**Java String – String Functions In Java With Examples**

String is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

There are two ways to create a String object:

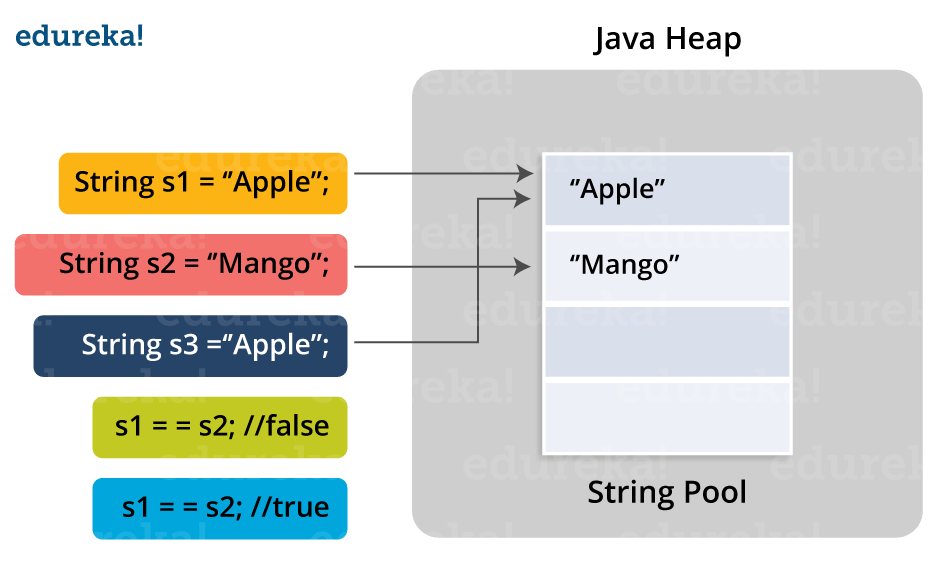
1. **By string literal** : Java String literal is created by using double quotes.  
   For Example: String s=“Welcome”;
2. **By new keyword** : Java String is created by using a keyword “new”.

**For example** : String s=new String(“Welcome”);

It creates two objects (in String pool and in heap) and one reference variable where the variable ‘s’ will refer to the object in the heap.

Now, let us understand the concept of Java String pool.

**Java String Pool:** Java String pool refers to collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned. Refer to the diagrammatic representation for better understanding:



In the above image, two Strings are created using literal i.e “Apple” and “Mango”. Now, when third String is created with the value “Apple”, instead of creating a new object, the already present object reference is returned. That’s the reason Java String pool came into the picture.

Before we go ahead, One key point I would like to add that unlike other data types in Java, Strings are immutable. By immutable, we mean that Strings are constant, their values cannot be changed after they are created. Because String objects are immutable, they can be shared. For example:

  String str =”abc”;  
 is equivalent to

char data[] = {‘a’, ‘b’, ‘c’};  
    String str = new String(data);

Let us now look at some of the inbuilt methods in String class.

## ****Java String Methods****

* **Java String length()**: The Java String length() method tells the length of the string. It returns count of total number of characters present in the String. For example:

public class Example{

public static void main(String args[]){

String s1="hello";

String s2="whatsup";

System.out.println("string length is: "+s1.length());

System.out.println("string length is: "+s2.length());

}}

Here, String length()  function will return the length 5 for s1 and 7 for s2 respectively.

* **Java String compareTo()**: The Java String compareTo() method compares the given string with current string. It is a method of ‘Comparable’ interface which is implemented by String class. Don’t worry, we will be learning about String interfaces later. It either returns positive number, negative number or 0. For example:

public class CompareToExample{

public static void main(String args[]){

String s1="hello";

String s2="hello";

String s3="hemlo";

String s4="flag";

System.out.println(s1.compareTo(s2)); // 0 because both are equal

System.out.println(s1.compareTo(s3)); //-1 because "l" is only one time lower than "m"

System.out.println(s1.compareTo(s4)); // 2 because "h" is 2 times greater than "f"

}

}

This program shows the comparison between the various string. It is noticed that    
if s1 > s2, it returns a positive number    
if s1 < s2, it returns a negative number   
if s1 == s2, it returns 0

* **Java String concat() :** The Java String concat() method combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string. For example:

public class ConcatExample{

public static void main(String args[]){

String s1="hello";

s1=s1.concat("how are you");

System.out.println(s1);

}}

|  |  |
| --- | --- |
|  |  |

* The above code returns “hellohow are you”.
* **Java String IsEmpty()** : This method checks whether the String contains anything or not. If the java String is Empty, it returns true else false. For example:

public class IsEmptyExample{

public static void main(String args[]){

String s1="";

String s2="hello";

