Lab 4: Writing Classes

Let's get started!

Today we'd like you to:

1. **Open Eclipse**. Remember 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. If you need some help organizing your workspace, we'd be happy to help!
2. **Create a new Java project**. Call it something that will help keep your code organized, i.e., COMP 1510 Lab 4.
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 4 project to quickly create a new Java class.
4. When you have completed the exercises, **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. Although you can 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. Design and implement some simple classes
2. Encapsulate your objects by using visibility modifiers and getters/setters
3. Write simple methods that accept parameters and return values
4. Implement constructors that initialize objects in logical and organized ways
5. Explore JavaFX and create your first Graphical User Interface (GUI).

Table of Contents

Let's get started! **1**

What will you DO in this lab? **1**

1. Experiment with the Integer Class **2**

2. Introducing JavaFX **2**

5. Colours and Shapes: Create a Self-Portrait! **4**

6. Create a Name class **5**

7. Create a Student class **5**

8. You're done! Show your lab instructor your work. **6**

1. Experiment with the Integer Class

Wrapper classes are described on pages 127-129 of the text. They are Java classes that allow a value of a primitive type to be "wrapped up" into an object, which is sometimes a useful thing to do (we will see a good example of this when we study collections later in the term).

Wrapper classes often provide useful methods for manipulating the associated type. A wrapper class exists for each of the primitive types: byte, short, int, long, float, double, boolean, char. The API document for the Integer wrapper class can be found here: https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html .

1. Create a new class called **IntegerWrapper** inside package ca.bcit.comp1510.lab4:
    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 followed by args followed by unused.
2. Use the constants and methods in the Integer class to complete the following tasks in your class. Be sure to **clearly label your output for each task before proceeding** to the next.
3. Prompt for and read in an integer, then print the **binary, octal and hexadecimal** representations of that integer.
4. Print the **maximum and minimum** possible Java integer values. Use the constants in the Integer class that hold these values — don't type in the numbers themselves. Note that these constants are static.
5. Prompt the user to enter two integers, one per line. Use the next method of the Scanner class to read each of them in. (The next method returns a String so you need to store the values read in String variables, which may seem strange.) Now **convert the strings to ints** (use the appropriate method of the Integer class to do this), add them together, and print the sum.

2. Introducing JavaFX

It’s time for our first JavaFX program. We will use JavaFX for our graphical user interfaces in COMP 1510. It’s fast, easy to understand, and efficient. Did we say it’s fun, too?

A JavaFX program is a wee bit different from a command line program:

3. The JavaFX classes we create must **EXTEND** the javafx.application.Application class. This is an example of inheritance which we will explore in detail later this term.
4. The **main method** only contains a single line of code: launch(args). The main method will invoke (call) the JavaFX launch method. This performs some background set-up and then invokes (calls) the start method.
5. We will always write our code in the **start** method. The start method constructs the graphics.
6. JavaFX embraces a **theatre metaphor** (see page 132 of the text). We create our JavaFX components and add them to a **Group** object. Then we create a **Scene** object and add the Group. Once our Scene is fully populated, we pass it to the **Stage** which displays our Scene.

The Java graphic coordinate system is discussed in Section 3.9 of the text. Under this system, the upper left-hand corner of the window is the point (0,0). The X axis is horizontal, and the Y axis is vertical. So the larger the X value, the farther a point is to the right. The larger the Y value, the farther it is down. There are no negative X or Y values in the Java coordinate system. (Actually, you can use negative values, but since they're off the screen they won't show up!)

1. Create a new class called **MyFirstGraphicProgram** inside package ca.bcit.comp1510.lab4:
    a. **NEW FOR JAVAFX!** In the new Java Class dialog box, enter javafx.application.Application as the superclass.
    b. 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.
    c. 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 followed by args followed by unused.
2. After Eclipse generates your class, note its contents. We declared the ‘superclass’ to be the Application class from the JavaFX package. Eclipse automatically added an extra method called start.
3. Inside your main method you only need one line of code
    launch(args);
4. We will add the rest of our code to the start method.
5. Let’s start by creating a Circle. You can read about it in Figure 3.11 (page 133) in the text, or look at the Java API document at https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Circle.html . Create a Circle of radius 100 centred at (0,0). Remember to import the class!

