Recitation 6 - February 13th A4/GR

Isaac and William

Static Methods and Instance Data

So We're Finally Covering the Static Keyword!

Static “Members”

Associated with a class!

- Static Members are associated with a class rather than any particular instance of an object.
- This means that they don’t need an instance to be called!
- Static methods cannot use instance data.
    - Why?

A Super Basic Static Example

```
public class Example {
     private static int testValue = 0;

     public static void setValue(int a) { testValue = a; }

     public static int getValue() { return testValue; }
}

public class Main {
     public static void main(String[] asdf) {
          Example.setValue(5);
          System.out.println(Example.getValue())
     }
}
```

- Notice how I can call the setValue Method and the getValue method on the Class Example!

So What is the Takeaway?

- Static Variables are used for when you want a variable to be the same among all instances of the object

- Static Methods are used when you want a method that can be called on the class
  ex. when it doesn't make sense for it to belong to any one instance

- It's for this reason that Static Members are often called Class Variables and Class Methods

Where is the Error?

```
public class Example {
    private static int testValue = 0;
    public int add = 0;
1.  public static void setValue(int a) { testValue = a; }

2.   public static int getValue() { return testValue + add; }
}

public class Main {
    public static void main(String[] asdf) {
        Example ex = new Example();
3.      ex.add = 5;
        System.out.println(Example.getValue())
    }
}
```

So what exactly is going on?

```
.\Example.java:10: error: non-static variable add cannot be referenced from a static context
                return testValue + add;
                                   ^
```

- So you've probably already seen this error before
- It happens when you try to access an instance variable in a class method
- So why can't we use instance variables with static methods?

Wrapper Classes

Java's Primitive Wrapper Classes

- Are "Wrapper" classes for Java's primitive types
- This allows you to treat primitives like double, int, boolean, etc. as **objects**
- We learn later it's actually bad practice to instantiate Java's primitive wrappers using the new keyword

```
Integer a = new Integer(5);

Integer b = new Integer("5");

Integer c = 5;

Double d = new Double(10.0);

Integer e = c + b;
```

Autoboxing

- `Integer i = 5`
  - Autoboxing

- `double d = new Double(1.22)`
  - Unboxing

Wrappers Static Methods - Parse/ValueOf

- These methods are static so you can call them on the class
- parseInt(), parseDouble(), etc, allows us to take in a string and return the primitive.
- valueOf(String s), or valueOf(int i) allows us to take in a string and returns the value of it to us in the Wrapper Class. The Java API tells us to use this method when we want to instantiate a new Wrapper

String s = “12”;

int a = Integer.parseInt(s);

// a = 12;

Integer b = Integer.valueOf(“12”)

Integer c = Integer.valueOf(7)

** Note: = new Integer(5) is considered deprecated, which means that its better to use the valueOf Method for better performance

What is outputted?

```
public static void main(String[] rip) {
    Integer a = Integer.parseInt(“10”);
    System.out.println(a);
}

-------------
```

Options:

A. Integer@1db9742
B. 10

Javadocs

Javadoc Example

```
/**
* This method is used to add two integers
* @param A the first number in a pair that you want to add
* @param B the second number in the pair you wish to add.
* @return returns the sum of variables A and B
*/
public int addNum(int A, int B) {
     return A + B;
}
```

We'll go over this in more depth during our live code session