Wrapper Classes and AutoBoxing

Introduction

- each of the primitive data types in Java have a Wrapper class
  - in primitive, the variable points directly to a single value (not an object)
- int, float, boolean, byte, char, double, etc…
- wrapper class named with a capital letters instead
  - Integer (wrapper class of int)
  - Float (wrapper class of float)
  - STRING DOES NOT HAVE A PRIMITIVE TYPE!!
- refresher on primitives
  - they are NOT Java objects
  - cannot be used in Collections
- Wrapper versions are used where Objects are required
  - as in Collections
- Wrapper classes are ***immutable***
  - will need to reset with "new" again

Primitives vs. Wrappers

ArrayList<int> newList = new ArrayList<int>(); // error
ArrayList< Integer > newList = new ArrayList<Integer>(); // same idea, but using Integer object
newList.add(new Integer(23));

Creating/Retrieving a Wrapper Instance value

- do not need new
- there are several ways of instantiating

Creating a Wrapper Instance

```
Integer intExample1 = Integer.valueOf(23);
Integer intExample2 = Integer.valueOf(“23”);
Integer intExample3 = new Integer(23);

Long longExample1 = Long.valueOf(65671263561);
Long longExample2 = Long.valueOf(Long.MAX_VALUE);

System.out.println(intExample1.intValue());
System.out.println(intExample1); // both will work because of unboxing

intExample3 = new Integer(40); // resetting immutable Integer value
```

ParseType() function

- if you don’t want to instantiate a full variable, shortcut to get a value
- (variables Scanner to types here)

Parsing a Wrapper

```
Integer fromArgs = Integer.parseInt(args[0]);
```

Wrapper Class features

- MAX_VALUE & MIN_VALUE value
  - sets the datatype to the maximum/minimum value possible
  - can be done during instantiation
- functions (Integer examples)
  - notice they accept a regular int
  - https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html
  - toBinaryString
  - toString
  - max(int a, int b)

Autoboxing and Unboxing

- automatic conversion of primitive data type to wrapper object
- and vise versa for unboxing
- helps in passing of primitive/wrapper pairs to functions
  - not need to code conversion
  - need to be the same related type
- autoboxing
  - putting an int into an Integer wrapper object
- unboxing
  - pulling an Integer and placing it into an int

Autoboxing/unboxing example

```
class Driver {

    public static void main(String[] args) {

       int intEx1 = 20;
       int intEx2 = 20;
       Integer wrapper1 = new Integer(10);
       Integer wrapper2 = new Integer(10);

       System.out.println(multiply(intEx1, intEx2));
       System.out.println(multiply(wrapper1, wrapper2));

    }

    public static int multiply(int a, int b)  { return a * b; }
}
```