6. Declare and instantiate a new Group object and add the Circle to it (refer to listing 3.8 for an example how to do this).
7. Declare and instantiate a new Scene object. Add your Group to the Scene, and make the Scene 500 pixels by 500 pixels in size.
8. Set the name of the primary Stage to "My first JavaFX program."
9. Set the primary Stage's scene to be the scene we created.
10. Invoke the primary stage's show() method.
11. Execute the program.
12. Did you see the Circle? Did you see the entire Circle? Adjust the parameters passed to the Circle class so that it is centred in the window. Can you do it without using magic numbers? (Hint: declare some constants!)
13. What colour was the Circle? Can you change the colour? Hint: the Circle has a method called setFill which accepts a Color as a parameter. Display a green Circle. You may want to refer to section 3.11 of the text, or visit the JavaFX Color Javadoc which is here: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html .
14. We can use the Text class to add text to our JavaFX, too. Can you add your name (in white!) to the centre of the Circle? Create a Text object, and remember to add it to the list of parameter you pass to the Group (so that it is added to the Group, which is added to the Scene). Hint: the Text class also has a method called setFill which can be used to set the colour of the Text. Don't forget to import the Text class.

3. Colours and Shapes: Create a Self-Portrait!

Time to get creative. Let's write a JavaFX program that draws a face. Whose face, you ask? Yours, of course! Use JavaFX to create a **self-portrait**. Include eyes, a nose, mouth, hair, etc. **Hint: use your phone! Take a selfie and use it for inspiration**.

1. Create a new class called **Face** inside package ca.bcit.comp1510.lab4:
    a. **NEW FOR JAVAFX!** In the new Java Class dialog box, enter javafx.application.Application as the superclass.
    b. 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.
    c. 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 followed by args followed by unused.

2. After Eclipse generates your class, note the contents of Face. We declared the 'superclass' to be the Application class from the JavaFX package. Eclipse automatically added an extra method called start.
3. Inside your main method you only need one line of code
   launch(args);
4. Add the rest of your code to the start method. Use the shapes introduced in section 3.10 of the text (and demonstrated in listings 3.8 and 3.9).
5. Are you ready to share your self-portrait? After your lab instructor marks your lab, take a screen shot of your self-portrait and email it (JPG or PNG only please) to cthompson98@bcit.ca . Next week I'll share everyone's images with you.

4. Create a Name class

Let's create a class that can be used to represent a Name.

1. Create a new class called **Name** inside package ca.bcit.comp1510.lab4:
   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. Do not include a **main method** inside the class definition. In this case, the class is not a complete program. It will simply be used to represent a Name concept.
2. A Name has three parts: a first name, a middle name, and a last name. Create three **private instance variables** to represent these three components of a Name.
3. A Name has a public **constructor** which accepts three parameters for the first, middle, and last names in that order. Add a constructor to the Name class. The body of the constructor should assign each parameter to its respective instance variable.
4. Add an **accessor** and a **mutator** for each instance variable (so there will be a total of three accessors and three mutators).
5. Create a **toString()** method which returns a String composed of the concatenation of the first, middle, and last names.

5. Create a Student class

Let's create a class that can be used to represent a Student.

1. Create a new class called **Student** inside package ca.bcit.comp1510.lab4:
   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. Do not include a **main method** inside the class definition. In this case, the class is not a complete program. It will simply be used to represent a Name concept.
6. A Student contains the following information: first name, last name, year of birth, student number, and GPA. Using sensible data types and variable names, declare private **instance variables** for each of these.
7. A Student has a public constructor which accepts one parameter for each of the instance variables. The body of the constructor should assign each parameter to its respective instance variable.
8. Create an accessor and a mutator for each instance variable.
9. Create a toString() method which returns a String composed of the concatenation of the information in the Student.

6. You're done! Show your lab instructor your work.