Lecture 02: Defining and Using Classes

8/28/2020

Compilation

```
Compiling and running code from the terminal:

$ javac HelloWorld.java
$ ls
HelloWorld.java    HelloWorld.class
$ java HelloWorld
Hello World!
```

Compilation

- The standard tools for executing Java programs use a two step process:
  - Hello.java -> javac compiler -> Hello.class -> java interpreter -> program runs
  - This is not the only way to run Java code
- Why make a class file at all?
  - .class file has been type checked. Distributed code is safer.
  - .class files are 'simpler' for the machine to execute. Distributed code is faster.
  - Minor benefit: Protects your intellectual property. No need to give out source.

Defining and Instantiating Classes

```
/** Dog.java file */
public class Dog {
    public static void makeNoise() {
        System.out.println("Bark!");
    }
}
/** Can't be run directly, no main method */

/** DogLauncher.java file */
/** The DogLauncher class will test drive the Dog class */
public class DogLauncher {
    public static void main(String[] args) {
        Dog.makeNoise();
    }
}
$ java DogLauncher
Bark!
```

Dog

- Every method is associated with some class
- To run a class, we must define a main method
  - Not all classes have a main method!

Object Instantiation

- We could create a separate class for every single dog out there, but his is going to get redundant very quickly
- Classes can contain not just functions (aka methods) but also data
- Classes can be instantiated as objects
  - We'll create a single Dog class, and then create instances of this Dog
  - This class provides a blueprint that all Dog objects will follow

```
public class Dog {
    public int weightInPounds;  // An instance variable

    /** One integer constructor for dogs. */
    public Dog(int w) {  // Constructor (similar to __init__)
        weightInPounds = w;
    }

    public void makeNoise() {  // Non-static method, instance method
        if (weightInPounds < 10) {
            System.out.println("yip!");
        } else if (weightInPounds < 30) {
            System.out.println("bark.");
        } else {
            System.out.println("wooooof!");
        }
    }
}

/** DogLauncher.java file */
/** The DogLauncher class will test drive the Dog class */
public class DogLauncher {
    public static void main(String[] args) {
        Dog smallDog;  // Declaration of a Dog instance
        new Dog(20);   // Instantiation of a Dog instance
        smallDog = new Dog(5);  // Instantiation and Assignment
        Dog mediumDog = new Dog(25);  // Declaration, Instantiation, and
Assignment
        d.makeNoise();  // Invocation of the 25 lb Dog's makeNoise method
    }
}
$ java DogLauncher
bark.
```

Defining a Typical Class (Terminology)

- **Instance variable**: Can have as many of these as you want

- **Constructor**: Similar to a method (but not a method), determines how to instantiate the class
- **Non-static method**: a.k.a., instance method. Idea: If the method is going to be invoked by an instance of the class, then it should be non-static
    - Roughly speaking: If the method needs to use "**my** instance variables", the method must be non-static

Arrays of Objects

- To create an array of objects:
    - First use the new keyword to create the array
    - Then use new again for each object you want to put in the array

```
Dog[] dogs = new Dog[2];  // Creates an array of Dogs of size 2
dogs[0] = new Dog(8);
dogs[1] = new Dog(20);
dogs[0].makeNoise();  // Yipping occurs
```

Static vs Instance Methods

Static vs Non-static

- Key differences between static and non-static (a.k.a. instance) methods
    - Static methods are invoked using the class name, e.g. `Dog.makeNoise();`
    - Instance methods are invoked using an instance name, e.g. `maya.makeNoise();`
    - Static methods can't access "my" instance variables, because there is no "me"

Why Static Methods?

- Some classes are never instantiated. For example, Math.
    - `x = Math.round(5.6);`
- Sometimes, classes may have a mix of static and non-static methods

```
public class Dog {
    public int weightInPounds;  // An instance variable

    /** A static variable that applies to all dog instances */
    public static String binomen = "Canis familiaris";

    /** One integer constructor for dogs. */
    public Dog(int w) {  // Constructor (similar to __init__)
        weightInPounds = w;
    }

    public void makeNoise() {  // Non-static method, instance method
        if (weightInPounds < 10) {
            System.out.println("yip!");
        } else if (weightInPounds < 30) {
            System.out.println("bark.");
        } else {
            System.out.println("wooooof!");
        }
    }

    public static Dog maxDog(Dog d1, Dog d2) {
        if (d1.weightInPounds > d2.weightInPounds) {
            return d1;
        }
        return d2;
    }

    public Dog maxDog(Dog d2) {
        if (this.weightInPounds > d2.weightInPounds) {
            return this;
        }
        return d2;
    }
}

public class DogLauncher {
    public static void main(String[] args) {
        Dog d = new Dog(15);

        Dog d2 = new Dog(100);

        Dog bigger = Dog.maxDog(d, d2);
        bigger.makeNoise();

        Dog bigger2 = d.maxDog(d2);
        bigger2.makeNoise();

        System.out.println(d.binomen);
        System.out.println(Dog.binomen);
    }
}

$ java DogLauncher
wooooof!
wooooof!
Canis familiaris
Canis familiaris
```

Static vs. Non-static

- A class may have a mix of static and non-static members
  - A variable or method defined in a class is also called a member of that class
  - Static members are accessed using class name, e.g. `Dog.binomen;`
  - Non-static members **cannot** be invoked using class name
  - Static methods must access instance variables via a specific instance of the class, e.g. `d2`

public static void main(String[] args)

- We already know what `public, static, void` means

One Special Role for String: Command Line Arguments

```
public class ArgsDemo {
    /** Prints out the -th command line argument. */
    public static void main(String[] args) {
        System.out.println(args[0]);
    }
}
$ java ArgsDemo hello some args
hello
```

ArgsSum Exercise

```
public class ArgsSum {
    public static void main(String[] args) {
        int N = args.length;
        int i = 0;
        int sum = 0;
        while (i < N) {
            sum = sums + Integer.parseInt(args[i]);
            i = i + 1;
        }
    }
}
```

Using Libraries (e.g. StdDraw, In)

Java Libraries

- There are tons of Java libraries out there
  - In 61B, we will provide all needed libraries including:
    - Built-in Java libraries
    - Princeton standard library
- As a programmer, you'll want to leverage existing libraries whenever possible
  - Saves you the trouble of writing code
  - Existing widely using libraries are (probably) will probably be less buggy
  - ... but you'll have to spend some time getting acquainted with the library
- Best ways to learn how to use an unfamiliar library:
  - Find a tutorial online
  - Read the documentation for the library (Java docs)
  - Look at example code snippets that use the library