**Java String**

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

For example:

char[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=new String(ch);

is same as:

String s="javatpoint";

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

**What is String 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.

**How to create a string object?**

There are two ways to create String object:

**By string literal**

**By new keyword**

**1) String Literal**

Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the "**string constant pool**" first.

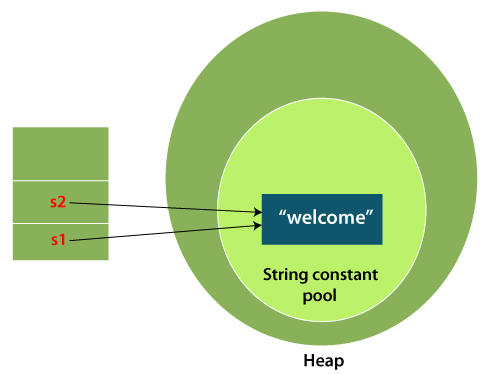
If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

For example:

String s1="Welcome";

String s2="Welcome";

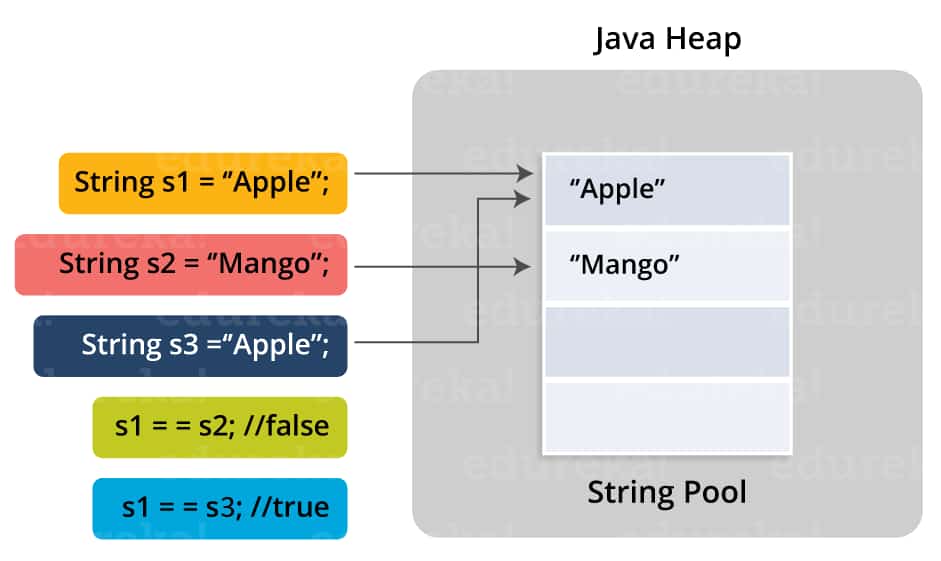
//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

**Note: String objects are stored in a special memory area known as the "string constant pool".**

**Java String Pool**: Java String (constant) 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.



**Why Java uses the concept of String literal?**

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

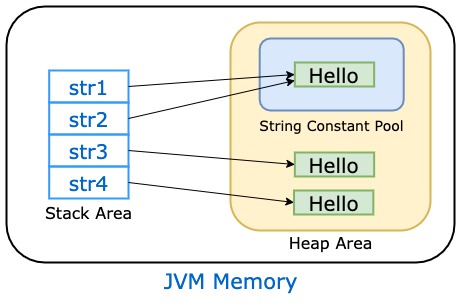
**2) By new keyword**

In this method, we use the "new" keyword to create an instance of String object. We will be following the syntax mentioned below to create a unique String object

String str3 = new String("Hello");

String str4 = new String("Hello");

In this approach, the string objects are created in the heap memory space instead of the string constant pool. **Both str3 and str4 String instances will point to their unique String object in memory.**

****

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

**Java Strings are Immutable**

**(IMP)**

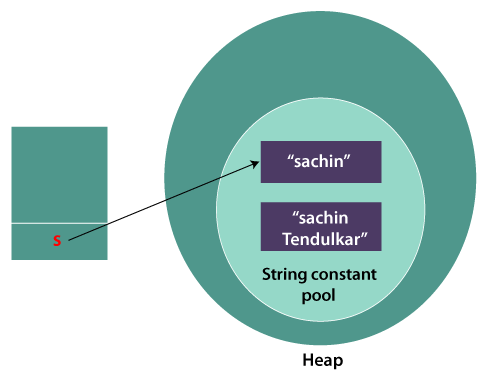
Strings in Java are immutable. This means, it cannot be changed once it is created. Let's understand this with an example:

// Creation of a String

String test = "Sachin";

Here, we have created a string variable named test. The variable holds the string "Apple". Now suppose we want to append a string " Pie" at the end of string variable test.

test.concat(" Tendulkar");



Here, we are using the concat() method to concatenate " Tendulkar" string with initial string "Sachin" to get final string as "Sachin Tendulkar". Here, it looks like we are able to change the content of the original string variable "test" but this is incorrect. Instead, JVM created a new string by combining two strings.

