Java String

Java String

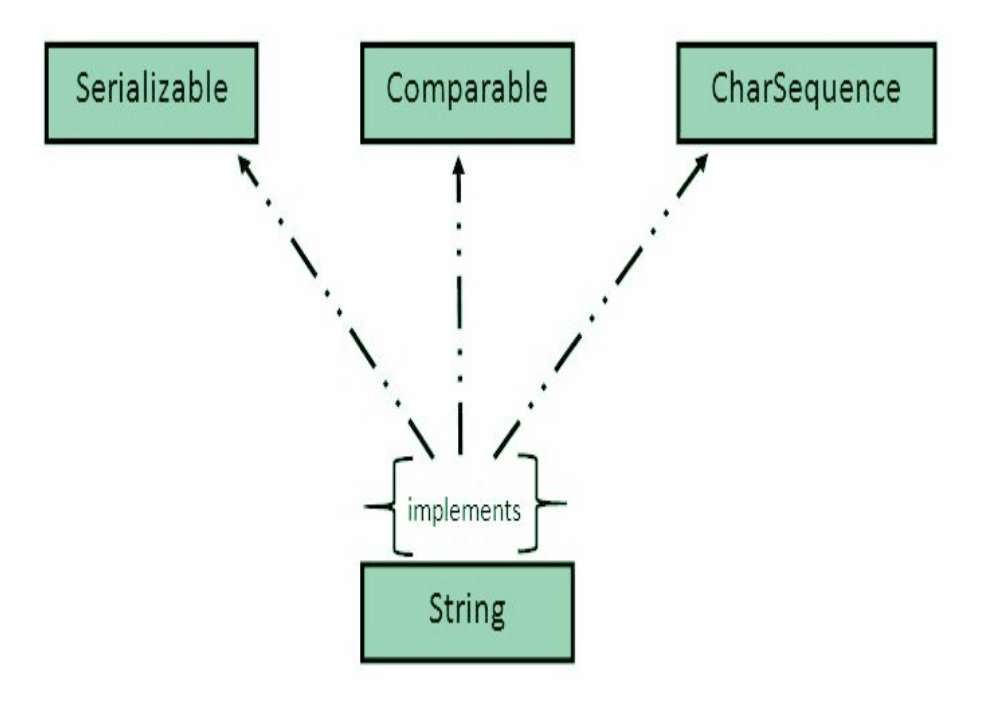
In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string.

String

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

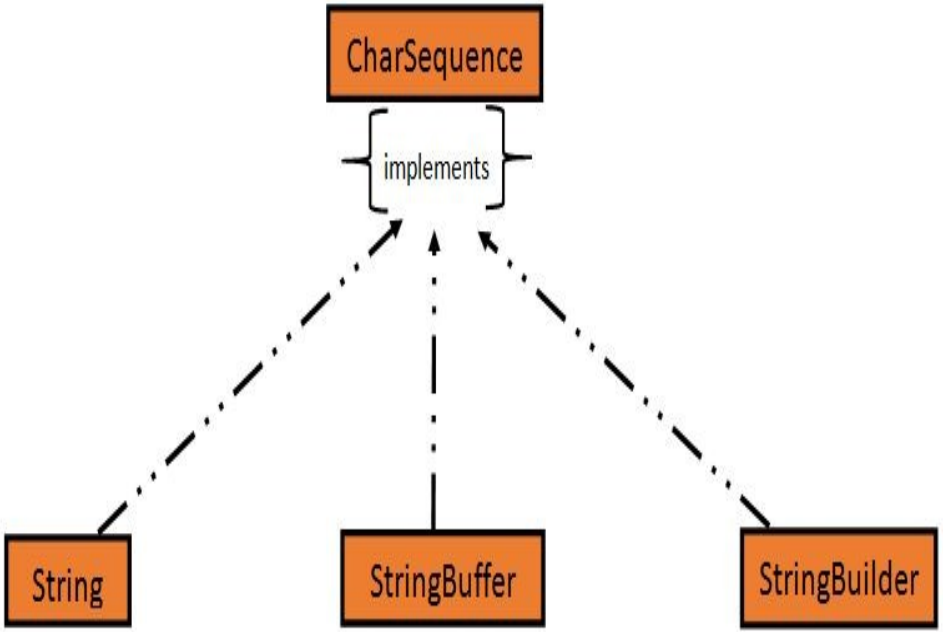
One important thing to notice about string object is that string objects are immutable that means once a string object is created it cannot be changed.

