COMP1006/1406 - Summer 2016 - Tutorial 2

Objectives: Practice using classes, using objects and writing constructors, getters and setters.

1: Basic Math

The `Math` class in Java provides lots of static methods that we can use to compute mathematical functions. The class is found in the `java.lang` package. This is an important package that contains many basic classes that are used very often. Because the classes in the package are used so much they are, by default, always visible to us when we write a Java program.

For example, when using the Math class, we can simply access one of its static methods using `Math.tan(0.3)`. If the `java.lang` package was not directly visible to us, we would have to use `java.lang.Math.tan(0.3)`, which specifies exactly where to find the `Math` class.

➨ Write a Java class called `Trig` with just a main method. Your class should verify the trigonometric identity $\cos^2(x) + \sin^2(x) = 1$, for many different values of x. In particular, print out the value of $\cos^2(x) + \sin^2(x)$ and the absolute difference between this number and 1 for values of x starting from 0.2 and incrementing by 0.0051 until you reach 0.4. Use `double`s for this.

Does your experiment validate the trigonometric identity? Can you explain the output of your program?

☞ The `String` and `System` classes are also found in the `java.lang` package.

2: Flipping a Coin

Programs often need some randomization. A simple way of generating pseudorandom numbers is with the `Random` class in Java. A pseudorandom sequence of numbers looks like a random sequence of numbers but it is actually computed using a deterministic algorithm. For our purposes, we can consider it as random.

Unlike the Math class, you will need to create an instance of the Random class (i.e., create an object of type Random) to use it. The methods in the class are not static.

The Random class is part of the `java.util` package. Unlike java.lang, we do not by default see this package in our programs. We have two alternatives in this situation to use the Random class.

1. We can write out exactly where the Random class is located when using it. To declare and instantiate (create) a Random object we would use

$$\underbrace{\texttt{java.util.Random}}_{\text{type}}\ \underbrace{\texttt{randomObject}}_{\text{variable name}} = \texttt{new}\ \underbrace{\texttt{java.util.Random();}}_{\text{constructor}}$$

This is probably not what you will do in practice. But, you can always access a class by giving the full path to find it like this.

2. We can `import` the `Random` class so that it is directly visible to us. To do this, we add the following line at the start of our `.java` file (outside of the class definition)

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

   The `Random` class is now visible to your class (program). You can now declare and instantiate Random objects with

   $$\underbrace{\texttt{Random}}_{\text{type}}\ \underbrace{\texttt{randomObject}}_{\text{variable name}} = \texttt{new}\ \underbrace{\texttt{Random();}}_{\text{constructor}}$$

   We could also import the entire `java.util` package (`import java.util.*;`), but this is not considered good programming practice. (Can you think of any reasons why this might not be a good idea?)

Once you have a `Random` object, you can then generate pseudorandom data (`int`s, `double`s, `boolean`s, etc) by calling the appropriate method of that object. For example, the following will simulate flipping a fair coin (even chance of getting a head or tail) 20 times using random booleans (true for heads and false for tails).

```
Random coinFlip = new Random();
boolean coin;
for(int i=0; i<20; i+=1){
  coin = coinFlip.nextBoolean();
  if(coin==true){
    System.out.println("heads");
  }else{
    System.out.println("tails");
  }
}
```

There are several useful methods in the `Random` class. Look at the API for the `Random` class. `http://docs.oracle.com/javase/8/docs/api/java/util/Random.html`

➨ Write a Java program called `Flip`. Your class only needs a main method and will use one command line argument. In your main method, create a `Random` object and use it to generate pseudorandom numbers (`double`s) to simulate a coin flip. The command line argument (a number between 0 and 1) will specify the bias of your flip towards "heads". For example, `java Flip 0.5` will simulate a fair coin which returns heads 50% of the time and tails 50%

of the time, while `java Flip 0.3` will simulate an unfair coin that returns a heads 30% of the time and tails 70% of the time.

