# Lab 3: Using Classes and Objects

## Welcome back!

Now that we've installed Java and Eclipse and familiarized ourselves with our development environment, we can focus on learning how to program. Today we'd like you to:

1. **Open Eclipse**. It's a good idea to keep all your projects in the same workspace. Think of a workspace as a shelf, and each project is a binder of stuff on the shelf.
2. **Create a new Java project**. Call it Comp1510Lab03*LastNameFirstInitial*. For example, if your name were Fred Bloggs, you would name the project **Comp1510Lab03BloggsF**.
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 3 project to quickly create a new package or Java class.
4. When you have completed the exercises, **hand them in** or **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. Although you may cut and paste from the lab document, it will help you learn if you **type in the code by hand**. The tactile effort of typing in the code will help reinforce the Java syntax.

## What will you DO in this lab?

In this lab, you will:

1. Instantiate objects and invoke their methods using the dot operator
2. Manipulate Strings using the methods in the String class
3. Generate random numbers using the Random class
4. Perform mathematical operations using the Math class
5. Format output using the NumberFormat and DecimalFormat classes
6. Explore the utility of enumerated types.

## Table of Contents

## 1. Some fun with Strings

Let's create a program that explores some useful methods in the String class.

1. Create a new class called **FunWithStrings** inside package ca.bcit.comp1510.lab03:
    a. Include a **Javadoc** comment at the top of the class. The Javadoc comment should contain:
        i. The name of the class and a (very) short description
        ii. An @author tag followed by your name
        iii. An @version tag followed by the version number.
    b. Include a **main method** inside the class definition. Remember the main method is what gets executed first when we start our program. Include a Javadoc comment for the main method. The Javadoc comment should contain:
        i. A description of the method. In this case, "Drives the program." is a good idea.
        ii. An @param tag followed by args. We will learn more about parameters and arguments in a few weeks. For now, you can just write `@param args unused`.
2. Let's use a **Scanner object** to ask the user to enter a String. Remember to import the Scanner class at the top of the class (after the package) statement.
3. Declare and initialize a Scanner object that reads from the keyboard inside the main method. Take a peek at your lab 2 code if you need a reminder how to do this.
4. Declare a variable of type String to store the user's input. Remember variable names are always lower (or camel) case, like **input** or **userInput**.
5. **Prompt** the user to enter the title of their favourite book.
6. This is a good time to pause, save our work, and **make sure Checkstyle is working** (this includes using the Comp 1510 checkstyle configuration). In order to earn full marks for this labs and future labs, your code must not generate any Checkstyle complaints (unless your lab instructor relaxes this). Remember to right-click the project file, and choose Checkstyle > Activate Checkstyle.
7. You can also use Eclipse to auto-format your code. With your source file open, choose Source > Format from the menu. Eclipse usually (but not always) does a good job.
8. Use the scanner object to **acquire the user's input** and assign it to the String variable we declared in step 4. The scanner object is waiting for the user to enter the title of a book, so you can use the nextLine() method
    a. Remember to invoke it on the scanner object you instantiated
    b. Write the name of the object followed by a dot followed by the method name.
    c. Follow the name of the method with parentheses.
9. **Print** the value stored in the String variable to the console. Did it work?
10. **Challenge Question**: what about the Scanner's next() method? Does it work? Try it, and be prepared to describe what happens to your lab instructor. Remember you can read about the Scanner class in the official Java 11 API here: Scanner. Read the descriptions for the nextLine() and next() methods. Someone wrote the Javadoc to produce this documentation. Is it helpful?
11. Now is a good time to look at the API for the String class, too. The link is here: String.

12. Using a helpful method from the String class which you can invoke on the String object print the **length** of the title.
13. **Does the title of the book start with the word "The"**? Use a helpful method from the String class to tell the user.
14. There is a method in the String class called **toUpperCase()**. Will it permanently change a String? Invoke it on the String object storing user's input, then print the user's input. Did it change?
15. What if we invoke the toUpperCase() method on the String storing the user's input, and then **assign the result to a new String variable**. Did it work?
16. Can you tell your lab instructor what this means? (Hint: **mutability**)
17. **Print** the name of the book in upper case, and then again in lower case.
18. There is a very helpful method in the String class called **trim()**. It removes any leading or trailing whitespace from a String. Invoke it on the user's input and assign the result to a new String variable called **trimmedUserInput**. You can test what it does by printing the length of trimmedUserInput, and then using book titles like:
    a. Nothing but tabs
    b. Nothing but spaces
    c. Spaces, then a word, then spaces
    d. Spaces, then some words, then more spaces.
19. Finally, display what the user typed, **fully trimmed, in lower case, with the first and last letters capitalized**. (Hint: you might find the substring methods useful.)