Here is how string concatenation happened here,

JVM first created a new string "Sachin" and assigned it to variable test.

JVM then created a new string by concatenating "Sachin" and "Tendulkar". Remember, the content of original string test is unchanged.

JVM then assiged the new string to the test variable if we write

test=test.concat("Tendulkar");

**Why Java Strings are Immutable**

As Java uses the concept of String literal. Suppose there are three String reference variables, all refer to same String object "Orange". If one reference variable changes the value of the object, it will be update the String value for all three reference variables.

**Benefits of Immutable Strings in Java**

String Pool is possible because String is immutable. Hence saving memory and fast performance.

Class loading is more secure since the string value passed as an argument to load a class can’t be changed.

Thread safety while working with strings in a multi-threaded environment.

More secure since we can’t change the value of string object.

**Overview of Java String Class**

Java String Class is present in java.util package.

A string is a sequence of characters. But, it’s not a primitive data type but an object.

String object is immutable in Java. So once created, we can’t change it.

String has a shortcut way to create its objects using double quotes. These string objects are also called string literals.

String is the only java class that supports operator overloading. We can use the ‘+’ operator to concatenate two strings.

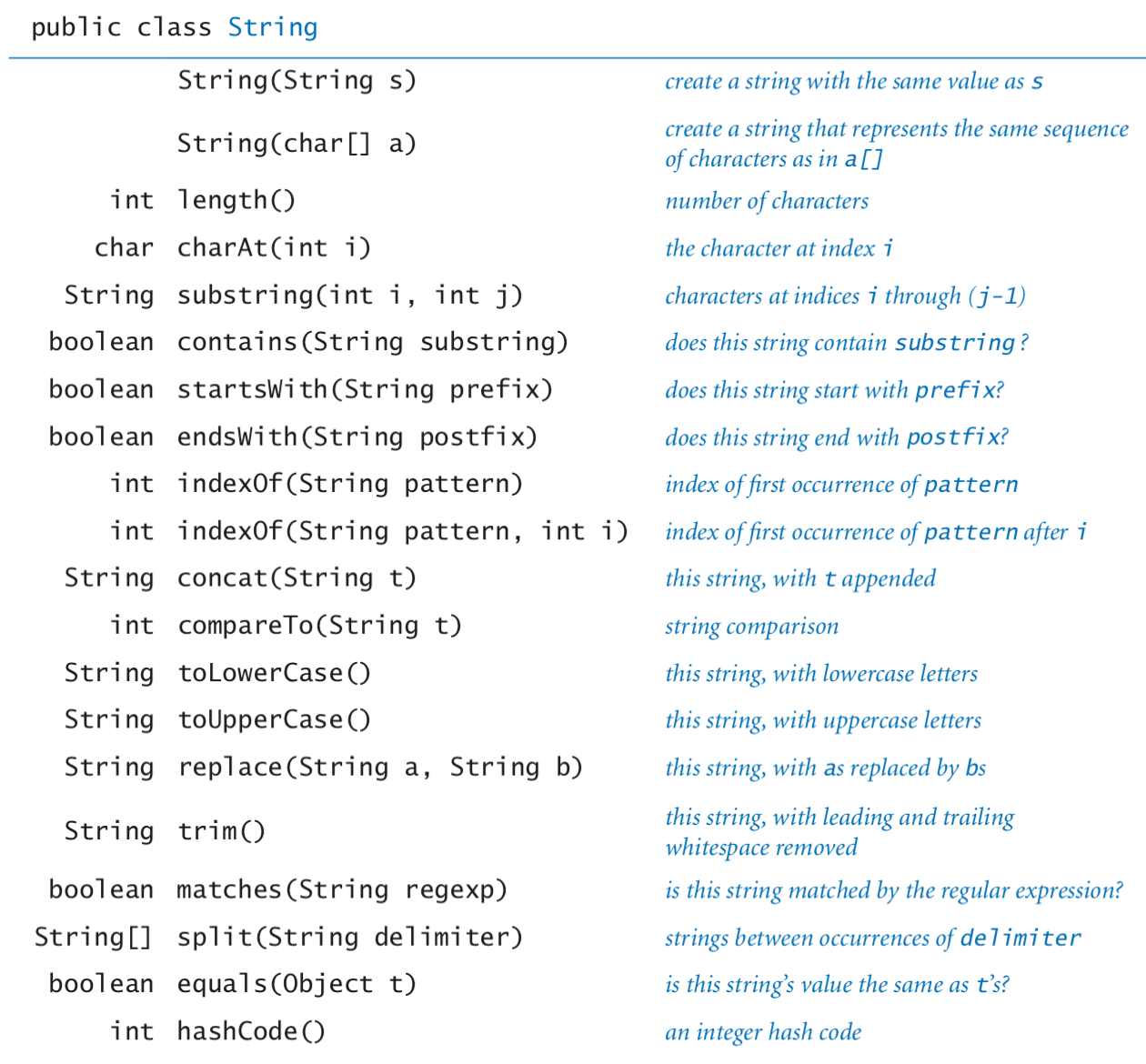
Java store string objects in a specific pre-defined area. String pool is the part of java heap space to store string literals

