LAB 5

NAME : DARJI NILESH MAHESHKUMAR
ROLL NO : CE025

AIM:Abstract class , Interface , Multithreading

1

```
package src.mypkg;

public abstract class Student {
    private String studentName;
    private int[] testScores ;
    private String testResult;

    public String getStudentName() {
     return studentName;
    }
    public void setStudentName(String studentName) {
     this.studentName = studentName;
    }
    public int[] getTestScores() {
     return testScores;
    }
    public void setTestScores(int[] testScores) {

     this.testScores = testScores;
    }
    public String getTestResult() {
     return testResult;
    }
    public void setTestResult(String testResult) {
     this.testResult = testResult;
    }
    public Student(String studentName) {
     this.studentName = studentName;
    }
    public abstract void generateResult();

}

class UndergraduateStudent extends Student {
```

```
    public UndergraduateStudent(String studentName) {
     super(studentName);
    }

    @Override
    public void generateResult() {
     int[] arr = getTestScores();
     int sum= 0;
     for(int a:arr) {
         sum+=a;
     }
     int avg = sum / arr.length;
     if(avg >= 60) {
         setTestResult("Pass");
     }else {
         setTestResult("Fail");
     }
    }

}

class GraduateStudent extends Student {

    @Override
    public void generateResult() {
     int[] arr = getTestScores();
     int sum= 0;
     for(int a:arr) {
         sum+=a;
     }
     int avg = sum / arr.length;
     if(avg >= 60) {
         setTestResult("Pass");
     }else {
         setTestResult("Fail");
     }
    }

    public GraduateStudent(String studentName) {
     super(studentName);
    }
}
```

```
Name : Jerry
Result : Pass
```

2

```java
package src.mypkg;

interface RelationInterface {
    public boolean isGreater(Line x );
    public boolean isLess(Line x );
    public boolean isEqual(Line x );
}
class Line implements RelationInterface {
    private double x1,y1,x2,y2;
    public Line(double x1, double y1, double x2, double y2) {
     this.x1 = x1;
     this.y1 = y1;
     this.x2 = x2;
     this.y2 = y2;
    }
    @Override
    public boolean isGreater(Line x) {
     return this.getLength() > x.getLength();
    }
    @Override
    public boolean isLess(Line x) {
     return this.getLength() < x.getLength();
    }
    @Override
    public boolean isEqual(Line x) {
     return this.getLength() == x.getLength();
    }
    public double getLength() {
     double length = Math.sqrt(((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)));
     return length;
    }
}
```

```
public class CompareLines {
    public static void main(String[] args) {
     Line l1 = new Line(1, 2,3, 4);
     Line l2 = new Line(1, 2,3, 4);
     if(l1.isEqual(l2)) {
         System.out.println("\nl1 is Equals to l2");
     }else if(l1.isLess(l2)){
         System.out.println("l1 is Less then l2");
     }else {
         System.out.println("l1 is Greater then l2");
     }
    }
}
```

```
l1 is Equals to l2
```

3

```
package src.mypkg;

class A {
    synchronized void foo(B b) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " : A.foo");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("A Interrupted");
        }
        System.out.println(name + " : 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 + " : B.bar");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("B Interrupted");
        }
        System.out.println(name + " : 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();
    }
}
```

```
MainThread : A.foo
RacingThread : B.bar
RacingThread : call A.last()
MainThread : call B.last()
```

4

```
package src.mypkg;

import java.util.LinkedList;
import java.util.Queue;
class Consumer implements Runnable {
    LinkedList<Integer> q;
    public Consumer(LinkedList<Integer> q) {
     this.q = q;
     new Thread(this,"Consumer").start();
    }
    @Override
    public void run() {
     while (true) {
         try {
             if (q.size() <= 0) {

                 Thread.sleep(200);

             } else {
                 q.pop();
                 System.out.println("Poped   " + q.size());
```

```
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
     }
   }
}

class Producer implements Runnable {
    static int max = 5;
    LinkedList<Integer> q;

    public Producer(LinkedList<Integer> q) {
     this.q = q;
     new Thread(this,"Producer").start();
    }
    @Override
    public void run() {
     while(true) {
         try {
         if(q.size() >= max) {
             Thread.sleep(200);
         }else {
             q.push(1);
             System.out.println("Pushed  "+ q.size());
             Thread.sleep(100);
         }
         } catch (InterruptedException e) {
             e.printStackTrace();
         }
      }
    }
}

public class Test {
    static LinkedList<Integer> q = new LinkedList<>();
```

```
    public static void main(String[] args) {
     Producer p = new Producer(q);
     Consumer c = new Consumer(q);

    }
}
```

```
Poped  1
Poped  0
Pushed  1
Pushed  1
Poped  1
Poped  0
Pushed  1
Poped  0
Pushed  1
Pushed  2
Poped  1
Poped  0
Pushed  1
Pushed  2
Poped  1
Poped  0
Pushed  1
Pushed  2
Poped  1
Poped  0
Pushed  1
```