Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

University of Ottawa
Faculty of engineering

School of Electrical Engineering
and Computer Science

Introduction to Computing II (ITI 1121)
MIDTERM EXAMINATION

Instructors: Opeyemi Adesina, Sherif Aly, Guy-Vincent Jourdan and Marcel Turcotte

March 2017, duration: 2 hours

Identification

Last name: ______________________ First name: ______________________________

Student #: ___________ Seat #: ____ Signature: _________________ Section: A or B or C or D

Instructions

1. This is a closed book examination.
2. No calculators, electronic devices or other aids are permitted.
   (a) Any electronic device or tool must be shut off, stored and out of reach.
   (b) Anyone who fails to comply with these regulations may be charged with academic fraud.
3. Write your answers in the space provided.
   (a) Use the back of pages if necessary.
   (b) You may not hand in additional pages.
4. Do not remove pages or the staple holding the examination pages together.
5. Write comments and assumptions to get partial marks.
6. Beware, poor hand-writing can affect grades.
7. Wait for the start of the examination.

Marking scheme

Question	Maximum	Result
1	25	
2	15	
3	30	
Total	**70**	

All rights reserved. No part of this document may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior written permission from the instructors.

Question 1 (25 marks)

This question is about Java basics. You are required to write 3 classes. First, you will write the class **Point**, which simply represents a point on a two-dimensional plane. Objects of this class have two **instance** variables, **x** and **y**, which record the coordinate of the point. The **constructor** of the class receives these 2 elements of information as parameters.

In the space below, provide the code for the class **Point**, including one **constructor**, all the necessary **getter** and **setter** methods, as well as an implementation of the instance method **toString**.

You now create a class **ColoredPoint**, which is a **specialized** version of the class **Point**. In addition to the coordinates, an instance of **ColoredPoint** has also an **instance** variable **c**, which is the colour of the point (**c** is a reference to an instance of a class **Color**, defined elsewhere in the program). The **constructor** of the class receives the 2 coordinates and the colour as parameters.

In the space below, provide the code for the class **ColoredPoint**, including the **constructor**, all the necessary **getters** and **setters**, as well as an implementation of the method **toString**.

Lastly, create a class **BoundingBox**. Its constructor receives, as a parameter, the reference of an array containing **a mix** of **Point** and **ColoredPoint** instances, and it computes a bounding box made of the top left and bottom right points of the smallest rectangle that contains all the points in the array.

If the constructor is passed the reference of an array that has a single element, then the bounding box is that element. If there is no element, or if the reference is **null**, then the bounding box is the point (0,0).

In the space below, provide the code for the class **BoundingBox**, including the **constructor** (which must compute the bounding box), as well as the **getters** for the top left and bottom right points, as well as an implementation of the method **toString**.

Question 2 (15 marks)

For this question, an index (a number) is associated with an element of information. You will provide two solutions to this problem, without and with the use of generics.

In the space below, write the implementation of the class **Indexed**. An object of the class **Indexed** has two instance variables, **index** of type **int** and **value** of type **String**. Give the implementation of a constructor having two parameters, which are the initial values for the instance variables. Implement getter methods for these two attributes, but no setters. Give the implementation of the instance method **isEqual** that receives the reference of another **Indexed** object and returns **true** if and only if this and the other object have the same content. This implementation should not make use of generics.

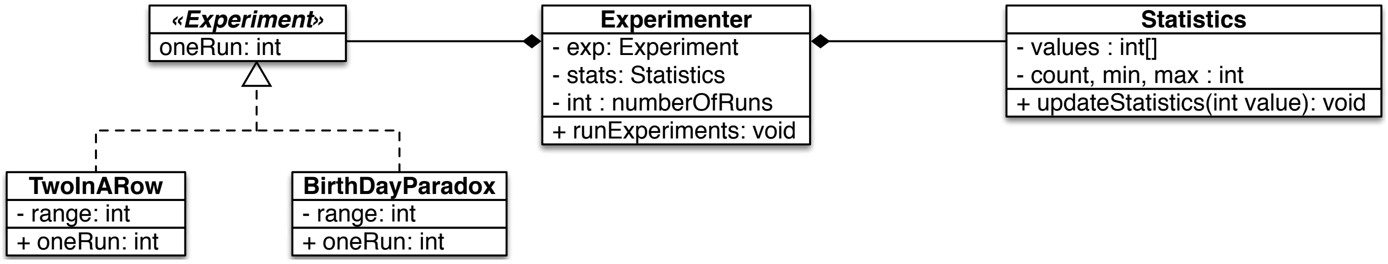
In the space below, declare a reference variable, create an object of the class **Indexed** to store the values 0 and "Ottawa". Finally, store the reference of that object in the reference variable.

