of test 4 :- 78
studentName=Nisarg
testResult=Fail

Process finished with exit code 0
```

- > postgraduate Class

```
import java.util.Scanner;
public class postgrad extends student {
    public postgrad(String studentName) {
        super(studentName);
    }

    @Override
    void generateResult() {
        for(int it : super.testScores)
        {
            if(it < 75)
            {
                super.setTestResult("Fail");
                return ;
            }
        }
    }
}
```

```

    }
}
super.setTestResult("Pass");
}
}

class Main1{
    public static void main(String[] args) {
        Scanner obj = new Scanner(System.in);
        System.out.print("Enter student name :- ");
        postgrad s1 = new postgrad(obj.nextLine());
        for (int i = 0;i<4;i++)
        {
            System.out.print("Enter result of test " + (i+1) + " :- " );
            s1.setTestScores(obj.nextInt(),i);
        }
        s1.generateResult();
        System.out.println(s1);
    }
}

```

Screenshot for Postgraduate Class

```

/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/java -jav
.local/share/JetBrains/Toolbox/apps/intellij-idea-u
Enter student name :- nisarg
Enter result of test 1 :- 75
Enter result of test 2 :- 89
Enter result of test 3 :- 78
Enter result of test 4 :- 78
studentName=nisarg
testResult=Pass

Process finished with exit code 0
|

```


Question 2)

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.

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

III. In class CompareLines.Java, create two objects of Line class, call the three

methods to compare the lengths of the lines.

Solution

– > Relation Interface

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

- > Line Class

```
public class Line implements RelationInterface{
    double x1 ,x2 ,y1,y2;

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

- > Main Method

```
import java.util.Scanner;

public class CompareLine {
```

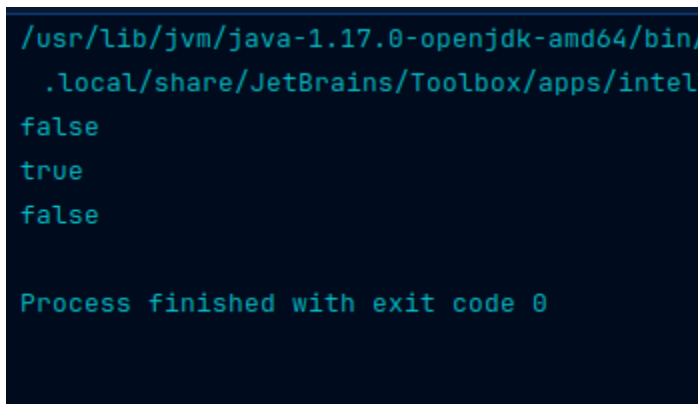
```

public static void main(String[] args) {
    Scanner obj = new Scanner(System.in);
    Line l1 = new Line(5,4,3,1);
    Line l2 = new Line(8,2,0,1);

    System.out.println(l1.isEqual(l2));
    System.out.println(l1.isLess(l2));
    System.out.println(l1.isGreater(l2));
}
}

```

Screenshot



```

/usr/lib/jvm/java-1.17.0-openjdk-amd64/bin/
.local/share/JetBrains/Toolbox/apps/intel
false
true
false

Process finished with exit code 0

```

Question 3)

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 producers’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.

Solution)

```
class Buffer {
    private int data;
    private boolean isEmpty = true;

    public synchronized void put(int value) {
        while (!isEmpty) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        this.data = value;
        this.isEmpty = false;
        notifyAll();
    }

    public synchronized int get() {
        while (isEmpty) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        this.isEmpty = true;
        notifyAll();
        return this.data;
    }
}
```

```

    }
}
class Producer extends Thread {
    private final Buffer buffer;

    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            buffer.put(i);
            System.out.println("Produced: " + i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) {}
        }
    }
}

```

```

class Consumer extends Thread {
    private final Buffer buffer;

    public Consumer(Buffer buffer) {
        this.buffer = buffer;
    }

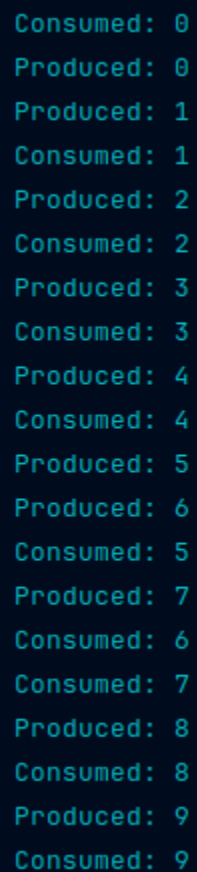
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("Consumed: " + buffer.get());
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) {}
        }
    }
}

```

```
}  
}  
}
```

```
class Main3 {  
    public static void main(String[] args) {  
        Buffer buffer = new Buffer();  
        Producer producer = new Producer(buffer);  
        Consumer consumer = new Consumer(buffer);  
  
        producer.start();  
        consumer.start();  
    }  
}
```

Screenshot)



```
Consumed: 0  
Produced: 0  
Produced: 1  
Consumed: 1  
Produced: 2  
Consumed: 2  
Produced: 3  
Consumed: 3  
Produced: 4  
Consumed: 4  
Produced: 5  
Produced: 6  
Consumed: 5  
Produced: 7  
Consumed: 6  
Consumed: 7  
Produced: 8  
Consumed: 8  
Produced: 9  
Consumed: 9
```

Question 4)

Write a multithreaded Java application to produce a deadlock condition.

Solution)

```
class Deadlock {
    public static void main(String[] args) {
        final Object resource1 = "resource1";
        final Object resource2 = "resource2";

        Thread thread1 = new Thread(() -> {
            synchronized (resource1) {
                System.out.println("Thread 1: locked resource 1");

                try { Thread.sleep(100);} catch (Exception e) {}

                synchronized (resource2) {
                    System.out.println("Thread 1: locked resource 2");
                }
            }
        });

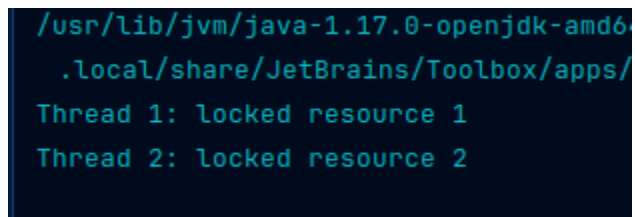
        Thread thread2 = new Thread(() -> {
            synchronized (resource2) {
                System.out.println("Thread 2: locked resource 2");

                try { Thread.sleep(100);} catch (Exception e) {}

                synchronized (resource1) {
                    System.out.println("Thread 2: locked resource 1");
                }
            }
        });
```

```
    });  
  
    thread1.start();  
    thread2.start();  
}  
}
```

Screenshot)



A screenshot of a terminal window with a dark blue background and light blue text. The text shows the Java installation path, the application path, and the output of two threads locking resources.

```
/usr/lib/jvm/java-1.17.0-openjdk-amd64  
    .local/share/JetBrains/Toolbox/apps/  
Thread 1: locked resource 1  
Thread 2: locked resource 2
```