Chapter 9: Wrapper Classes and Tokenizing Strings

Prof Clark

COSC 1437

Topic List Covered From Chapter 9

- Wrapper Classes (9.1 and 9.6)
  - What are they?
  - Why do we need them?
  - Autoboxing and Unboxing
- Tokenizing Strings (9.5)
  - String's split method
  - Delimiters

Wrapper Classes

Introduction

- Recall that Java has two types of variables: Primitives and References (Objects).
- Wrapper classes allow us to create **objects** that represent a primitive type.
- Java provides wrapper classes for all of the primitive types in `java.lang`, therefore no `import` statement is needed to use them. They are primarily the name of the primitive type, starting with a capital letter.
- You've already used these wrapper classes to parse data from an input dialog box (Integer.parseInt(), Double.parseDouble()).
- Wrapper objects are *immutable*, which means that once you create one, the value inside of it cannot change.

Primitive Data Type Wrappers

Wrapper Class	Primitive Type It Applies To
`Byte`	`byte`
`Double`	`double`
`Float`	`float`
`Integer`	`int`
`Long`	`long`
`Short`	`short`
`Character`	`char`

Creating a Wrapper Object

- To create objects from these wrapper classes, you can pass a value to the constructor:

```
Integer number = new Integer(7);
```

- You can also assign a primitive value to a wrapper class object:

```
Integer number;
number = 7;
```

Wait a minute...

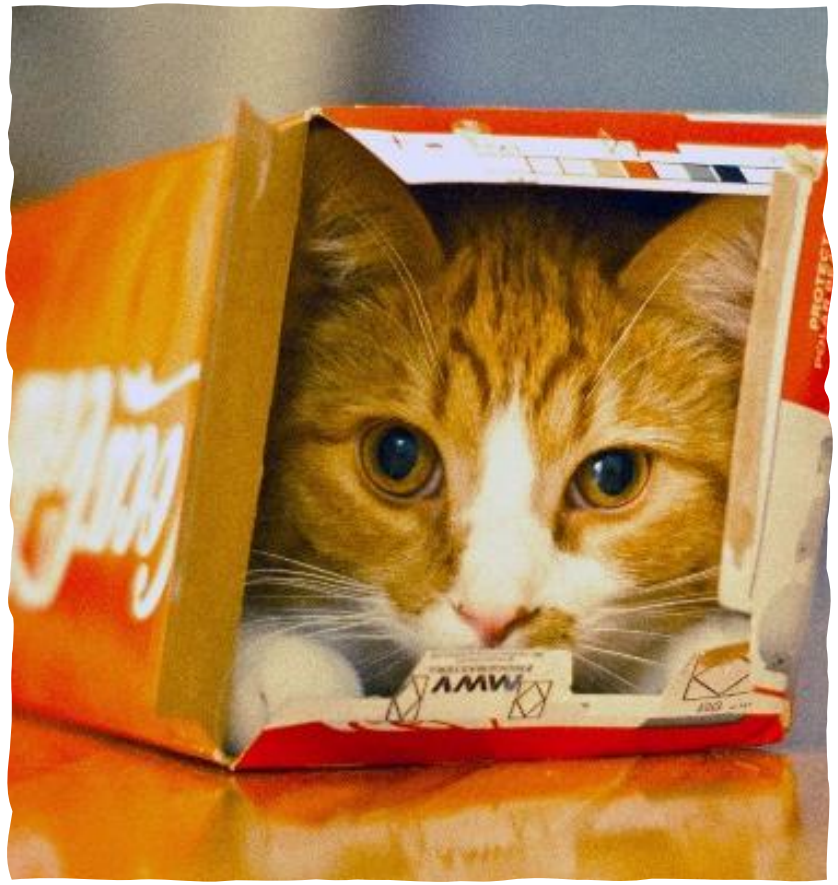
I just said this was ok:

```
Integer number;

number = 7;
```

But I'm assigning a primitive int value to an Integer object. Shouldn't that be an error?