Your program should generate 100 pseudorandom numbers (coin flips) and count how many times heads and how many times tails was found. The program should output something like `28 heads, 72 tails, bias 0.3` when running `java Flip 0.3`.

The `nextDouble()` method in the `Random` class returns a number in the range [0.0, 1.0). If we assume all numbers in this range are equally likely to be returned, then we can simulate the coin flip by comparing the output with the probability. For example, suppose r is the random number returned and p is the probability we want. If $r < p$ then we assign the flip to heads and otherwise it is tails. The larger p is the more likely we get a head.

Are the results of your program expected? Try running it several times with the same and different input biases.

➨ Modify your program to accept two command line arguments: the first being the bias of the coin and the second being the number of coin flips you use to create your statistics. (The original program used 100 coin flips.) Try running your program with 10 flips, 100 flips, 1000 flips and 100000 flips. What do you notice?

☞ Random numbers have numerous uses in computer science. From simply adding variation to video games, to creating "random" cryptographic keys, to scientific computations using the Monte Carlo method, to sorting. Often, pseudorandom numbers, like the ones obtained from the Random class are good enough. In some situations, like generating secret cryptographic keys, true random numbers are needed (but these are costly to obtain).

3: Constructors

➨ Download the `Money.java` file from the tutorial website. (Note that is NOT the same money class from Assignment 2.) This is a simple class that stores money as dollars and cents. For example, $12.73 will be stored as 12 dollars and 73 cents. The cents value should never be greater than 99, so 3 dollars and 164 cents should actually be stored as 4 dollars and 64 cents. The class has only one method, `getMoney()`, which returns a String representation of the money object. Your task is to create three constructors for the class as follows:

```
public Money(){...}
   // creates a Money object with zero money

public Money(int dollars, int cents){...}
   // creates a Money object with value as specified by the input values
   // input values are assumed to satisfy dollars >= 0 and cents >= 0.

public Money(int cents){...}
   // creates a Money object with value as specified by the input cents
   // input value is assumed to satisfy cents >= 0
```

Download the testing program TestMoney.java from the tutorial website and use this to test your constructors.

➨ Add the following instance methods to your Money class:

```
public void add(int c){...}
 // adds c cents to the current value

public void add(int d, int c){...}
 // adds d dollars and c cents to the current value

public int remove(int c){...}
 // removes c cents from current value if current value is large enough
 // otherwise, removes as much as it can
 // returns the actual amount of cents removed (may be > 100)
```

Be sure to test your methods. Pay special attention to remove method. As with the constructors, the intention is that your internal representation of the money will satisfy the condition `0 <= cents <= 99`. Adjust your dollars and cents so that this is always maintained.

Play Computer

The code will not be working Java code. It might not be any computer language at all. However, you should still be able to read and understand what the code is doing (or trying to do).

```
function helper(int n)
----------------------------------------
   if  n < 10  then return 3
   if  n < 100 then return 2
   return 1
----------------------------------------

function mystery(int n)
----------------------------------------
   integer x = n

   while( x < 200*n )
      x = x + (x * helper(x))

   return helper(x/10) + helper(n) + n
----------------------------------------
```

What is printed when this function is called with input 14? That is, what should be displayed if `mystery(14)` is called? (Trace thorough this code with pencil and paper to determine what the output is.)

Extra Coding

➨ Write a program that takes 3 command line arguments (n, m and p). Your program should create a 2-dimensional array (n X m) of `char`s. Populate your array with either 'x' or 'o', where an 'x' is placed with probability p and an 'o' is placed otherwise.

For example, running your program with command line arguments `10 10 0.5` will create an array (10 rows and 10 columns) where each element is an 'x' with probability 1/2.

Your program should create the array and then display it nicely to the screen.

➨ Write a text-based version of the minesweeper game.

http://en.wikipedia.org/wiki/Minesweeper_(video_game)
http://minesweeperonline.com/