String



The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.

CharSequence Interface is used for representing a sequence of characters.



The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in Java by using these three classes.

The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

Creation of String

There are two ways to create String object:

- By string literal

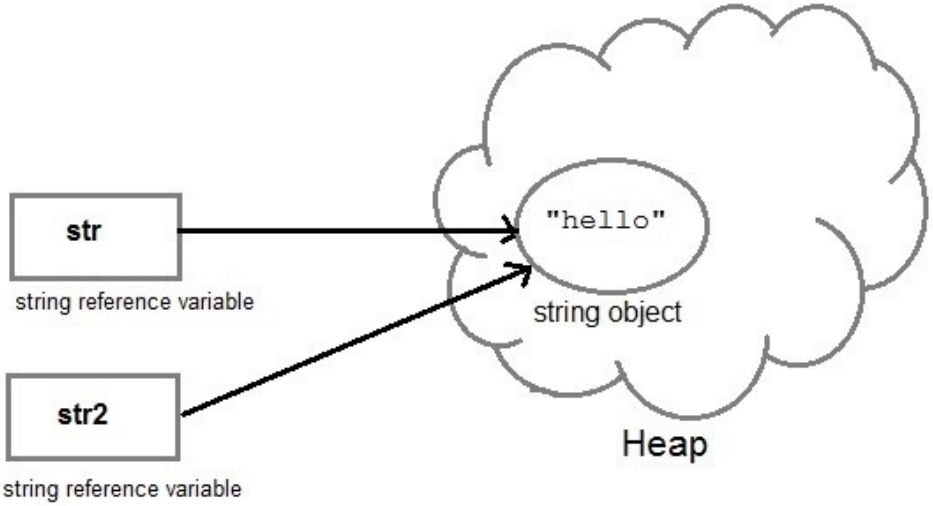
  ```
  String s1 = "Hello DDU";
  ```

- By new keyword

  ```
  String s1 = new String("Hello DDU");
  ```

String Literals

- String objects are stored in a special memory area known as string constant pool inside the heap memory.
- Each time you create a string literal, the JVM checks the "string constant pool" first.
- If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.




```
String s1="hello";
String s2="hello";//It doesn't create a new instance
//String s2 = s1
```

String Object

String s=new String("hello");

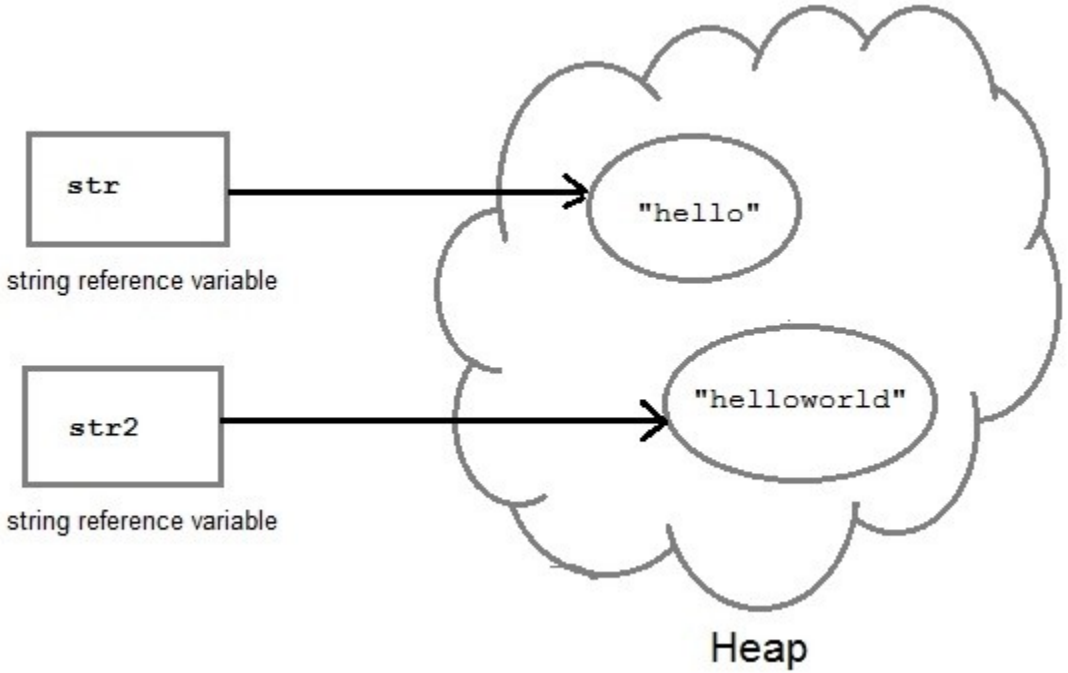
In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "hello" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Change in String will Change the reference

```
String str= "hello";

String str2 = str;

str2=str2.concat("world")
;
```



