Unit III: Java as Object Oriented Programming Language- Overview

Fundamentals of JAVA, Arrays: one dimensional array, multi-dimensional array, alternative array declaration statements,

String Handling: String class methods

Classes and Methods: class fundamentals, declaring objects, assigning object reference variables, adding methods to a class, returning a value, constructors, this keyword, garbage collection, finalize() method, overloading methods, argument passing, object as parameter, returning objects, access control, static, final, nested and inner classes, command line arguments, variable - length arguments.

Introduction to Java String Handling

- String is an **object** that represents sequence of characters. In Java, String is represented by String class which is located into java.lang.
- Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.
- The Java platform provides the String class to create and manipulate strings.
- The most direct way to create a string is to write −

String greeting = "Hello world!";

- Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

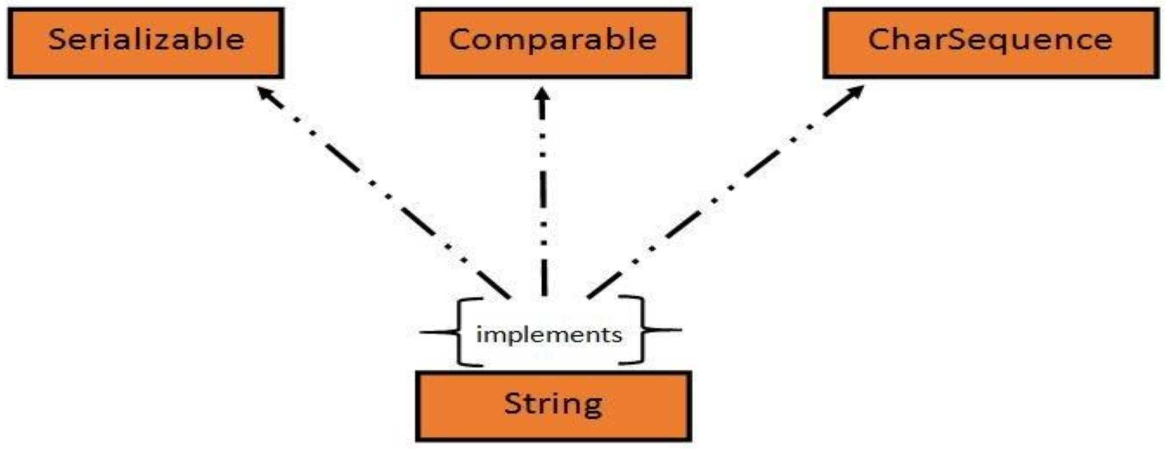
  Example

  ```
  public class StringDemo {
     public static void main(String args[]) {
        char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
        String helloString = new String(helloArray);
        System.out.println( helloString );
     }
  }
  ```

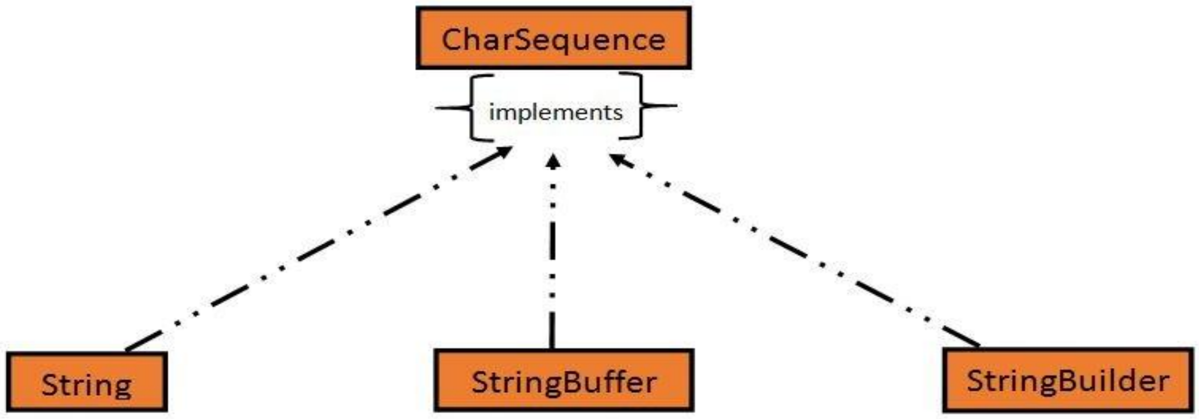
  Output

  hello.
- It is probably the most commonly used class in java library. In java, every string that we create is actually an object of type **String**.
- 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.

- The Java String class implements Serializable, Comparable and CharSequence interface that we have represented using the below image.



- In Java, **CharSequence** Interface is used for representing a sequence of characters. CharSequence interface is implemented by String, StringBuffer and StringBuilder classes. This three classes can be used for creating strings in java.



