

# Tutorial 3

## W22 – COMP1006B/1406C – Introduction to Computer Science II

---

### Objectives

Basic object creation (constructors and instance methods). Using Java's Random class.

### Notes

Question 1 is asking you to add some constructors to a simple class.

Questions 2 is asking you to add some behavior to the same class.

Question 3? In this question, you are simulating the act of rolling a pair of dice. You are asked to roll a pair of dice many times and keep track of each result using a 2-d array. From this 2-d array, you can then determine the distribution of the total value of the dice rolls (that is, the sum of the dice in a given roll). You need to use a Random object for each die.

Question 4 is optional extra practice. It is asking you to process an array of strings: printing something on the screen for each string (capitalized version of the string, the length of the string), keeping track of the total number of characters in all the strings seen and the longest string.

---

**QUESTION 1**

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. The cents value stored should never be greater than 99, so 3 dollars and 164 cents should be stored as 4 dollars and 64 cents.

The class has only one method, **toString()**, which returns a String representation of the money object. Your first 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 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

// optional extra constructor if you are looking for more practice
// public Money(int[] coins){...}
//     // creates a Money object with value as specified by the input array.
//     // The array will have six (6) elements, corresponding to the coins
//     // toonies, loonies, quarters, dimes, nickels, pennies
//     // ($2, $1, $0.25, $0.10, $0.05, $0.01)
```

In all the constructors, be sure that the internal state (dollars and cents) represents the total money and that cents is not greater than 99.

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

**QUESTION 2**

Add some behaviour to the Money class. Add the following two methods:

```
public Money add(int d, int c)
    // adds d dollars and c cents to the current money object
    // (like the constructors, the internal state of the object
    // will always have 0 <= cents <= 99)
    // pre-conditions: d >= 0 and c >= 0.
    // return a reference to the current object

public Boolean remove(int d, int c)
    // removed d dollars and c cents from the current money object if possible
    // pre-conditions: d >= 0 and c >= 0.
    // returns true if d dollars and c cents was removed, false otherwise
```

## QUESTION 3

The **Random** class allows us to generate pseudorandom numbers (and other things). A sequence of pseudorandom random numbers *looks* like a random sequence of numbers but is actually computed with a deterministic algorithm using a **seed** (which sets the initial state). If you use the same seed when you start your sequence, 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

```
public static int[][] rollRandomDice(int n, long seed)
```

in the **Tutorial03** program. Be sure to **import** the Random class. You do this by adding the line

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

at the top of your Tutorial03.java file.

Your method will create two **Random** objects to simulate the dice. You will need two variables as follows:

```
Random d1 = new Random();
Random d2 = new Random();
```

Here d1 and d2 are the simulated fair (6-sided) dice. Using the **nextInt()** method, you will *roll* each die. Consult the **API** for the **Random** class to see how to use the method to get values from 1 to 6 (inclusive).

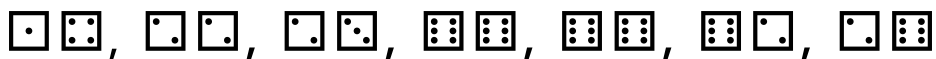
<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Random.html>

Your method will simulate a total of **n** rolls of the dice. The results of the rolls will be stored in a (6 × 6) two-dimensional array of integers (ints)

```
int[][] rolls = new int[6][6];
```

For each roll of the pair of dice, increment ONE element in the array that corresponds to value the dice. For example, if d1 rolled a 3 and d2 rolled a 6, then you would increment the array element `rolls[2][5]` (where the first index corresponds to the value of d1 and the second to the value of d2; remember that index values start with zero so we need to shift by one).

For example, suppose we roll the dice 7 times as follows (each roll showing d1 d2 in that order):



1-4,      2-2,      2-3,      6-6,      6-6,      6-2,      2-6

The `rolls` array would look like

```
[ [0, 0, 0, 1, 0, 0],
  [0, 1, 1, 0, 0, 1],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0],
  [0, 1, 0, 0, 0, 2] ]
```

Notice that the last two rolls each affect a different element in `rolls` even though the total of both are the same. Your method will return this `rolls` array after the dice are rolled a total of `n` times.

Next, add a method

```
public static void diceRollResults(int[][] rolls)
```

that takes a 2-d array of integers (as output by the method above) and prints the following to the terminal (standard output):

- The input array (you can use `Arrays.deepToString()` to help)
- The number of dice rolls for the input array (you need to compute this)
- The distribution of the combined *totals* for the rolls. Each time the dice are rolled, their values sum to one of 2,3,...,12. You are asked to find the fraction of rolls that result in 2, and 3, and 4, etc. (It is here that the total of the d1 and d2 are used.)

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

[https://mathinsight.org/media/image/image/two\\_dice\\_distribution.png](https://mathinsight.org/media/image/image/two_dice_distribution.png)

Run your methods several times with different values of `n`. In particular, use `n = 36`, `n=360`, and `n=36000`. For now, use any value for the input **seed** for this. (The value is not actually used in the method yet.)

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 securing internet connections).

## Distribution of fair dice rolls

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%

---

**QUESTION 4 [OPTIONAL FOR EXTRA PRACTICE]**

In the **Tutorial03** 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/17/docs/api/java.base/java/lang/String.html>

Note that Java’s String class does NOT have a capitalize() method like Python’s str class does. You’ll have to build up this string by using the upper-case version of the first character followed the by lower-case version of the rest of the original string.