OOP Assignment III

1. Write an application that displays a series of at least five student ID numbers (that you have stored in an array) and asks the user to enter a numeric test score for the student. Create a ScoreException class, and throw a ScoreException for the class if the user does not enter a valid score (zero to 100). Catch the ScoreException and then display an appropriate message. In addition, store a 0 for the student's score. At the end of the application, display all the student IDs and scores.

```
//
//   ScoreTest.java
//
//   Created by Avikant Saini on 9/29/15
//

import java.util.*;

class Student {

    String id;
    int score;

    public Student (String id) {
      this.id = id;
      score = 0;
    }
}

class ScoreException extends Exception {
    ScoreException (String errorMessage) {
      super(errorMessage);
    }
}

public class ScoreTest {

    public static void main (String [] args) {
      Scanner sc = new Scanner(System.in);

      Student students[] = new Student[3];

      for (int i = 0; i < 3; ++i) {

            String randomUUID =
UUID.randomUUID().toString().replaceAll("-", "").substring(0, 10);
// Random String for the glory of Satan...
```

```
            students[i] = new Student(randomUUID); // Create a new instance

            System.out.print("\n\tStudent " + (i+1) + ", id: " +
students[i].id + "\n\tEnter score (0-100): ");

            int score = 0;

            try {
                  score = sc.nextInt(); // Input the score
                  if (score < 0 || score > 100) {
                        score = 0;
                        throw new ScoreException("Invalid Score. Using
default 0 value."); // Throw the exception and set value to 0
                  }
            }

            catch (ScoreException exception) {
                  System.err.println("\n\tError: " + exception);
            }

            finally {
                  students[i].score = score;
// Finally set the score value to the inputted value or 0
            }

      }

      System.out.println("\n\tID\t\tScore");
      for (Student stud : students)
            System.out.println("\t" + stud.id + "\t" + stud.score);

    }
}
```

2. Write and execute a java program to create five threads, the first thread checking the uniqueness of matrix elements, the second calculating row sum, the third calculating the column sum, the fourth calculating principal diagonal sum, the fifth calculating the secondary diagonal sum of a given matrix. The main thread reads a square matrix from keyboard and will display whether the given matrix is magic square or not by obtaining the required data from sub threads.

```
//
// MagicSquare.java
//
// Created by Avikant Saini on 9/29/15
//

import java.util.*;

class MagicSquare implements Runnable {

    int mat[][];      // To store the matrix
    int sum[];        // To store the sum : rows + columns + diagonals
    boolean isUnique;

    public MagicSquare (int n) {
      mat = new int[n][n];
      sum = new int[n + n + 2];
      isUnique = false;
    }

    public void input () {
      Scanner sc = new Scanner(System.in);
      int n = mat.length;
         for(int i = 0; i < n; ++i) {
             for(int j = 0; j < n; ++j) {
                 System.out.print("\tEnter matrix[" + (i+1) + "][" + (j
+1) + "] : ");
                 mat[i][j] = sc.nextInt();
             }
         }
    }
```

```
public void run () {
  // Get the current thread's name
  String threadName = Thread.currentThread().getName();

  int n = mat.length;

  // Checking for uniqueness
  if (threadName.equals("Uniqe")) {
        isUnique = true;
        int temp[] = new int[n * n];
        int i, j, k = 0;
        for (i = 0; i < n; ++i)
              for (j = 0; j < n; ++j)
                    temp[k++] = mat[i][j];
        for (i = 0; i < n * n; ++i)
              for (j = i + 1; j < n; ++j)
                    if (temp[j] == temp[i])
                          isUnique = false;
  }
  // Storing the row sum
  else if (threadName.equals("RSum")) {
        for (int i = 0; i < n; ++i) {
              int rt = 0;
              for (int j = 0; j < n; ++j)
                  rt += mat[i][j];
              sum[i] = rt;
        }
  }
  // Storing the column sum
  else if (threadName.equals("CSum")) {
        for (int i = 0; i < n; ++i) {
              int ct = 0;
              for (int j = 0; j < n; ++j)
                  ct +=  mat[j][i];
              sum[i + n] = ct;
        }
  }
  // Storing the principal diagonal sum
  else if (threadName.equals("PDSum")) {
        for (int i = 0; i < n; ++i)
              sum[n + n] += mat[i][i];
  }
  // Storing the secondary diagonal sum
  else {
        for (int i = 0; i < n; ++i)
              sum[n + n + 1] += mat[i][i];
  }
}
```

```
    public boolean isMagicSquare () {
      int n = mat.length;
      if (isUnique) {
            for (int i = 1; i < n + n + 2; ++i)
                  if (sum[i] != sum[0])
                        return false;
            return true;
      }
      return false;
    }
}

class MagicTest {
    public static void main (String [] args) {
      // Take the matrix...
      Scanner sc = new Scanner(System.in);
      System.out.print("\n\tEnter the number of elements in the matrix:
");
      int n = sc.nextInt();
      MagicSquare msq = new MagicSquare(n);
      msq.input();

      // Create five named threads...
      String threadNames[] = {"Uniqe", "RSum", "CSum", "PDSum",
"SDSum"};
      Thread threads[] = new Thread[5];
      for (int i = 0; i < 5; ++i) {
            // Label the threads and start them.
            threads[i] = new Thread(msq, threadNames[i]);
            threads[i].start();
      }
      // Wait for each thread to finish...
      for (Thread thrd : threads) {
            try {
                  thrd.join();
            }
            catch (Exception exception) {
                  System.err.println("Error in threads: " + exception);
            }
      }
      // Now call the isMagicSquare Method
      if (msq.isMagicSquare())
            System.out.println("\n\tMatrix is a magic square!\n\n");
      else
            System.out.println("\n\tMatrix is not a magic square!\n
\n");

    }
}
```

