INTRODUCTION TO STRINGS IN JAVA

A String is a sequence of characters (e.g. "Hello World").

A String is an object in java, and not a primitive.

Creating Strings

We can create a String object in two ways:

1. Assigning a String literal to a String variable

- e.g. **String greeting = "Hello world!";**
- Here, "Hello world!" is a string literal.
- Java keep only one copy of a string literal object and reuses them. This process is called String interning.
- In this approach, string objects are not created again and again, but reused.

2. Creating a String object using the new keyword and one of the String constructors like any other object

- e.g. **String greeting = new String("Hello world!");**
- In this approach, minimum one new String object is created every time.

♣ Whenever the new keyword is used, an object is created allocating memory from the heap.

♣ The String literal is also placed in the String pool if this string literal is used for first time. Thus if the String object is not already present in the pool two objects will be created in total.

Important Note!

1. It is preferred to use String literal (approach 1) when possible as it will reuse objects and won't create a new object every time.
2. In some cases it might be needed to use approach two (where you use the new keyword and a String constructor).

- The String class has thirteen constructors that allow you to provide the initial value using different sources, such as an array of characters.
- There are two constructors that accept the StringBuffer and StringBuilder classes.

Immutability of String objects

1. A String object is immutable; this means that once an Object is created it, cannot be changed.
2. If you call a function on the string object, it will return you the string object; but it will not change the string object.
1. However you may assign this returned value to the reference variable overriting the previous value.

Example: Calling methods on String object

```
String helloString = "Hello";

helloString.concat(" world");

System.out.println(helloString);
```

- This will print only Hello.
- The statement "*helloString.concat(" world");*" will create a new String object with content as "Hello World" and return it, but will not change the original String.
  To print "Hello world", you have to assign the concat result to the helloString reference variable as:

  helloString = helloString.concat(" world");.

StringBuffer and StringBuilder

- Any modification on a String object will create a new string object that contains the result of the operation.
  - Therefore, if you try to append 100 string literals to a String object, 100 String objects will be created, which is not a good thing.
  - Java hence provide two alternatives to String called StringBuffer and StringBuilder which will modify the current object and will not create a new object like String every time the object value is modified.

Important methods of String class

1. The **length()** method returns the number of characters contained in the string object.
2. The String class includes a **concat method** for concatenating two strings.
   - E.g. string1.concat(string2);

♣ This returns a new string that is string1 with string2 added to it at the end, but not that since a String object is immutable, it will not change the values of string1 or string2.

3. The **static format() method** allows you to create a formatted string that you can reuse, as opposed to a one-time print statement.
   - Example:
     - ♣ String fs;
     - ♣ fs = String.format("The value of the float variable is %f", floatVar);
     - ♣ System.out.println(fs);
   - You could also write the same as:
     - ♣ System.out.printf("The value of the float variable is %f", floatVar);
4. The **static valueOf method** will convert a number to a string:
   - E.g. String s1 = String.valueOf(777);
5. The **charAt(int index) method** returns the char value at the specified index. The index ranges from 0 to length() - 1.
   - **StringIndexOutOfBoundsException** is thrown by String methods to indicate that an index is either negative or greater than the size of the string.
     - ♣ For some methods such as the charAt method, this exception also is thrown when the index is equal to the size of the string.
   - Example:
     - ♣ String str = "Hello World";
     - ♣ System.out.println(str.charAt(1));
     - ♣ str.charAt(1) will print the second element, which is e.
     - ♣ System.out.println(str.charAt(11));
     - ♣ str.charAt(11) will throw a StringIndexOutOfBoundsException as the size of the string is 11 and str.charAt(11) refers to the 12th element.
     - ♣ Note that the index ranges from 0 to length() - 1.
6. String class overrides the **equals method of the Object class**.
   - The equals method see if two String objects has the same value whereas == checks if two String objects are actually referring to same memory location.
   - Example
     - ♣ String h1 = "Java";
     - ♣ String h2 = new String ("Java");

- ♣ System.out.println(h1==h2);
- ♣ This will print false and second print statement will print true. Refer to String interning in Java for more details.
- ♣ System.out.println(h1.equals(h2);
  - The **equals** method is **case sensitive**. String contains a method **equalsIgnoreCase()** which is similar to equals, but compares value **case insensitively**.

Chaining

Most methods of String class (e.g. concat) return a new string object; we can call any of the String methods on that new object returned through chaining.

Example: Chaining of string methods that return a string object

```
String s = new String("Hello").concat("World").concat("Program");

System.out.println(s);
```

This will print:

```
Hello World Program
```

- Even with Chaining we have to assign the final object to a reference or the changes will not be saved (as String objects are immutable).

Collator class

- The Collator class can be used to manipulate strings in a locale-specific manner removing the problems of different internal string representations.
- For instance, many languages use the accent to differentiate or emphasize a character. If these different representations are compared using the equals method, the method would return false.

Examples

Example: Empty String

Find the output:

```
String s = "HelloWorld";

if(s.startsWith(""))

  System.out.println("Strings in java start with an empty string");
```

```
else
  System.out.println("Strings in java DOES NOT start with an empty string");
```

- This will print:
  - Strings in java start with an empty string

Example: Empty String check with == and equals
Find the output:

```
String s=new String();
if(s =="")
  System.out.println("==");
if(s.equals(""))
  System.out.println(".equals");
```

- This will print:
  - .equals
- Always use .equals() for comparison. == may return false even if two strings contain same value. This can happen even for empty strings.

Example: String concatenation
Find the output:

```
System.out.println(5 + 3);
System.out.println("Hello"+5 + 3);
System.out.println(5 + 3 + "Hello");
```

- This will print:
  - 8
  - Hello53
  - 8Hello

- The + operator is overloaded.
- When a string is one of the operand for a + operator, string concatenation happens.
- In the second statement "Hello" + 5 gives a new string "Hello5" and then it is again concatenated to 3 to get "Hello53".
- To get Hello8, you should use parenthesis as System.out.println("Hello"+(5+3));

  Source : http://javajee.com/introduction-to-strings-in-java