## 2. Simulating Dice with the Random class

A long time ago in a social space far, far away we developed role playing games that used dice. (Interesting fact: dice is a plural form, and the singular form is die.) One of the more popular and enduring games has been Dungeons and Dragons.

Dungeons and Dragons (D&D) uses five dice, but only one is a typical 6-sided die. D&D uses a **4-sided die, a 6-sided die, an 8-sided die, a 10-sided die, a 12-sided die, and a see it to believe it 20-sided die**.

Write a complete Java program called **Dice** that simulates the rolling of a collection of D&D dice. You must use the **Random** class. Don't forget to import it. You can see its API here: Random .

1. Start by creating a new class called **Dice** inside package ca.bcit.comp1510.lab03.
2. Include **comments** and activate **Checkstyle**
3. **For each die in the collection, the program should generate a random number** between 1 and the maximum value (inclusive)
4. The program should print out the result of the roll for each die and the **total** roll (the sum of the six), all appropriately labeled.

## 3. Calculating the distance between two points

Write a program called **Distance** that calculates the distance between two points:

1. Create a class called **Distance** inside package ca.bcit.comp1510.lab03
2. Include **comments** and activate **Checkstyle**
3. The two points considered are on the **x-y plane**, and have x and y coordinates
4. Use a **Scanner** object to ask the user for the points. We can ask the user to enter two values at the same time separated by white space. Then we can scan each one in order, like this:

   ```
   Scanner scan = new Scanner(System.in);
   System.out.println(Your message here);
   double x1 = scan.nextDouble();
   double y1 = scan.nextDouble();
   ```

5. The **distance** between two points $(x_1, y_1)$ and $(x_2, y_2)$ is **calculated** by taking the square root of the quantity $(x_2 - x_1)^2 + (y_2 - y_1)^2$
6. **Use methods from the Math class to help you**. Check out the Math API document here: Math. Note that the methods in the Math class are all static. We invoke static methods on the name of the class. We don't need to create an object. So we should not do this:

   

   ```
   Math mathObject = new Math();
   double result = mathObject.sqrt(someNumber);
   ```

   We should just do this:

   

   ```
   double result = Math.sqrt(someNumber);
   ```

7. **Print** the result to the console. Does it work for points in all 4 quadrants?
8. Refine the result by using a DecimalFormat object:
   i. Declare and instantiate a **DecimalFormat**. Let's print the distance to two decimal places. We can do this by passing the pattern "#.##" to the DecimalFormat constructor. Does this print a leading zero if the result is small?
   ii. Visit the DecimalFormat API at DecimalFormat. Note the extensive discussion on Patterns. You can use this as a reference. There are some interesting methods, and we are interested in the **format** method.
   iii. Invoke the format method, pass the distance as the parameter, and print the formatted value.

## 4. Enumerations and Playing Cards

For your final exercise, let's work with enumerations:

1. Create a new called **CardGame** inside package ca.bcit.comp1510.lab03
2. Inside the class, but not inside the main method, define an enumerated type named Rank that contains all the standard card values `ace, two, three, four, …, nine, ten, jack, queen, king`.

3. Inside the class, but not inside the main method, define an enumerated type named Suit that contains all the standard card suites `hearts, diamonds, clubs, spades`.
4. This is a good time to pause, save our work, and **make sure Checkstyle is working**. In order to earn full marks for this lab and future labs, your code must not generate any Checkstyle complaints (*your lab instructor can override this*). Remember to right-click the project file, and choose Checkstyle > Activate Checkstyle.
5. The following code must be added to your main method:
6. Let's simulate a random card generator. Declare and instantiate a Random object.
7. Use the Random object to randomly select an integer between 0 and the length of the Rank enumeration. Your code snippet will look like this:

   ```
   int randomRankChoice = random.nextInt(Rank.values().length);
   ```
8. You can use than randomly generated number to select a Rank like this:

   ```
   Rank randomRank = Rank.values()[randomRankChoice];
   ```
9. Write similar code to generate a random number and select a random suit.
10. Print the results to the screen.

## 5. You're done! Show your lab instructor your work.

Did you notice? With all the API links given, you can get from one to the other by using the search function. Or you can start from the top API link (which is hard to find from the Oracle site): https://docs.oracle.com/en/java/javase/17/docs/api/.

If your instructor wants you to submit your work in the learning hub, export it into a Zip file in the following manner:

1. Right click the project in the Package Explorer window and select export…
2. In the export window that opens, under the General Folder, select Archive File and click Next
3. In the next window, your project should be selected. If not click it.
4. Click *Browse* after the "to archive file" box, enter in the name of a zip file (the same as your project name above with a zip extension, *such as* Comp1510Lab03BloggsF.zip if your name is Fred Bloggs) and select a folder to save it. Save should take you back to the archive file wizard with the full path to the save file filled in. Then click Finish to actually save it.
5. Submit the resulting export file as the instructor tells you.