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.

What is an Immutable object?

An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper classes objects are immutable.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) Using a String literal

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

```
public class Demo{
   public static void main(String[] args) {
        String s1 = "Hello Java";
        System.out.println(s1);
   }
}
```

Output

```
Hello Java
```

2) Using new Keyword

We can create a new string object by using new operator that allocates memory for the object.

```
public class Demo{

   public static void main(String[] args) {

        String s1 = new String("Hello Java");

        System.out.println(s1);

   }

}
```

Output

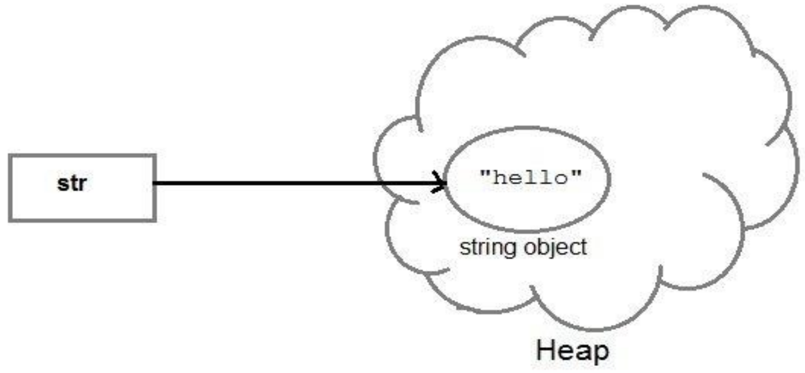
```
Hello Java
```

- Each time we create a String literal, the JVM checks the string pool first.
- If the string literal already exists in the pool, a reference to the pool instance is returned.
- If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as **string constant pool** inside the heap memory.

String object and how they are stored:

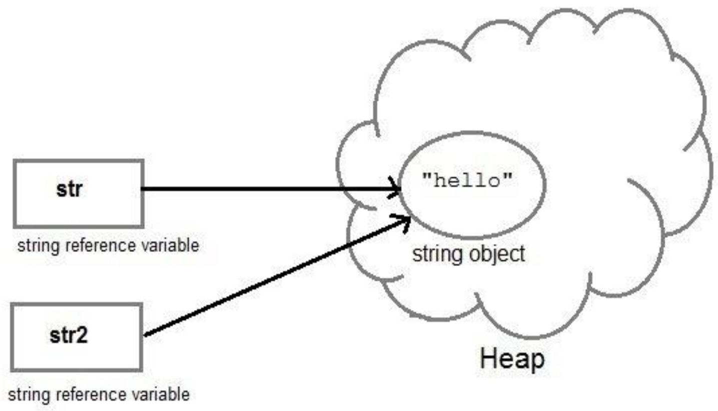
When we create a new string object using string literal, that string literal is added to the string pool, if it is not present there already.

```
String str= "Hello";
```



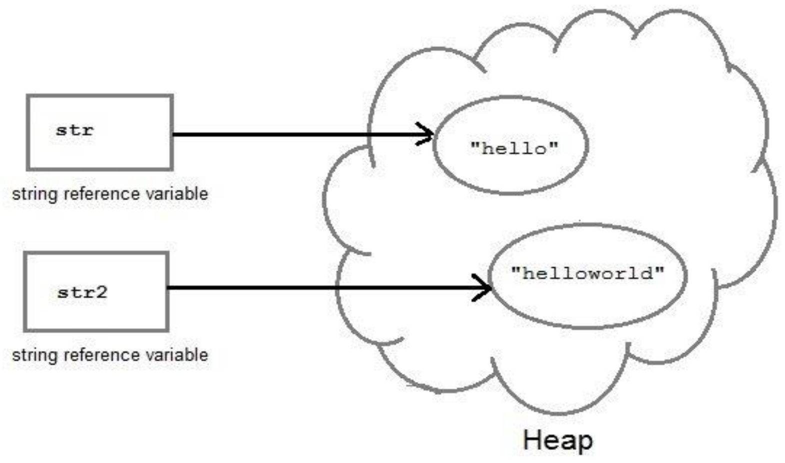
And, when we create another object with same string, then a reference of the string literal already present in string pool is returned.

```
String str2 = str;
```



But if we change the new string, its reference gets modified.

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



Concatenating String

There are 2 methods to concatenate two or more string.

