Tutorial 3

Objectives

Practice creating and using constructors. Generate random numbers.

Attendance Quiz

Please log on to cuLearn using on of the computer 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.

Play Computer [10 minutes]

```
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.)

Constructors

Modify the *Money* class that is provided. 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. However, the cents value stored 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 current state of a given money object . Your first task is to create three constructors for the class as follows:

```
public Money(){...}
```

```
public Money(){...}
   // creates a Money object with zero money
   
public Money(int dollars, int cents){...}
   // creates a Money object with total 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
```

The *Money* class has a *getMoney()* method to help test/debug your code. It returns a String representation of the money.

Use the testing program *TestMoney.java* from the tutorial repository to help test your constructors.

Next, 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 the *remove* method. As with the constructors, the intention is that your internal representation of the money will satisfy the condition *0 <= cents <= 99* at all times. Adjust your dollars and cents so that this is always maintained.

Flipping a coin

Programs often need some "randomization". We will consider two ways to easily generate a pseudorandom number. 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 good enough to be random.

Math.random()

Math.random() will return a pseudorandom double, call it X, that satisfies *0.0 <= X < 1.0*.

Using the output of this static method, we can simulate die rolls, coin flips, etc.

For example, to simulate the roll of a die, we can generate random integers in the range [1,6] with

```
1 + (int) (Math.random()*6)
```

Why does this never generate 7? The *Math.random()*6* will generate a random number, call it X, that satisfies *0.0 <= X < 6.0*. When we cast this to an integer, Java simply throws away the decimal part, so we only get the integers 0,1,2,3,4 and 5. The final output, based on the value of X will be

```
0.0 <= X < 1.0   -->   1
1.0 <= X < 2.0   -->   2
2.0 <= X < 3.0   -->   3
3.0 <= X < 4.0   -->   4
4.0 <= X < 5.0   -->   5
5.0 <= X < 6.0   -->   6
```

The Random Class

The *Random* class provides methods for generating random sequences (integers, floating point numbers, booleans). 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.

To simulate the roll of a die (an integer in the range [1,6]) we would use

```
java.util.Random die = new Random();
int die_roll = 1 + die.nextInt(6);
```

See the API for the class to see what other methods the class provides.
https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Code

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 (of type *double*) 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, running the program from the command line (or from the interactions window in DrJava) with the single command line argument 0.5

```
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) (just like Math.random()). 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 bias. For example, suppose r is the random number returned and p is the probability we want for heads. 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.

More

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?

Command Line Arguments

A program in Java must have the method

```
public static void main(String[] args){...}
```

which is the entry point to your program. Notice that the method has an input parameter called args. We can specify the value of the argument passed into the main method when we run our program using a command line. If our class was called *CommandLineArguments*, then

```
java CommandLineArguments one 2 three
```

will the JVM (Java virtual machine) will run the main method of the *CommandLineArguments* class with the input argument *["one", "2", "three"]*.

We can then use this data inside the *main* method. The *CommandLineArguments* program that is provided gives a small example of a program that uses command line arguments. Compile this and try running with different command line arguments.