Lab 4: Introduction to JavaFX

Intro to Eclipse

To do this lab, and all future projects in the course, you'll be using Eclipse. Eclipse is a very popular Java "Integrated Development Environment" (IDE). An IDE is a software application that is designed to make writing code in a specific language easier for you. Up until now, you have been using Atom, which conveniently color-codes and auto-completes (somewhat), but it is limited other than that. Eclipse has a number of tools that will make your life much easier. These include:

- Automatic compiler that highlights syntax errors for you so you won't have to worry about compile time errors anymore
- Code hinting, which gives you options to autocomplete what you are typing
- An auto-formatting function that automatically indents code when necessary
- Extremely time-saving automatic importing (great for JavaFX apps!)
- A run button that allows you to run code straight from the IDE
- A User Interface (UI) that is easier to use with big projects
- **A powerful debugger**

Click here to access setup instructions (we will set up eclipse in lab!)

The Javadocs

One more thing before we dive in. Java has a huge collection of built-in classes for programmers to use. You're probably familiar with a couple of these classes by now, such as **`String`** and **`javafx.scene.paint.Color.`**

You will be using several of these built-in classes in this lab. But each one has so many methods and properties. However will you keep them straight? Luckily, Java developers have created the Javadocs — a set of detailed descriptions for each class and method you can use. There is a document here which provides an intro to the javadocs. There is also a convenient link to them on the CS15 website (Documents » Java Documentation >> Javadocs), or choose the first result when you Google "Java 8 API".

Review: Private methods and classes

Let's do a quick review on the purpose of private methods and classes. In Java, there's an important concept called "encapsulation." The idea of encapsulation is that each part of your program is only accessible by the other parts of your program that directly need it. Most classes are public because we want to be able to instantiate them from everywhere, but their instance

```
}
```

variables are private because we only want that particular class to modify them. Encapsulation helps give you more control over your code and helps you avoid a lot of particularly nasty bugs.

We've seen private instance variables, but we can also write private methods too! If we're interested in factoring out some common code within a class, we can use a private helper method. Consider the following example, which creates a **`PerryThePlatypus`** class and provides a number of methods that give Perry certain capabilities:

```
public class PerryThePlatypus {
      // constructor code elided

      public void solveMystery() {
            this.getGadgets();
      }

      public void stopDoof() {
            this.getGadgets ();
      }

      private void getGadgets() {
            // implementation elided
      }
}
```

In the above case, Perry is able to both solve a mystery and stop Doof by getting gadgets. Any other part of the program that accesses this **`PerryThePlatypus`** class should be able to solve mysteries and stop doof so we make these methods public. However, we make getGadgets() a private method because no other classes in the program need to call it. Why? Because Perry only needs to get gadgets when solving mysteries or stopping Doof (which are in the same class). Because getGadgets() is a private method, the following code would NOT work:

```
public class OWCA {
      public OWCA() {
            PerryThePlatypus perry = new PerryThePlatypus();
            perry.getGadgets(); //Will not work!
      }
```

```
        }
```

In a similar vein, when you only need to reference an object from within one particular class, it’s cleaner to store that object in an inner *private class*. Private classes have direct access to the instance variables and methods of the class that contains them, and vice versa, which can come in handy. When designing a program, you should carefully consider which classes should be public and which can be private.