System.out.println(s1.isEmpty());      // true

System.out.println(s2.isEmpty());      // false

}}

|  |  |
| --- | --- |
|  |  |

* **Java String Trim()** : The java string trim() method removes the leading and trailing spaces. It checks the unicode value of space character (‘\u0020’) before and after the string. If it exists, then removes the spaces and return the omitted string. For example:

public class StringTrimExample{

public static void main(String args[]){

String s1="  hello   ";

System.out.println(s1+"how are you");        // without trim()

System.out.println(s1.trim()+"how are you"); // with trim()

}

}

|  |  |
| --- | --- |
|  |  |

* In the above code, the first print statement will print “hello how are you” while the second statement will print “hellohow are you” using the trim() function.
* **Java String toLowerCase()** : The java string toLowerCase() method converts all the characters of the String to lower case. For example:

public class StringLowerExample{

public static void main(String args[]){

String s1="HELLO HOW Are You?”;

String s1lower=s1.toLowerCase();

System.out.println(s1lower);

}

}

|  |  |
| --- | --- |
|  |  |

The above code will return “hello how are you”.

* **Java String toUpper()** : The Java String toUpperCase() method converts all the characters of the String to upper case. For example:

public class StringUpperExample{

public static void main(String args[]){

String s1="hello how are you";

String s1upper=s1.toUpperCase();

System.out.println(s1upper);

}}

|  |  |
| --- | --- |
|  |  |

The above code will return “HELLO HOW ARE YOU”.

* **Java String ValueOf()**: This method converts different types of values into string.Using this method, you can convert int to string, long to string, Boolean to string, character to string, float to string, double to string, object to string and char array to string. The signature or syntax of string valueOf() method is given below:

public static String valueOf(boolean b)  
public static String valueOf(char c)  
public static String valueOf(char[] c)  
public static String valueOf(int i)  
public static String valueOf(long l)  
public static String valueOf(float f)  
public static String valueOf(double d)  
public static String valueOf(Object o)

Let’s understand this with a programmatic example:

|  |
| --- |
| public class StringValueOfExample{  public static void main(String args[]){    int value=20;  String s1=String.valueOf(value);  System.out.println(s1+17);       //concatenating string with 20    }  } |

In the above code, it concatenates the Java String and gives the output – 2017.

* **Java String replace()**: The Java String replace() method returns a string, replacing all the old characters or CharSequence to new characters. There are 2 ways to replace methods in a Java String.

|  |
| --- |
| public class ReplaceExample1{  public static void main(String args[]){    String s1="hello how are you";  String replaceString=s1.replace('h','t');  System.out.println(replaceString);  }} |

* In the above code, it will replace all the occurrences of ‘h’ to ‘t’. Output to the above code will be “tello tow are you”. Let’s see the another type of using replace method in java string:

**Java String replace(CharSequence target, CharSequence replacement) method** :

|  |
| --- |
| public class ReplaceExample2{  public static void main(String args[]){  String s1="Hey, welcome to SSVPS";  String replaceString=s1.replace("SSVPS","Brainforce");  System.out.println(replaceString);  }  } |

* In the above code, it will replace all occurrences of “SSVPS” to “Brainforce”. Therefore, the output would be “ Hey, welcome to Brainforce”.
* **Java String contains()** :The java string contains() method searches the sequence of characters in the string. If the sequences of characters are found, then it returns true otherwise returns false. For example:

|  |
| --- |
| class ContainsExample{  public static void main(String args[]){    String name=" hello how are you doing?";  System.out.println(name.contains("how are you"));  // returns true  System.out.println(name.contains("hello"));        // returns true  System.out.println(name.contains("fine"));         // returns false  }  } |

* In the above code, the first two statements will return true as it matches the String whereas the second print statement will return false because the characters are not present in the string.
* **Java String equals()** : The Java String equals() method compares the two given strings on the basis of content of the string i.e Java String representation. If all the characters are matched, it returns true else it will return false. For example:

|  |
| --- |
| public class EqualsExample{  public static void main(String args[]){  String s1="hello";  String s2="hello";  String s3="hi";  System.out.println(s1.equals(s2));   // returns true  System.out.println(s1.equalsIgnoreCase(s3));   // returns false  }} |

* **Java** **String equalsIgnoreCase():** This method compares two string on the basis of content but it does not check the case like equals() method. In this method, if the characters match, it returns true else false. For example:

|  |
| --- |
| public class EqualsIgnoreCaseExample{  public static void main(String args[]){  String s1="hello";  String s2="HELLO";  String s3="hi";  System.out.println(s1.equalsIgnoreCase(s2));   // returns true  System.out.println(s1.equalsIgnoreCase(s3));   // returns false  }} |

