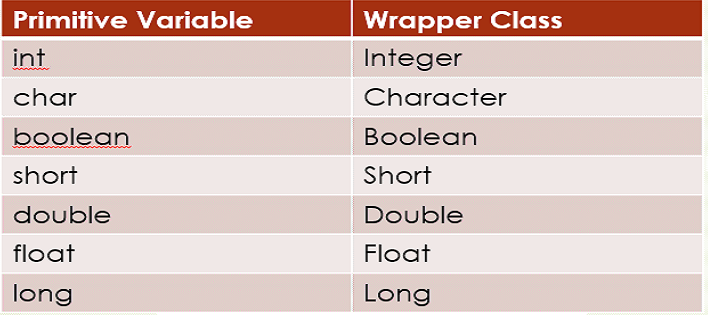
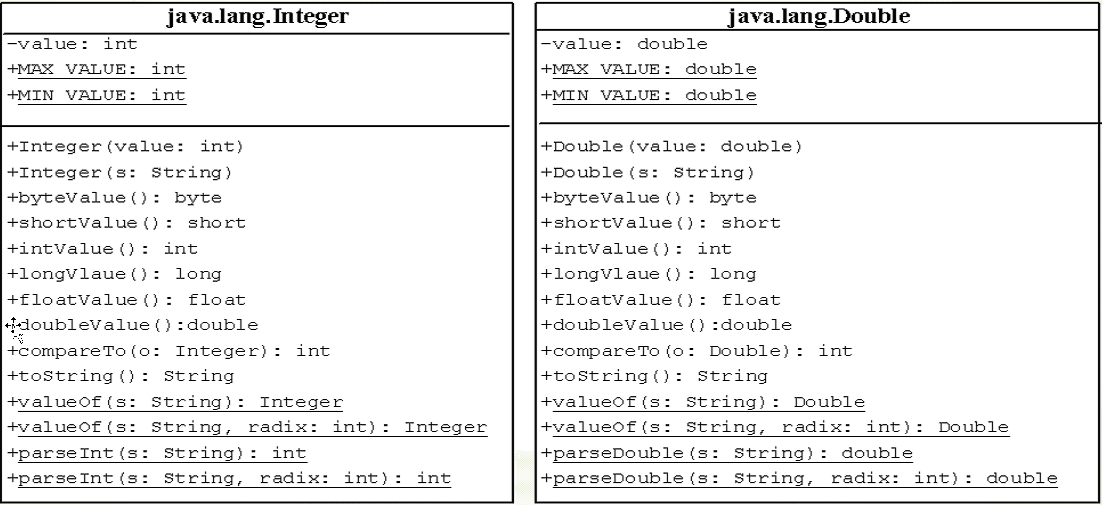
**Wrapper Classes**

* Some languages don’t have primitive data types
  + In these languages, everything is an object
  + Objects have methods associated with them and are generally more useful
* Java allows us to work with primitive data types acting like objects by use of something called a **Wrapper Class**
  + **Wrapper** comes from their ability to **“wrap”** themselves around primitive data types, and provide utilities similar to those of objects

**List of Wrapper Classes**



**The Integer and Double Classes**



* underline means Static, the attribute/method belongs to the **whole class**, not just the object

**Boxing and Unboxing**

* Wrapper classes provide functionality but around **JDK 1.5,** additional functionality such as **Boxing, and Unboxing,** were added
  + Java Development Kit version 1.5, we are at JDK 1.8
* Boxing and Unboxing lets us treat wrapper classes and primitives in the same ways
* **Boxing** refers to the process of taking a primitive variable and boxing it as a **Wrapper Class** automatically
  + No conversions, or method calls are required for this
* **Unboxing** refers to the process of taking a Wrapper Class object and using it like a primitive variable
  + Again, no conversions, or method calls are necessary for this

**String Class**

* Given a String sOne that has the value “Hello World”
* Find the Index of the first **“e”** in the string
  + sOne.indexOf(‘e’)
  + sOne.indexOf(“e”)
* Concatenate two strings, sOne and sTwo, together
  + sOne + sTwo
* Given the String sOne = “Hello World, Java is fun”
  + In one line of code, how can you remove the first word?
  + sOne = sOne.substring(sOne.indexOf(‘ ‘) + 1)
* Given the String sTeam = “Raptors,22,4”
  + Assign each of the 3 string values to a string arrow called Standings

String[] sStandings = sTeam.split(“,”)

* + - **Split** command will split up a string on a certain delimiter
* int nWin = Integer.parseInt(sStandings[1]);

**Construction of Strings**

* If you had an object oriented background in another language, you would expect to create string using the following syntax

***String sOne = new String(“Hello World”);***

* However, because we work with strings so often in Java, the language allows us to use the following to complete the above declaration

***String sOne = “Hello World”;***

* Strings are **immutable**, meaning they cannot be changed
* However, consider the following

***String sVal = “Java”;***

***sVal = “Python”;***

* + In the above scenario, sVal is being created as “Java”. When it’s reassigned “Python”, sVal is changing memory reference to the “Python” block in memory
    - “Java” still exists, just has no references to it anymore

**Interned String**

* Strings are immutable, but often, the Same Literal Pattern might be used in different Strings
  + “COSC180”
* To save space on the system, the Java Virtual Machine (JVM) will recognize duplicate string literals and reuse the same objects/space instead of creating new ones
  + Such an instance of a string is called **Interned**

***String sLang1 = “Java”***

***String sLang2 = new String(“Java”)***

***String sLang3 = “Java”***

* + sLang1 and sLang3 refer to the same memory location as JVM detects that they use the same literals
    - These are **interned**
  + sLang2 uses a different one as it was declared as a new object

**Regular Expressions**

* A concise specification for detecting patterns in Strings
* Regular expressions can be intimidating, but you can do a lot of pattern matching with minimal knowledge
* There are 3 Java string methods that work directly with regular expressions
  + Matches
  + replaceAll
  + Split

**Basics of Regular Expressions**

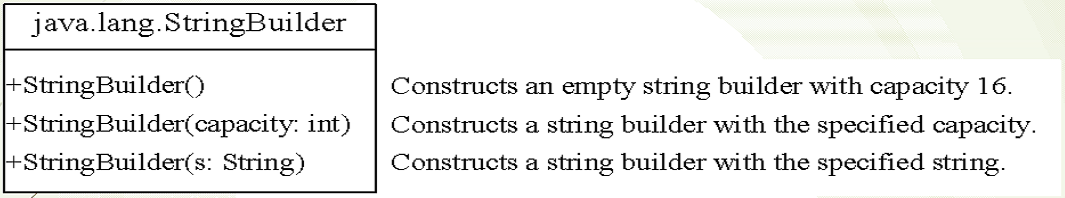
* The special character **“.”** matches any alphanumeric, or space character
* **Quantifiers** – how many times a pattern occurs

|  |  |
| --- | --- |
| Quantifier | Meaning |
| \* | 0, or more Times |
| + | 1, or more Times |
| ? | 0, or 1 Time |

**StringBuilder and StringBuffer Classes**

* **StringBuilder** and **StringBuffer** are alternatives to the String class
* They can be used wherever a string is used
* However, these are **mutable** objects
  + Their content can be changed
  + We can append to a string, insert into the string, replace patterns or characters within the string, but it still remains the same object
* **StringBuilder** is almost identical to **StringBuffer**, except that operations on the String are **synchronized**
  + Multiple running threads have to take turns when accessing the content of a **StringBuilder**

**StringBuilder Constructors**



**Modifying Strings in the Builder**