1. Using **concat()** method
2. Using + operator

1) Using concat() method

- Concat() method is used to add two or more string into a single string object. It is string class method and returns a string object.

```
public class Demo{
   public static void main(String[] args) {
        String s = "Hello";
        String str = "Java";
        String str1 = s.concat(str);
        System.out.println(str1);   }      }
```

Output

HelloJava

2) Using + operator

Java uses "+" operator to concatenate two string objects into single one. It can also concatenate numeric value with string object. See the below example.

```
public class Demo{
   public static void main(String[] args) {
        String s = "Hello";
        String str = "Java";
        String str1 = s+str;
        String str2 = "Java"+11;
        System.out.println(str1);
        System.out.println(str2);
   }
}
```

Output

```
HelloJava
Java11
```

String Comparison

To compare string objects, Java provides methods and operators both. So we can compare string in following three ways.

1. Using equals() method
2. Using = = operator
3. By CompareTo() method

1. Using equals() method

- equals() method compares two strings for equality. Its general syntax is,

```
boolean equals (Object str)
```

Example

It compares the content of the strings. It will return **true** if string matches, else returns **false**.

```
public class Demo{
   public static void main(String[] args) {
        String s = "Hell";
        String s1 = "Hello";
        String s2 = "Hello";
        boolean b = s1.equals(s2);    //true
        System.out.println(b);
        b =s.equals(s1) ;   //false
        System.out.println(b);
   }
}
```

Output:
true
false

2. Using == operator

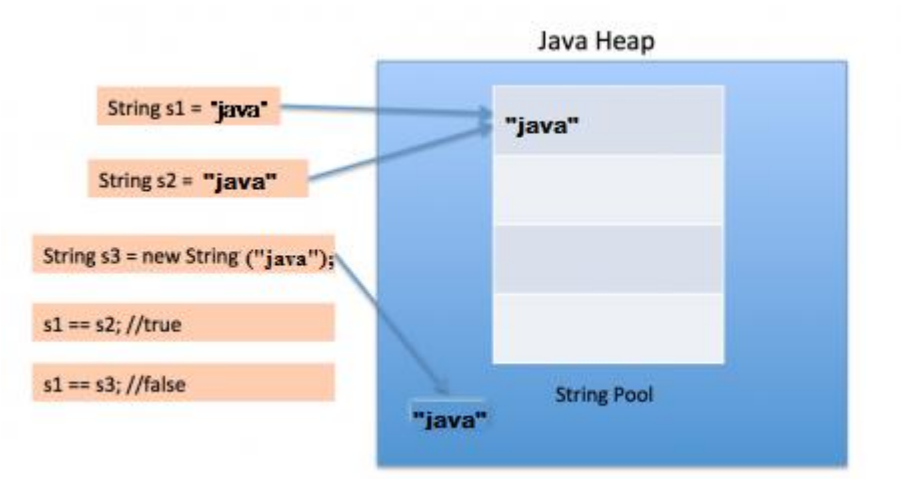
The double equal (==) operator compares two object references to check whether they refer to same instance. This also, will return **true** on successful match else returns false.

```
public class Demo{
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java";
        String s3 = new String ("Java");
        boolean b = (s1 == s2);     //true
        System.out.println(b);
        b =    (s1 == s3);      //false
        System.out.println(b);
    }                                              }
```

Output:
true
false

Explanation

- We are creating a new object using new operator, and thus it gets created in a non-pool memory area of the heap.
- s1 is pointing to the String in string pool while s3 is pointing to the String in heap and hence, when we compare s1 and s3, the answer is false.
- The following image will explain it more clearly.



3. By compareTo() method

- String compareTo() method compares values and returns an integer value which tells if the string compared is less than, equal to or greater than the other string.
- It compares the String based on natural ordering i.e alphabetically. Its general syntax is.

Syntax:

```
int compareTo(String str)
```

Example:

```
public class HelloWorld{
   public static void main(String[] args) {
          String s1 = "Abhi";
          String s2 = "Viraaj";
          String s3 = "Abhi";
          int a = s1.compareTo(s2);    //return -21 because s1 < s2
          System.out.println(a);
          a = s1.compareTo(s3);    //return 0 because s1 == s3
          System.out.println(a);
          a = s2.compareTo(s1);    //return 21 because s2 > s1
          System.out.println(a);
   }
}
```

Output:

```
-21
0
21
```

Java String class functions

The methods specified below are some of the most commonly used methods of the String class in Java. We will learn about each method with help of small code examples for better understanding.

1. charAt() method

String charAt() function returns the character located at the specified index.

```
public class Demo {
   public static void main(String[] args) {
      String str = "Computer";
      System.out.println(str.charAt(2));
   }
}
```
Output: m

NOTE: Index of a String starts from 0, hence str.charAt(2) means third character of the String str.