Immutable object

- An object whose state cannot be changed after it is created is known as an Immutable object.
- In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.
- Once string object is created its data or state can't be changed but a new string object is created.

```
String s="DharmsinhDesaiUniversity";

s.concat(“MCADepartment");//concat() method appends the string at the end

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

Why immutable?

- Java uses the concept of string literal.
- Suppose there are 5 reference variables,all referes to one object "DDU".
- If one reference variable changes the value of the object, it will be affected to all the reference variables.
- That is why string objects are immutable in java.

String compare

We can compare string in java on the basis of content and reference.

- authentication (by equals() method)
- sorting (by compareTo() method)
- reference matching (by == operator)

compare by equals()

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

public boolean equals(Object obj)

//compares this string to the specified object.

public boolean equalsIgnoreCase(String str)

//compares this String to another string, ignoring case.

compare by == operator

The = = operator compares references not values.

```
String s1="DDU";
String s2="DDU";
String s3=new String("DDU");
System.out.println(s1==s2); //true (because both refer to same instance)
System.out.println(s1==s3); //false(because s3 refers to instance created in nonpool)
```

compare by compareTo()

The String compareTo() method compares values lexicographically (alphabetical order.) and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

- s1 == s2 :0
- s1 > s2 :positive value
- s1 < s2 :negative value

String Class Methods

charAt(int) Returns the character at the specified index.

compareTo(String) Compares this String to another specified String.

concat(String) Concatenates the specified string to the end of this String.

copyValueOf(char[], int, int) Returns a String that is equivalent to the specified character array.

copyValueOf(char[]) Returns a String that is equivalent to the specified character array.

endsWith(String) Determines whether the String ends with some suffix.

equals(Object) Compares this String to the specified object.

equalsIgnoreCase(String) Compares this String to another object.

String Class Methods

getChars(int, int, char[], int) Copies characters from this String into the specified character array.

indexOf(String) Returns the index within this String of the first occurrence of the specified substring.

indexOf(String, int) Returns the index within this String of the first occurrence of the specified substring.

lastIndexOf(int, int) Returns the index within this String of the last occurrence of the specified character.

lastIndexOf(String) Returns the index within this String of the last occurrence of the specified substring.

length() Returns the length of the String.

String Class Methods

replace(char, char) Converts this String by replacing all occurences of oldChar with newChar.

startsWith(String) Determines whether this String starts with some prefix.

toCharArray() Converts this String to a character array.

toLowerCase() Converts all of the characters in this String to lower case.

toString() Converts this String to a String.

toUpperCase() Converts all of the characters in this String to upper case.

trim() Trims leading and trailing whitespace from this String.

valueOf(Object) Returns a String that represents the String value of the object.

```java
import java.util.Scanner;

class L3_3_String6
{
 public static void main(String args[])
 {
 char[] ch=new char[10];
 Scanner input=new Scanner(System.in);

 System.out.print("Enter first string = ");
 String strObj1=new String(input.nextLine());
 System.out.print("Enter second string =");
 String strObj2=new String(input.nextLine());
 System.out.println(strObj1);
 System.out.println(strObj2);

 System.out.println("public int compareTo(String anotherString) : "+ strObj1.compareTo(strObj2));
 System.out.println("public int compareToIgnoreCase(String str) : "+ strObj1.compareToIgnoreCase(strObj2));
 System.out.println("public int indexOf(int ch) : "+strObj1.indexOf("A"));
 System.out.println("public int lastIndexOf(int ch) : "+strObj1.lastIndexOf("K"));
 System.out.println("public boolean isEmpty() :"+strObj1.isEmpty());
 System.out.println("public char charAt(int index) :"+strObj1.charAt(3));
 System.out.println("public boolean equals(Object anObject) :"+strObj1.equals(strObj2));
 System.out.println("public boolean startsWith(String prefix) : "+strObj1.startsWith("Aj"));
 System.out.println("public boolean endsWith(String suffix) : "+strObj1.endsWith("y"));
 System.out.println("public int hashCode() : "+strObj1.hashCode());
 System.out.println("public String substring(int beginIndex) :"+strObj1.substring(2));
 }
}
```