Autoboxing



```
Integer number;
number = 7;
```

- You may think this is an error, but because `number` is a wrapper class object, *autoboxing* occurs.
- Autoboxing is Java's process of automatically "boxing up" a primitive value and putting it inside of a wrapper object

Autoboxing Examples

```
Double myNum = 7.8;
```

This creates a Double object named myNum that is a reference to a memory location on the heap with 7.8 inside of it. It is doing exactly the same thing as this line:

```
Double myNum = new Double(7.8);
```

```
Long myBigNumber;
myBigNumber = 12345678912345678 9;
```

This creates a Long object named myBigNumber on the first line, but it doesn't reference a value yet.

The second line creates the memory location on the heap and puts the value 12345678912345678 9 inside that memory location on the heap. It's doing the same thing as this line:

```
Long myBigNumber = new Long(12345678912345678 9);
```

Unboxing

- *Unboxing* does the opposite with wrapper class variables:

```
Integer myInt = 5;          // Autoboxes the value 5
int primitiveNumber;
primitiveNumber = myInt;    // unboxing
```

- Autoboxing takes a primitive and puts it **inside** a wrapper object
- Unboxing takes the value **out** of a wrapper object and puts it into a primitive variable

Unboxing at the North Pole

- You rarely need to declare numeric wrapper class objects, but they can be useful when you need to work with primitives in a context where primitives are not permitted.
- For example…remember `ArrayList` only work with objects.

```
ArrayList<int> list = new ArrayList<int>(); // Error!
ArrayList<Integer> list = new ArrayList<Integer>(); // OK!
```

- Wrapper classes also have many useful methods detailed throughout the chapter if you have time to read about them!


Tokenizing
Strings

Concept

- Tokenizing a string is breaking a string down into its components, which are called *tokens*. The tokens in the string are separated by one or more *delimiters*.
- In the string, "Hamilton won every rap battle" there are:
- 5 tokens:
  - Hamilton
  - won
  - every
  - rap
  - battle
- And 1 delimiter:
  - The space character " "

String's split method

The String class has a method we can use to tokenize a string (break it down into its parts). It's called the split method, and works like this:

```
String str = "I'm looking for a mind at work!"; //Hamilton reference ☺

String[] tokens = str.split(" ");  //split the string on every space, and put each
                                   //token into a String array
```

To then print out all of the String tokens,

you can loop through the array:

```
for (String s : tokens)
    System.out.println(s);
```

Output:
I'm
looking
for
a
mind
at
work!

Delimiters

- A delimiter is a character that separates tokens.
- We saw that the space character is a typical delimiter, but you can use any character you like.
- Examples:
  - `String nums = "3,7,8,13,4,6,11";  //This delimiter would be a comma`
    `String[] tokens = nums.split(","); //split the string based on commas`
  - `String myDate = "04-01-2020";  //this delimiter would be a -`
    `String[] tokens = myDate.split("-"); //split the string based on dashes.`

Using more than one delimiter

- You can split a string on more than one delimiter if you like:

```
String email = "Mandy.M.Mcdonald@somecompany.com";
```

- You can split this into all tokens where there are two delimiters:

  . and @

```
String[] tokens = email.split("[@.]");
```

- To use more than one delimiter, you enclose the ones you want inside of square brackets
  - If you're interested, this is called a *regular expression (regex)*

Practice

Write a small program to print out all of the tokens in this specific string:

```
/Users/profClark/Desktop/awesome.jpg
```

You'll need multiple delimiters, and you'll need a loop to print all the tokens.

Answer

```
public class StringSplit
{
   public static void main(String[] args)
   {
      String path = "/Users/profClark/Desktop/awesome.jpg";
      String[] tokens = path.split("[/.]");
      for (String s : tokens)
        System.out.println(s);
   }
}
```

Let's go use .split with a large file...and learn about

Records!

CountriesData

CountriesRecords

projects in our GitHub examples folder

You finished Chapter 9! Looks like you've made it onto the Nice List!