Tutorial 2

Objectives

Basic object use in Java (String, Random, and Point objects).

Attendance Quiz

Please log on to cuLearn using one of the computers in the tutorial room and complete the attendance quiz. You can only access the quiz if you log in using one of the computers in the lab. You cannot use a laptop for this. This is a time limited quiz. Be sure to do this as soon as you arrive.

At the end of the tutorial a TA will assign you a grade, call it G, which is 0, 1 or 2, depending on the progress you make during the tutorial. If you spend your time reading mail and chatting with your friends you will receive 0. If you have completed the attendance quiz on time then G will be your tutorial grade. If you have not completed the attendance quiz on time, then your tutorial grade will be max(0, G - 1/2). Each tutorial grade will therefore be one of 0, 0.5, 1.5 or 2.

In order to receive full marks for this tutorial, you must fully complete parts 1 and 2, and make good progress into part 3.

---

0) Create a folder/directory called **comp1406t2** and download/copy the tutorial files into this folder/directory. Code for this tutorial belongs to the **comp1406t2** package so it will need to be in this folder/directory.

1) In the **Tutorial02** program, modify the static method called **strings** that has the following modifiers, return type and signature (name and input argument types)

```
public static void strings()
```

The method will access the (static) class attribute **words**. The method will access each string in the words array and display to the terminal (System.out) the following:

```
Cat, is 3 letters long.
Dog, is 3 letters long.
...
Kit, it 3 letters long.

The longest word is KITTEN.
There were 39 characters in total.
```

For each string, print the **capitalized** version of the word, followed by "is X letters long", where X is the length of the

string. After processing all the words, print out the string that has the longest length (in **full caps**) and then the total of all characters in all the words.

You can find details of the behaviour of string objects (methods) in the **API** for the String class

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

2) The **Random** class allows us to generate pseudorandom numbers (and other things). A sequence of pseudorandom random numbers *looks* like random sequence of numbers but is actually computed with a deterministic algorithm using a **seed**. If you use the same seed you get the exact same sequence of numbers. If the seed is chosen randomly then a good pseudorandom number generator will be indistinguishable from a truly random number generator. For our purposes, we won't worry about truly random numbers. (If you are securing a website you would want to use truly random numbers.)

Modify the method called

```
public static void rollRandomDice(long seed)
```

in the **Tutorial02** program. Be sure to **import** the Random class. You do this by adding the line

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

between your package line and your class declaration line in the Tutorial02.java file.

Your method will create two **Random** objects. You will create your dice (random objects) as follows:

```
Random die1 = new Random();
Random die2 = new Random();
```

Each random object will be used to simulate a fair die (6-sided die). Using the **nextInt** method, you will *roll* each die and add up their face values. Consult the **API** to see how to use the method to get values from 1 to 6 (inclusive).

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html

The distribution of values (if each die is fair) is given at the end of this document. You can also see the distribution at

https://mathinsight.org/media/image/image/two_dice_distribution.png

You Roll the dice 10000 times and record how many times each total (sum of face values) is rolled.

Hint: Use an array of size 11 (since there are 11 possible outcomes ranging from 2 to 12) to record the frequency of each rolled total (sum of the two dice).

Note that when you called the **rollRandomDice** method, you need to pass a seed value. It is not used yet so just pass any value you want.

Print, to **System.out**, the frequency distribution you get after all the dice rolls.

Question: Is it as expected (i.e., is it the same as the plot shown in the link above)?

Now go back and after you create each die, set the **seed** for each using

```
die1.setSeed(seed);
die2.setSeed(seed);
```

Here **seed** is the value passed as input to the method. What happened to your distribution? What happens if you change this to

```
die1.setSeed(seed);
die2.setSeed(seed+1);
```

Can you explain the differences?

Note: In practice, pseudorandom numbers are *usually* good enough. Obtaining **truly random** numbers is expensive (in time and effort). We obtain truly random numbers by observing nature: measuring radioactive decay, observing microscopic temperature fluctuations of the cpu or other various electromagnetic/quantum phenomena. It is not always possible to generate enough truly random numbers for our needs. Only some applications require truly random numbers though (like when securing a bank's website). Using a pseudorandom sequence with a single truly random seed os often good enough. For us, we usually won't even worry the seed.

3) Modify the **Point2D** class to add some behaviour. In particular, modify the (NON-static) methods

```
public double magnitude()
public double distance(Point2D other)
public String toString()
```

The **magnitude** returns the distance from that point to the origin (i.e., the point (0,0)). The **distance** method returns the distance between the point and the input point. The **toString** method returns a string that represents the point. For example, if you have the code fragment

```
Point2D p = new Point2d();
p.x = 1.2;
p.y = 2.6;
System.out.print( "magnitude of point " );
System.out.print( p.toString() );
System.out.println( " is " + p.magnitude() );
```

it should output something like

```
magnitude of point (1.2, 2.6) is 2.86356
```

The return value of the toString() method here was the string **"(1.2, 2.6)"**. Your method should return this string.

Note: the distance between points (a,b) and (c,d) is sqrt((a-c)*(a-c) + (b-d)*(b-d)). Use the Math class OR your Babylonian method to compute square roots.

In the **Tutorial02** program, modify

```
public static void testPoint2D()
```

that tests your Point2D class. It should generate several points, compute the lengths of them and the distance between some of them. Be sure to print useful information in this testing. For example, include what the input is, what the output is and what the expected output it.

```
The chance of rolling a total of 2 is 2.78%
The chance of rolling a total of 3 is 5.56%
The chance of rolling a total of 4 is 8.33%
The chance of rolling a total of 5 is 11.11%
The chance of rolling a total of 6 is 13.89%
The chance of rolling a total of 7 is 16.67%
The chance of rolling a total of 8 is 13.89%
The chance of rolling a total of 9 is 11.11%
The chance of rolling a total of 10 is 8.33%
The chance of rolling a total of 11 is 5.56%
The chance of rolling a total of 12 is 2.78%
```