Lab 1: Introduction to Java

Welcome to the second CS15 lab! In the reading, we went over objects, methods, and parameters, as well as how to put all of them together into Java classes. It's perfectly okay if you are feeling a little overwhelmed at this point—you've gone over a lot of new material very quickly. But don't worry! This lab will help reinforce what you've learned by guiding you through writing a small Java program. The TAs are here to help, so feel free to ask any questions about the lab, lectures, or general course material.

Partner Programming

For this lab, we will be working in pairs! You should have already received your partners for this lab. Make sure you're sitting with them at one computer!

Normally, you will complete a reading before the lab, and submit a mini-assignment that will enforce concepts from that reading. However, for this lab, we will not be doing a mini assignment. Instead, look through the reading for this lab (linked on the "Section" page of the course website). Make sure you and your partner are discussing the concepts in this reading, since you will need to refer back to it throughout this lab.

Classes

In the reading for this lab, we reviewed the use of ***packages***. Now that we understand packages, let's move onto making a new class. Each Java class should be in its own file, with the name of the class being the name of the file[1]. All Java files should end in **`.java`**. For example, a Clown class would be saved as **`Clown.java`**. If you don't remember how to declare a class, you can look back over the lecture notes.

The `CupcakeDecorator` Class

- Run **`cs0150_install lab1`** in your terminal to get the stencil code for lab1. This will create an **`App.java`** file in **`/home/<yourlogin>/course/cs0150/lab1/`**.
- Open **`App.java`** in Atom by typing **`atom App.java &`** into your terminal to launch Atom).
- Create a new class file and name it **`CupcakeDecorator.java`**. You can do this by typing **`touch <file name>`** into the terminal (but make sure you are in the correct directory!). Or if you ever forget the command, simply select "File -> New File" in atom and "Save as" into the correct folder.
- Declare the package as **`lab1`**.

[1] There are classes called inner classes that do not follow this rule. We will be going over them in lecture in two weeks.

- Write a class declaration and an empty constructor for **`CupcakeDecorator`**. This is just a simple constructor that does nothing. If you're confused on how to do this, refer to the lecture slides for more information, or read the following two sections for more information, and then come back to this one.

- Compile and run your program from the terminal.
    - **`javac *.java`** to compile
    - **`java lab1.App`** to run

You should now see a yellow cupcake and its owner on the right.

Making a Program

For this lab, you're going to be creating your own Cupcake! Currently, the Cupcake doesn't have frosting, sprinkles, or a cherry. The **`CupcakeDecorator`** class that you've started writing will be responsible for creating these cupcake essentials. It will then add these parts to the empty cupcake. In order to do this, **`CupcakeDecorator`** will have to know about the empty cupcake, which is a class we will provide to you. Since **`CupcakeDecorator`** didn't instantiate **`Cupcake`**, the best way to do so is through an association using the constructor.

Passing Parameters

- Modify the constructor for **`CupcakeDecorator`** so that it receives one parameter of type **`cs015.labs.CupcakeSupport.Cupcake`**.

- Inside the constructor of **`App.java`**, instantiate an instance of **`CupcakeDecorator`** by typing in the line below. (Look in the code comments to see where this is done!)

```
CupcakeDecorator decorator = new CupcakeDecorator();
```

- Try compiling your program now and notice that an error is thrown, as shown below:

```
App.java:16: error: constructor CupcakeDecorator in class CupcakeDecorator
cannot be applied to given types;
        CupcakeDecorator decorator = new CupcakeDecorator();
                                     ^
required: Cupcake
found: no arguments
reason: actual and formal argument lists differ in length
```

- We can see from this error message that there is a mistake in our code in the file **App.java** . Additionally, the mistake is that we are calling the constructor **CupcakeDecorator()** without giving it the proper arguments.
- Fix the bug by passing in the **plainCupcake** to the **CupcakeDecorator** constructor.
- Make sure your parameters are syntactically correct and your code compiles without errors.

Checkpoint 1: Now is a good time to switch off with your partner. Make sure to follow along and assist your partner while they are coding.