**Why String class is Final in Java ?**

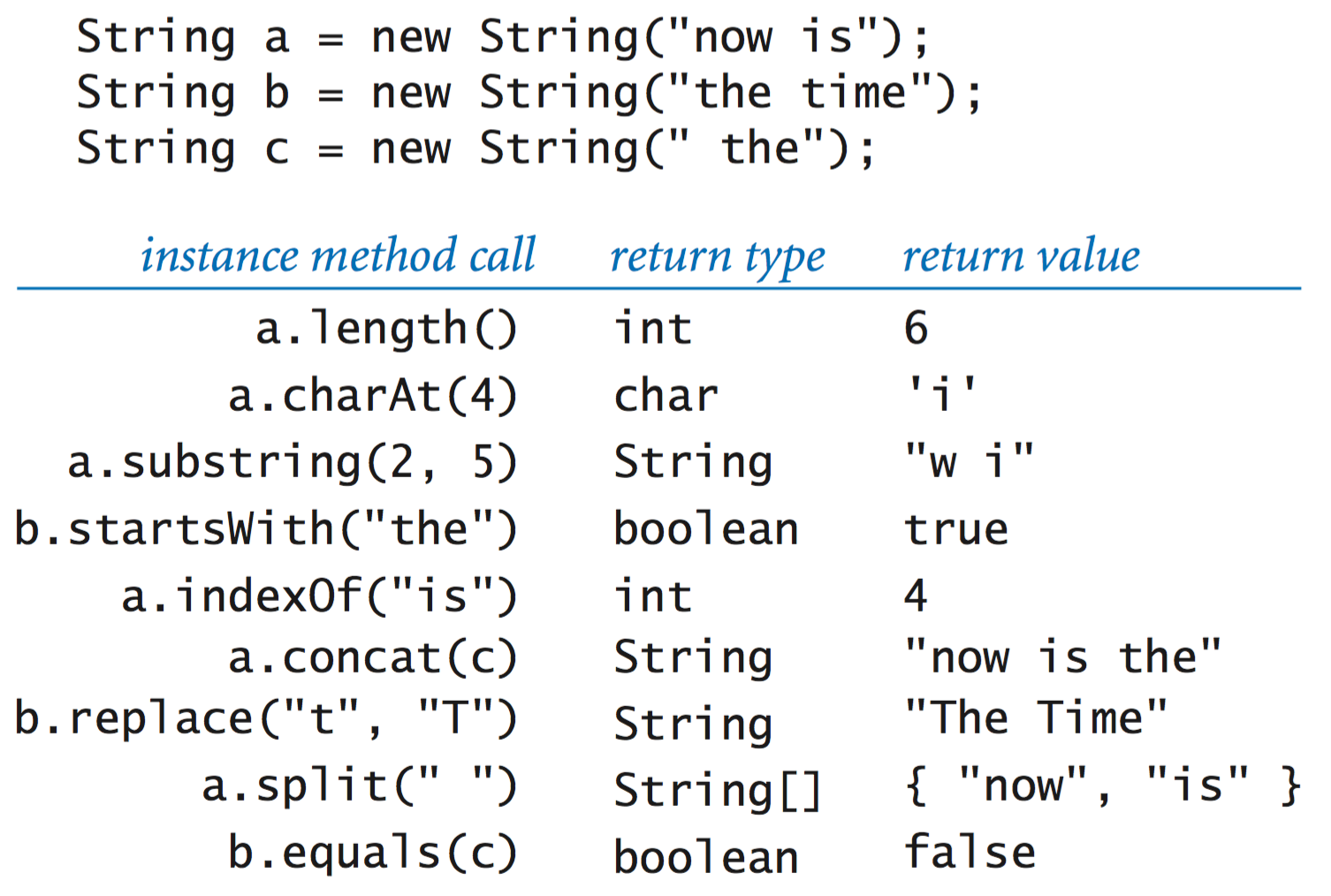
The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

**Java String class methods**

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

****

**Some example**

****