3. Create a class StudentThread. Create n threads in it (n = total number of semesters). Each thread finds the Student with the highest CGPA in a particular semester. The main thread finds the student with the highest CGPA among all semesters.

```
//
// StudentThreadTest.java
//
// Created by Avikant Saini on 9/29/15
//

import java.util.*;

class StudentThread implements Runnable {

    double cgpa[][];  // Row: 'n' semesters, Coln: CGPA for 'm' students in a sem
    double max[];     // Maximum in a semster for 'n' semesters

    public StudentThread (int n) {
      cgpa = new double[n][];
      max = new double[n];
    }

    public void input () {
      Scanner sc = new Scanner(System.in);
      for (int i = 0; i < cgpa.length; ++i) {
            System.out.print("\n\tEnter number of students in semester
\'" + (i+1) + "\': ");
            int m = sc.nextInt();
            cgpa[i] = new double[m];
            System.out.print("\tEnter CGPA of " + m + " students in
semester " + (i+1) + ": ");
            for (int j = 0; j < m; ++j)
                  cgpa[i][j] = sc.nextDouble();
      }
    }

    public void run () {
      // Get the thread number that we put in main as string
      int tno = Integer.parseInt(Thread.currentThread().getName());

      // Calculate max for that semester and put that in the max array
      max[tno] = 0.0;
      for (int i = 0; i < cgpa[tno].length; ++i)
            if (cgpa[tno][i] > max[tno])
                  max[tno] = cgpa[tno][i];
    }
```

```
    public double getAllMax () {
      double allMax = 0;
      for (double i : max)
            if (i > allMax)
                  allMax = i;
      return allMax;
    }

}

public class StudentThreadTest {

    public static void main (String [] args) {

      Scanner sc = new Scanner(System.in);

      System.out.print("\n\tEnter the number of semesters: ");
      int n = sc.nextInt();

      StudentThread sthrd = new StudentThread(n);
      sthrd.input();

      // Create 'n' threads
      Thread threads[] = new Thread[n];

      for (int i = 0; i < n; ++i) {
            // Set title of the thread to string equivalent of 'i', we'll retrieve it
later, and start the thread.
            threads[i] = new Thread(sthrd, ""+i);
            threads[i].start();
      }

      for (Thread thrd : threads) {
            try {
                  thrd.join();
            }
            catch (Exception exception) {
                  System.err.println("Error in threads: " + exception);
            }
      }

      System.out.println("\n\tAll time best GPA: " +
sthrd.getAllMax());
    }
}
```