2. equalsIgnoreCase() method

String equalsIgnoreCase() determines the equality of two Strings, ignoring their case (upper or lower case doesn't matter with this method).

```
public class Demo {
   public static void main(String[] args) {
      String str = "java";
      System.out.println(str.equalsIgnoreCase("JAVA"));
   }
}
```

```
Output:

true
```

3. indexOf() method

String indexOf() method returns the index of first occurrence of a substring or a character. indexOf() method has four override methods:

- int indexOf(String str): It returns the index within this string of the first occurrence of the specified substring.
- int indexOf(int ch, int fromIndex): It returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
- int indexOf(int ch): It returns the index within this string of the first occurrence of the specified character.
- int indexOf(String str, int fromIndex): It returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

Example:

```
public class StudyTonight {
    public static void main(String[] args) {
        String str="StudyTonight";
        System.out.println(str.indexOf('u'));   //3rd form
        System.out.println(str.indexOf('t', 3));    //2nd form
        String subString="Ton";
        System.out.println(str.indexOf(subString)); //1st form
        System.out.println(str.indexOf(subString,7));   //4th form
    }
}
```

Output:

```
2
11
5
-1
```

NOTE: -1 indicates that the substring/Character is not found in the given String.

4. **length() method**

String length() function returns the number of characters in a String.

```
public class Demo {
   public static void main(String[] args) {
      String str = "Count me";
      System.out.println(str.length());
   }
}
```

Output:

```
8
```

5. **replace() method**

String replace() method replaces occurances of character with a specified new character.

```
public class Demo {
   public static void main(String[] args) {
      String str = "Change me";
      System.out.println(str.replace('m','M'));
   }
}
;
```

Output:

Change Me

6. substring() method

String substring() method returns a part of the string. substring() method has two override methods.

1. public String substring(int begin);

2. public String substring(int begin, int end);

The first argument represents the starting point of the subtring. If the substring() method is called with only one argument, the subtring returns characters from specified starting point to the end of original string.

If method is called with two arguments, the second argument specify the end point of substring.

```
public class Demo {
    public static void main(String[] args) {
        String str = "0123456789";
        System.out.println(str.substring(4));
        System.out.println(str.substring(4,7));
    }
}
```

Output:

```
456789

456
```

7. toLowerCase() method

String toLowerCase() method returns string with all uppercase characters converted to lowercase.

```
public class Demo {
    public static void main(String[] args) {
        String str = "ABCDEF";
        System.out.println(str.toLowerCase());
    }
}
```

Output:

```
abcdef
```

8. toUpperCase() method

This method returns string with all lowercase character changed to uppercase.

```
public class Demo {
    public static void main(String[] args) {
        String str = "abcdef";
        System.out.println(str.toUpperCase());
    }
}
```

Output:

```
ABCDEF
```

9. valueOf() method

String class uses overloaded version of valueOf() method for all primitive data types and for type Object.

NOTE: valueOf() function is used to convert **primitive data types** into Strings.

```
public class Demo {
    public static void main(String[] args) {
         int num = 35;
         String s1 = String.valueOf(num);    //converting int to String
         System.out.println(s1);
         System.out.println("type of num is: "+s1.getClass().getName());
    }
}
```

Output:

```
35

type of num is: java.lang.String
```

10. toString() method

String toString() method returns the string representation of an object. It is declared in the Object class, hence can be overridden by any java class. (Object class is super class of all java classes).

```
public class Car {
    public static void main(String args[])
    {
        Car c = new Car();
        System.out.println(c);
    }
    public String toString()
    {
        return "This is my car object";
    }
}
```

Output:

```
This is my car object
```

Whenever we will try to print any object of class Car, its toString() function will be called.

NOTE: If we don't override the toString() method and directly print the object, then it would print the object id that contains some hashcode.

String Methods List

Method	Description

char charAt(int index)	It returns char value for the particular index
int length()	It returns string length
static String format(String format, Object... args)	It returns a formatted string.
static String format(Locale l, String format, Object... args)	It returns formatted string with given locale.
String substring(int beginIndex)	It returns substring for given begin index.
String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
boolean equals(Object another)	It checks the equality of string with the given object.
boolean isEmpty()	It checks if string is empty.
String concat(String str)	It concatenates the specified string.
String replace(char old, char new)	It replaces all occurrences of the specified char value.
String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
String[] split(String regex)	It returns a split string matching regex.
String[] split(String regex, int limit)	It returns a split string matching regex and limit.
String intern()	It returns an interned string.
int indexOf(int ch)	It returns the specified char value index.
int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
int indexOf(String substring)	It returns the specified substring index.
int indexOf(String substring, int	It returns the specified substring index starting with

fromIndex)	given index.
String toLowerCase()	It returns a string in lowercase.
String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
String toUpperCase()	It returns a string in uppercase.
String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
String trim()	It removes beginning and ending spaces of this string.
static String valueOf(int value)	It converts given type into string. It is an overloaded method.