```
Enter first string = vivek
Enter second string =vivek
vivek
vivek
public int compareTo(String anotherString) : 0
public int compareToIgnoreCase(String str) : 0
public int indexOf(int ch) : -1
public int lastIndexOf(int ch) : -1
public boolean isEmpty() :false
public char charAt(int index) :e
public boolean equals(Object anObject) :true
public boolean startsWith(String prefix) : false
public boolean endsWith(String suffix) : false
public int hashCode() : 112220169
public String substring(int beginIndex) :vek
Press any key to continue . . .
```

StringBuffer class

StringBuffer class

StringBuffer class is used to create a mutable string object. It means, it can be changed after it is created. It represents growable and writable character sequence.

StringBuffer()
//creates an empty string buffer with the initial capacity of 16.

StringBuffer(String str)
//creates a string buffer with the specified string.

StringBuffer(int capacity)
//creates an empty string buffer with the specified capacity as length.

StringBuffer(charSequence []ch):
//It creates a stringbuffer object from the charsequence array.

- append(Object) - Appends an object to the end of this buffer.
- capacity() - Returns the current capacity of the String buffer.
- charAt(int) - Returns the character at the specified index.
- ensureCapacity(int) - Ensures that the capacity of the buffer is at least equal to the specified minimum.

  StringBuffer stringBuffer = new StringBuffer(“Hello World”);

  System.out.println(stringBuffer.length()); //11

  System.out.println(stringBuffer.capacity()); //27

- getChars(int, int, char[], int) - Copies the characters of the specified substring (determined by srcBegin and srcEnd) into the character array, starting at the array's dstBegin location.
- insert(int Object) - Inserts an object into the String buffer.
- length() - Returns the length (character count) of the buffer.
- setCharAt(int, char) - Changes the character at the specified index to be ch.
- toString() - Converts to a String representing the data in the buffer

```
class StringBufferExample
{
    public static void main(String args[]){
    StringBuffer sb=new StringBuffer("Hello ");

    sb.append("Java");//now original string is changed
    System.out.println(sb);//prints Hello Java

    sb.insert(1,"Java");//now original string is changed
    System.out.println(sb);//prints HJavaello

    sb.replace(1,3,"Java");
    System.out.println(sb);//prints HJavavaello Java

    sb.delete(1,3);
    System.out.println(sb);//prints Hvavaello Java

    sb.reverse();
    System.out.println(sb);//prints avaJ olleavavH
}
}
```

StringBuilder class

- StringBuilder is identical to StringBuffer except for one important difference that it is not synchronized, which means it is not thread safe.

- StringBuilder also used for creating string object that is mutable and non synchronized. The StringBuilder class provides no guarantee of synchronization. StringBuffer and StringBuilder both are mutable but if synchronization is not required then it is recommend to use StringBuilder class.

StringBuffer class	StringBuilder class
StringBuffer is synchronized.	StringBuilder is not synchronized.
Because of synchronisation, StringBuffer operation is slower than StringBuilder.	StringBuilder operates faster.
StringBuffer is thread-safe	StringBuilder is not thread-safe
StringBuffer is less efficient as compare to StringBuilder	StringBuilder is more efficient as compared to StringBuffer.
Its storage area is in the heap	Its storage area is the stack
It is mutable	It is mutable
Methods are synchronized	Methods are not synchronized
It is alternative of string class	It is more flexible as compared to the string class

Introduced in Java 1.0	Introduced in Java 1.5
Its performance is moderate	Its performance is very high
It consumes more memory	It consumes less memory

```java
class StringBuilderExample
{
    public static void main(String args[])
    {
    StringBuilder sb=new StringBuilder();
    System.out.println(sb.capacity());//default 16

    sb.append("Hello");
    System.out.println(sb.capacity());//now 16

    sb.append("Java is a language");
    System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2

    }
}
```