4. Design a stack class. Provide your own stack exceptions namely push exception and pop exception, which throw exceptions when the stack is full and when the stack is empty respectively. Show the usage of these exceptions in handling a stack object in the main.

```
//
// StackTest.java
//
// Created by Avikant Saini on 9/29/15
//

import java.util.*;

class PushException extends Exception {
    PushException (String errorMessage) {
      super(errorMessage);
    }
}

class PopException extends Exception {
    PopException (String errorMessage) {
      super(errorMessage);
    }
}

class Stack {
    int tos;
    int arr[];

    public Stack (int n) {
      tos = -1;
      arr = new int[n];
    }


    public void push (int item) {
      try {
            if (tos == arr.length - 1)
                  throw new PushException("Stack Overflow!");
            else
                  arr[++tos] = item;
      }
      catch (PushException exception) {
            System.err.println("Error: " + exception);
      }
    }
```

```
    public int pop () {
      int item = 0;
      try {
            if (tos == -1) {
                  item = -32767;
                  throw new PopException("Stack Underflow!");
            }
            else
                  item = arr[tos--];
      }
      catch (PopException exception) {
            System.err.println("Error: " + exception);
      }
      finally {
            return item;
      }
    }

    @Override
    public String toString () {
      String stck = "";
      for (int i = 0; i <= tos; ++i)
            stck += "\t" + arr[i];
      return stck;
    }
}

public class StackTest {

    public static void main (String [] args) {

      Scanner sc = new Scanner(System.in);

      System.out.print("\tEnter the number of elements in the stack:
");
      int n = sc.nextInt();
      Stack stack = new Stack(n);

      int choice;
      do {
            System.out.print("\n\t1. Push\n\t2. Pop\n\t3. Display\n
\tChoice: ");
            choice = sc.nextInt();

            if (choice == 1) {
                  System.out.print("\n\tEnter item to push: ");
                  int item = sc.nextInt();
                  stack.push(item);
            }
```

```
            else if (choice == 2) {
                  int item = stack.pop();
                  if (item != -32767)
                        System.out.print("\n\tPopped item: " + item);
            }

            System.out.println("\n\tCurrent Stack: " + stack);

      } while (choice == 1 || choice == 2 || choice == 3);

    }
}
```

5. Create an interface called Player. The interface has an abstract method called play() that displays a message describing the meaning of "play" to the class. Create classes called Child, Musician, and Actor that all implement Player. Create an application that demonstrates the use of the classes (UsePlayer.java). Save the files as Player.java and keep inside package p1; Child.java , Actor.java, Musician.java and keep inside package p2, and UsePlayer.java in package p3. Show all necessary steps to compile and execute the application.

```
// ~/p1/Player.java

package p1;

public interface Player {
    void play ();
}


// ~/p2/Child.java
package p2;

import p1.Player;

public class Child implements Player {
    @Override
    public void play () {
      System.out.println("Child plays with Lego.");
    }
}
```

```java
// ~/p2/Musician.java

package p2;

import p1.Player;

public class Musician implements Player {
    @Override
    public void play () {
      System.out.println("Musician plays a piano.");
    }
}


// ~/p2/Actor.java

package p2;

import p1.Player;

public class Actor implements Player {
    @Override
    public void play () {
      System.out.println("Actor plays in a film.");
    }
}

// ~/p3/Player.java

package p3;

import p1.Player;
import p2.*;

public class UsePlayer {
    public static void main (String [] args) {
      Player player;

      player = new Child();
      player.play();

      player = new Musician();
      player.play();

      player = new Actor();
      player.play();
    }
}
```

```
// Compilation steps

 #ROOT ~> cd ~/Player

 #ROOT Player> javac p1/*
 #ROOT Player> javac p2/*
 #ROOT Player> javac p3/*

 #ROOT Player> java p3.UsePlayer

Child plays with Lego.
Musician plays a piano.
Actor plays in a film.

 #ROOT Player>

File Paths:

      ~/Player/p1/Player.java

      ~/Player/p2/Child.java

      ~/Player/p2/Musician.java

      ~/Player/p2/Actor.java

      ~/Player/p3/UsePlayer.java
```