Lab 1: Introduction to Java

Welcome to the second CS15 lab! By now we've gone over objects, modeling, properties, attributes, and how to put all of these things 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 leading you through writing a small Java program. The TAs walking around are there to help, so feel free to ask any questions about the lab, the lectures, or general course material.

General Programming Guidelines

Our goal is to get started using Java and learn how to create a class from scratch. Before we get started on the mini-assignment, let's go over some general Java and programming guidelines:

- Packages
- Classes
- Methods and Parameters
- Constructors

Packages

Java programs are built upon classes — each program must have at least one. As the programs you write grow in size, it can be messy to have all of your classes lying around. Thankfully, Java organizes classes into packages to keep code organized and manageable. You can think of a package as a stand alone group of classes that performs a certain task. For example, if your program was a video game, its packages could include the graphics package, the physics package, the AI package, etc. Packages can also be nested (the physics package could contain a math package, the graphics package could contain a geometry package, and so on). All Java classes should belong to a package\(^1\), and you specify that package by typing:

    package <some package>;

at the very beginning of the file (before you even declare the class), and without the angle brackets. All this being said, the programs in CS15 aren't big enough to need multiple packages, so the name of your packages will simply be the names of the programs.

---

\(^1\) The only real exception is when you are writing a very small Java program that consists of only one class.
Classes

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 file². 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 CupcakeMaker Class

- Run the `cs015_install lab1` script to get the stencil code for lab 1. This will create an App.java file in `/home/<yourlogin>/course/cs015/lab1/`.
- Open App.java in sublime.
- Create a new class file and name it CupcakeMaker.java.
- Declare the package as lab1.
- Write a class declaration and an empty constructor for CupcakeMaker. This is just a simple constructor that does nothing.
- Compile and run your program from the terminal.
  - javac *.java to compile
  - java lab1.App to run

Methods and Parameters

A class can't do anything without you defining some capabilities for it. If you had a Robot class, you would want it to do something cool like: walk, dance, cook, maybe even do your CS15 assignments for you. Java models these capabilities using methods. Methods have a specific syntax; refer to lectures or the book if you don’t remember.

Sometimes a method needs outside information in order to perform its task. A way to pass information to a method is through parameters. For example, a Robot’s `cook(...)` method needs to know what to cook. Since your Robot is a culinary genius, if you didn’t use parameters, you would end up writing tons of cook methods for every possible dish your Robot can cook! Just think: `cookChicken()`, `cookPork()`, `cookSteak()`, `cookTomatoes()` ...Or you can write one method that takes in a food that you want your Robot to cook as a parameter. When a method takes a parameter, it uses that parameter as a variable.

---

² There are classes called inner classes that do not follow this rule. We will be going over them in class in 2 weeks.
Constructors

Let's not forget constructors; they are special methods that are automatically called when an object gets instantiated (i.e. every time you call new <someConstructor>() ;). Every class needs one\(^3\), and they are usually used to instantiate instance variables and set default values. Going back to our Robot example, once a new Robot has been built, Java will look in the constructor to see what default values should be given to its color, name, height, weight, etc. Constructors have to be named the same as their class, so a Robot class will have a Robot constructor. Look into the lecture slides for specific syntax.

Just as in any method, parameters are very useful for constructors. They can be used to set up associations (knowledge of other objects) that are not known until an object is instantiated. Remember the DogGroomer and PetShop example from lecture? We can use a parameter in the Robot class to let the Robot know which student it belongs to and what color the student ordered its Robot to be.

Now that you're more familiar with the Java syntax, it's time to get down to business...

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 CupcakeMaker class that you've started writing will be responsible for creating frosting, sprinkles, and a cherry. It will then add these parts to the empty cupcake. In order to do this, CupcakeMaker will have to know about the empty cupcake. Since CupcakeMaker didn't instantiate Cupcake, the best way to do so is through an association using the constructor.

\(^3\) If you don't write a constructor, Java will automatically create one for you, but it won't do anything.
Passing Parameters

- Modify the constructor for CupcakeMaker so that it receives one parameter of type cs015.labs.CupcakeSupport.Cupcake.
- Inside the constructor of App.java, instantiate an instance of CupcakeMaker passing it the plain Cupcake (look in the code comments to see where this is done).
- Make sure your parameters are syntactically correct and your code compiles without errors.

Now it's time to add all the other components. A great place to instantiate the object's components is in the constructor of CupcakeMaker. Let's begin by adding frosting (cs015.labs.CupcakeSupport.CupcakeFrosting) to the cupcake--HOLD UP, ain't no one got time to type out cs015.labs.CupcakeSupport.<...> every time they want to access support code, which brings us to importing 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 cs015.labs.CupcakeSupport (see, it's already annoying!).

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 is this a benefit to your quality of life, but it also brings added readability and conciseness to your code.

How to Import Classes:

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

```java
package SaladMaker;

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

public class SaladMaker {
```
As you can see, we've imported classes Lettuce, Tomato, and Dressing from kitchen.allThingsSalad. But because we know that we are going to use all of the classes contained in the allThingsSalad package, so we can import kitchen.allThingsSalad.*, which indicates that we’re importing all classes contained in the package. This saves us a couple lines of code. Therefore, the following code is synonymous to the above code:

```java
package SaladMaker;

import kitchen.allThingsSalad.*;

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

Ok, now we’re ready for some frosting.

### Imported Frosting

- **Import all** the support code for this lab, which can be found in the package cs015.labs.CupcakeSupport.

- **Inside the constructor for** CupcakeMaker, instantiate an instance of CupcakeFrosting.

  **Hint:** if you’re getting an error, make sure you’ve completed the first step correctly.

- **Run the App.** You'll see nothing has changed. This is because although you've made the frosting, the empty cupcake has no idea that you've made them. Fortunately, the 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.

- **Call the add(...) method inside the** CupcakeMaker constructor on the Cupcake object that was passed in and pass it 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 there are 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 support code package, do you have to include the package name?

Coding Conventions

Programming has a very unique style element to it. While the compiler couldn't care less about what your code looks like, neat and well formatted programs make it much easier to develop, debug, and look back at old code later on. Please take a look at CS15’s Style Guide before moving on.

Remember: Code Style will be part of the rubric for each programming assignment.

Coding Conventions Mini-Quiz

To make sure that you understand our style expectations, please take the following quiz:
- https://docs.google.com/a/brown.edu/forms/d/1iVmuRNOu32MqUi8xFIKn5wJ99LaGUwSwKnj6faEiPYk/viewform

Note: Make sure to keep your completed quiz open to show the TAs later!
TA Hours

TA Hours policies will now become enforced as we move out of shopping period. To review and understand the policies that we have laid out, please read Section 8 of the Collaboration Policy then complete the following mini-quiz.

TA Hours Mini-Quiz

To make sure that you understand our expectations for TA Hours, please take the following quiz:
- [https://docs.google.com/forms/d/1aDqVSyzdevFGOBa2_O6KfWzx-FDDHFbKC06u4bko8_k/viewform?usp=send_form](https://docs.google.com/forms/d/1aDqVSyzdevFGOBa2_O6KfWzx-FDDHFbKC06u4bko8_k/viewform?usp=send_form)

**Checkpoint:** Keep your completed Quiz webpage up. Call a TA over to check you off for your working Cupcake Maker and your completed quiz.

Congratulations on finishing Lab 1!