* In the above code, the first statement will return true because the content is same irrespective of the case. Then, in the second print statement will return false as the content doesn’t match in the respective strings.
* **Java String toCharArray():** This method converts the string into a character array i.e first it will calculate the length of the given Java String including spaces and then create an array of char type with the same content. For example:

|  |
| --- |
| public class StringToCharArrayExample{  public static void main(String args[]){  String s1="Welcome to SSVPS";  char[] ch=s1.toCharArray();  for(int i=0;i<ch.length;i++)  {  System.out.print(ch[i]);  }  }  } |

* The above code will return “Welcome to SSVPS”.
* **Java StringGetBytes()** : The Java string getBytes() method returns the sequence of bytes or you can say the byte array of the string. For example:

|  |
| --- |
| public class StringGetBytesExample {    public static void main(String args[]){    String s1="ABC";  byte[] b=s1.getBytes();  for(int i=0;i<b.length;i++)  {  System.out.println(b[i]);  }  }  } |

* In the above code, it will return the value 65,66,67.
* **Java String IsEmpty()** : This method checks whether the String is empty or not. If the length of the String is 0, it returns true else false. For example:

|  |
| --- |
| public class IsEmptyExample{  public static void main(String args[]) {    String s1="";  String s2="hello";  System.out.prinltn(s1.isEmpty());     // returns true  System.out.prinltn(s2.isEmpty());     // returns false  }  } |

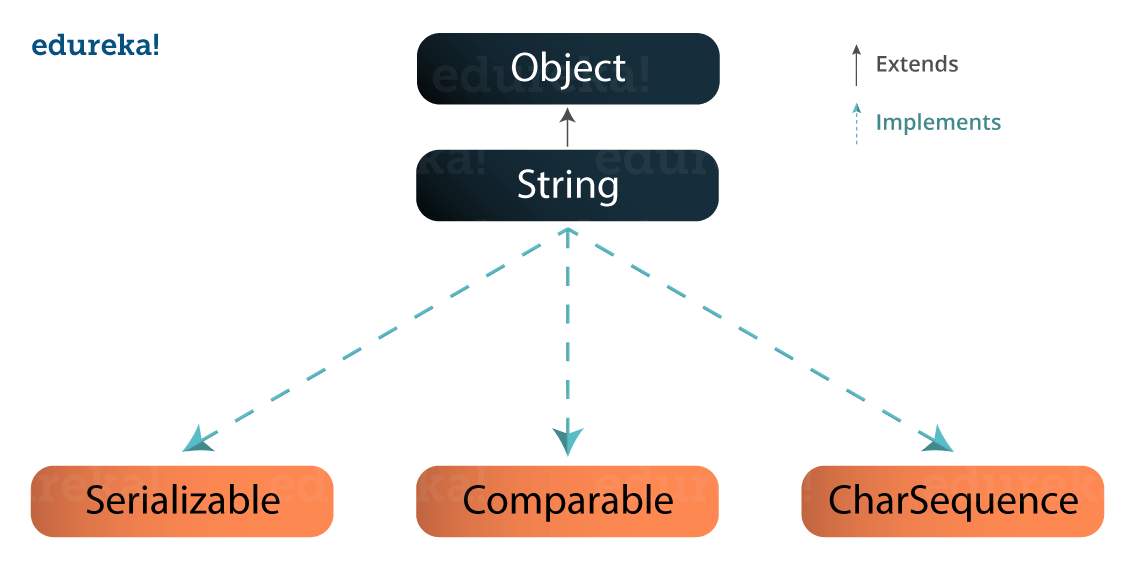
In the above code, the first print statement will return true as it does not contain anything while the second print statement will return false.

* **Java String endsWith()** : The Java String endsWith() method checks if this string ends with the given suffix. If it returns with the given suffix, it will return true else returns false. For example:

|  |
| --- |
| public class EndsWithExample{  public static void main(String args[]) {    String s1="hello how are you”;  System.out.println(s1.endsWith("u"));       // returns true  System.out.println(s1.endsWith("you"));     // returns true  System.out.println(s1.endsWith("how"));     // returns false  }  } |

This is not the end. There are more Java String methods that will help you make your code simpler.

Moving on, Java String class implements three **interfaces**, namely – Serializable, Comparable and CharSequence.

Since, Java String is immutable and final, so a new String is created whenever we do String manipulation. As String manipulations are resource consuming, Java provides two utility classes: StringBuffer and StringBuilder.  
Let us understand the difference between these two utility classes:

* StringBuffer and StringBuilder are mutable classes. StringBuffer operations are thread-safe and sync
* hronized whereas StringBuilder operations are not thread-safe.
* StringBuffer is to be used when multiple threads are working on same String and StringBuilder in the single threaded environment.
* StringBuilder performance is faster when compared to StringBuffer because of no overhead of synchronized.