CSC207 Lecture 2

September 12, 2017

Sadia Sharmin

Topics for today

- Practice with classes: Circle, Balloon
- Static variables and methods
- Collections
- We will also briefly touch upon: Javadoc, UML Diagrams, Unit Testing

Practice: Circle class

- Define a Circle class: has a radius and color

Practice: Circle class

- If you do not put a constructor, then Java uses a default one with no parameters and no body

The **main** method

```
public static void main(String[] args) {
    //...
}
```

- Every Java application must contain this method

- This is the method that gets called when your application is run -- it's the "entry point" for your application

- So far, we can know a few things about this method:
  - **public**: so that we can call the method
  - **void**: this method runs the program; not meant to return anything
  - **String[] args**: takes in a list of String arguments; these are parameters that could be passed in to the application when run from command line

More here: https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Circle: Adding constructors

- Remember: The constructor has same name as class, and no return type (not even void)
- Note: Once we define our own constructors, the default parameter-less body-less constructor is no longer available
- Practice: We want to have two ways to construct a Circle
    - Only a radius is provided, and color is set to invisible
    - Both radius and color are provided

Multiple Constructors

- In Java, a class is allowed to have multiple constructors as long as their signatures (the types of parameters they take in) is different

- Note: You have the ability to call one constructor from within another using the **this** keyword

```
public class Rectangle {
private int x, y;
private int width, height;

public Rectangle() {
    this(0, 0, 1, 1);
}
public Rectangle(int width, int height) {
    this(0, 0, width, height);
}

public Rectangle(int x, int y, int width, int height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
}
```

Source: https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

Static variables

- Static variables are **shared** across all instances of an object
- They are part of the class, not any individual object
- Only one copy of the variable is created; any changes to it will affect all the instances
    - *If you think of the **class** as a cookie cutter, and **instances** of it as the actual cookies created using that cutter, then **static variables belong to the cutter**, not to any individual cookie*

References:

- *https://stackoverflow.com/questions/2649213/in-laymans-terms-what-does-static-mean-in-java'*
- *https://beginnersbook.com/2013/05/static-variable/*
- *https://www.caveofprogramming.com/java/java-for-beginners-static-variables-what-are-they.html*
- *http://crunchify.com/java-static-methods-variables-static-block-and-class-with-example/*

Practice: Counting circles

- Add a private, static variable **numCircles** to the Circle class and update it each time a new Circle is created
- Add a static getter method which returns the current number of circles that this Circle class has been used to create
    - Why should we make this method **static** as well? What if we didn't make it static?

The **main** method

```
public static void main(String[] args) {
    //...
}
```

- The main method is **static** because we need to call it before instantiating any class – it is the entry point to our application

Practice: Extending Balloon class

- Add a private, static variable **numBalloons** to the Balloon class and update it each time a new Balloon is created, and a static **getNumBalloons** method which returns this value

- Add two more constructors:
  1) a parameter-less constructor which constructs an 'invisible' colored Balloon
  2) a constructor that takes in amount, capacity, popped and color

Practice: Balloon demo

What would the output of the following be?

```
public static void main(String[] args) {

    System.out.println(Balloon2.numBalloons);

    Balloon2 b1 = new Balloon2("pink");
    System.out.println(b1);

    Balloon2 b2 = new Balloon2();
    System.out.println(b2);

    Balloon2 b3 = new Balloon2(50, 500, true, "green");
    System.out.println(b3);

    System.out.println(Balloon2.numBalloons);

}
```

Aliasing

Aliasing is when there are multiple variables referring to the same object. Any changes that affect one variable also affect the other.

```
Balloon b3;
b3 = b1; // refers to same object
b3.setColor("dark green"); // this will change both b1 and b3's color
System.out.println(b1);
System.out.println(b3);
```

Pass-by-value

Java is pass-by-value, which is defined as follows:

> *The actual parameter (or argument expression) is fully evaluated and the resulting value is copied into a location being used to hold the formal parameter's value during method/function execution.*

More here: http://javadude.com/articles/passbyvalue.htm

The value of a reference is the address of the object.
So, when I pass in a reference, I don't actually pass in the object, but the address where it is stored.

Collections - ArrayList, Hashmap

See **GenericsExample.java** and **CollectionsExamples.Java**

Javadoc

Javadoc is a tool for generating API documentation for your classes.

Add documentation for each class, constructor or method by preceding each of them with a block description in the following format:

```
/**
 *
 * A few lines of description of what this method does.
 *
 * @param  arg1  description of what arg1 is
 * @param  arg2  description of what arg2 is
 * @return       description of what is returned
 */
public int methodName(int arg1, String arg2) {
    ...
}
```

In Eclipse, you may use Shift-Alt-J (Windows) or Command-Alt-J (Mac) to auto-generate a template of this for you.

Once you have the documentation written up, click **Project > Generate Javadoc** in the top menu to create an HTML page with all this information.

More here: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#styleguide
http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html

Unit testing

A unit test targets a small unit of code, e.g., a method or a class.

Write tests for the critical and complex parts of your application. Generally, no need to write tests for trivial code like getter and setter methods which simply assign values to fields.

Example test cases for a method **multiply(int, int)** in class MyClass:

```
public class MyTests {
    @Test
    public void multiplicationOfZeroIntegersShouldReturnZero() {
        MyClass tester = new MyClass(); // MyClass is tested

        // assert statements
        assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
        assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
        assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
    }
}
```

To create test case templates in Eclipse, select File > New > JUnit Test Case.

More here: http://www.vogella.com/tutorials/JUnit/article.html
http://www.vogella.com/tutorials/JUnit/article.html#usingjunit_asserts