Now it's time to add all the other components of a cupcake (i.e. sprinkles, frosting, and cherry). A great place to instantiate these components is in the constructor of **CupcakeDecorator**. First we'll add frosting by instantiating the **cs015.labs.CupcakeSupport.CupcakeFrosting** object. We'll do the same for **CupcakeSprinkles** and **CupcakeCherry**.

However, before you actually add these components, notice that you'll be essentially typing the same things (**cs015.labs.CupcakeSupport.<...>**) for each one. We don't want to waste time typing this for every single class, so instead we should import these classes.

When we type **cs015.labs.CupcakeSupport.CupcakeFrosting**, we are telling Java that we want to use the **CupcakeFrosting** class from the package **cs015.labs.CupcakeSupport**. It would be pretty annoying to have to write out the full package name every time we want to use support code from the package. By importing a class, the Java compiler will automatically fill in the package name of a class, so you can simply refer to **cs015.labs.CupcakeSupport.CupcakeFrosting** as **CupcakeFrosting**.

Not only will this make your life easier, but it also will make your code more readable and concise.

Importing Classes

In order to import classes, you must tell Java at the top of your code. Here's an example:

```
package SaladMaker;

import kitchen.allThingsSalad.Lettuce;
import kitchen.allThingsSalad.Tomato;
import kitchen.allThingsSalad.Dressing;

public class SaladMaker {
      public SaladMaker(Lettuce l, Tomato t, Dressing d) {
```

```
                <code elided>
          }
    }
```

As you can see, we've imported classes **`Lettuce`**, **`Tomato`**, and **`Dressing`** from **`kitchen.allThingsSalad`**. But in this case, we know that we are going to use all of the classes contained in the **`allThingsSalad`** package. Therefore, we can instead call **`import kitchen.allThingsSalad.*`**, which indicates that we're importing all classes contained in the package and saves us a couple lines of code. In this way, the following code is synonymous to the first code example:

```
package SaladMaker;

import kitchen.allThingsSalad.*;

public class SaladMaker {
      public SaladMaker(Lettuce l, Tomato t, Dressing d) {
            <code elided>
      }
}
```

Note that **`import <package>.*`** should only be used if you need **all** of the classes from a package. According to the CS15 Style Guide, if you are only using some of the classes from a package, you should import them individually, as demonstrated in the first example. The idea is that we don't want to import classes that we are not using. Note: for the purposes of this course, you will almost never need every class in a package.

Now we are ready to frost a cupcake.

Imported Frosting

- Open up **`CupcakeDecorator.java`**
- Import ***all*** the support classes for this lab, which can be found in the package **`cs015.labs.CupcakeSupport`**.
- Inside the constructor for **`CupcakeDecorator`**, instantiate an instance of **`CupcakeFrosting`**.

  Hint: if you're getting an error, make sure you've correctly imported all support classes.
- Run the App. You'll see nothing has changed. Although you've made the frosting, the empty cupcake has no idea that you've made them. Fortunately, **`Cupcake`** has a method you can call to tell it about the frosting you've just made. **`cs015.labs.CupcakeSupport.Cupcake`** has a method **`add(...)`** that takes in a **`CupcakeFrosting`** as a parameter and does not return anything.

- Inside the **`CupcakeDecorator`** constructor, call the **`add(...)`** method on the **`Cupcake`** object that was passed into the constructor. As a parameter to the **`add(...)`** method, pass the instance of **`CupcakeFrosting`** that you've just instantiated.

- Run the App again. Yay! The cupcake has frosting now!

If you are having trouble getting the method invocation syntax correct, review the lecture notes. Remember that there are 3 parts to a method invocation: the receiver of the message, the name of the method, and the parameters (if any).

The Cherry on Top

- Finish up the empty cupcake by adding sprinkles and a cherry in the same way that you added frosting. Here are the classes to add:
    - **`cs015.labs.CupcakeSupport.CupcakeSprinkles`**
    - **`cs015.labs.CupcakeSupport.CupcakeCherry`**

  Hint: If you successfully imported the entire support code package, you don't have to include the full package name when instantiating.

Checkpoint: After you are done, call over a TA and show them your cupcake. Then get checked off for the lab. Congratulations on finishing Lab1!