Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-six)

6.2: Type Wrappers

As you may recall, Java treats objects differently from primitive types. Primitive method arguments are pass by value while object arguments are pass by reference.

Because Java is a strict typing language, using primitives as arguments are is restrictive. In order to take advantage of most of the JDK, we need to somehow create objects out of primitive types.

Enter type wrappers. Type wrappers or wrappers are classes that have corresponding primitive types. Wrappers make it easy to convert between primitive and object types. See below for a list of the most common primitive types and their wrapper equivalents.

Table: Wrapper Class Equivalents

Primitive Type	Wrapper Class Type
boolean	java.lang.Boolean
char	java.lang.Char
short	java.lang.Short
int	java.lang.Integer
float	java.lang.Float
long	java.lang.Long
double	java.lang.Double

To create wrappers, we encapsulate a primitive value within the appropriate type. This process is called **boxing**. Boxing involves the following format:

```
Type var = new Type(value);
```

Where *Type* is one of the wrapper classes listed above (note that you do not need the java.lang part as it is default). For example:

```
Double dbl = new Double(123.0);
```

value can also be a String type. The wrapper type constructor will automatically parse the string as in the following:

```
Double dbl = new Double("123.0");
```

The process of retrieving a primitive value from a Type Object is called **unboxing**. Unboxing uses the following format:

```
var.typeValue();
```

In our dbl example, we would use:

```
dbl.doubleValue()
```

However, it is cumbersome to have to write extra code if all you want to do is get a value when a primitive works just fine. This is where autoboxing comes in.

NOTE: As subclasses of Object, wrappers inherit Object class methods, which are useful in threading, introspection and other features. On the other hand, primitive types do not have the overhead of wrapper types, so use discretion in deciding whether to use primitives or Objects.