In the space below, write the implementation of the class **Indexed** that takes a type parameter, **T**. The class **Indexed** declares two instance variables, **index** of type **int** and **value** of type **T**. Give the implementation of a constructor having two parameters, which are the initial values for the instance variables. Implement getter methods for these two attributes, but no setters. Give the implementation of the instance method **isEqual** that receives the reference of another **Indexed** object and returns **true** if and only if this and the other object have the same content.

In the space below, declare a reference variable, create an object of the class **Indexed** to store the values 0 and "Ottawa". Finally, store the reference of that object in the reference variable.

Question 3 (30 marks)

This question takes advantage object oriented and interface concepts to generalize the tasks of assignment 2. The UML diagram below presents the organization of the classes and the interface.



The Java source code below shows the intended use of these objects.

```
TwoInARow t;
Experimenter e;

t = new TwoInARow(10);
e = new Experimenter(t, 100);

e.runExperiments();
```

The execution of the above program should produce the following output on the console.

```
We have run 100 experiments:
 the minimum was 3
 the maximum was 41
 the mean was 10.62
 the standard deviation was 7.86
```

A. Implement the interface **Experiment**. The interface declares a method **oneRun** with no parameter, the return value of the method is of type **int**.

B. Implement the class **TwoInARow**.

- The class **TwoInARow** implements the interface **Experiment**.
- Its constructor has a parameter specifying the **range** of values for this experiment.
- The method **oneRun** generates random numbers in a specified range of values. It stops once two **consecutive** values are equal. Finally, it returns the number of attempts that were necessary for finding two consecutive values that are equal. In the example below, the method **oneRun** needed 8 attempts to produce two consecutive numbers that were equal. The return value is 8.

```
4, 7, 2, 9, 7, 5, 1, 1
```

```
import java.util.Random;
```

Hint: an object of the class **Random** has a method **nextInt** that returns a random number between 0 (inclusive) and **n** (exclusive), where **n** is the parameter of the method.

C. Implement the class **Experimenter**.

- The constructor of the class **Experimenter** has two parameters. The first parameter is the reference of an object whose class implements the interface **Experiment**. The second parameter is the specified number of runs.
- The method **runExperiments** will run the experiment a specified number of times, collect the results in an object of the class **Statistics** (see next page), and display statistics at the end of the runs.

D. On the next page, modify the class **Statistics** to use the dynamic array technique presented in class. In the implementation on the right write only the portions of the code that need to change. The other portions are assumed to remain the same.

```
public class Statistics {

    private int[] values;
    private int count, min, max;

    public Statistics(int numberOfRuns) {
        values = new int[numberOfRuns];
        count = 0;
    }

    public void updateStatistics(int value) {
        if (count == 0) {
            min = max = value;
        }
        min = Math.min(min, value);
        max = Math.max(max, value);
        values[count] = value;
        count = count + 1;
    }

    public int getMin() { return min; }

    public int getMax() { return max; }

    public double average() {
        double result = 0.0;
        for (int i = 0; i < count; i++) {
            result = result + values[i];
        }
        return result / count;
    }

    public double standardDeviation() {
        double mean = average();
        double squareSum = 0;
        for (int i = 0; i < count; i++) {
            squareSum += Math.pow(values[i] - mean, 2);
        }
        return Math.sqrt((squareSum) / count);
    }

}
```

```
public class Statistics { // Only the necessary changes

}
```