

STRINGS

Unlike other programming languages, Strings in java are different. Well how you wonder? Let us first understand what is a string.

C → *is this a single character or multiple character?*

If you can recollect, it is a single character. Now let me ask you the same question by considering one more example as shown below:

CODE → *is this a single character or multiple character?*

Now you will answer, **it is a sequence of characters or collection of character or series of characters.**


The different words you have used to answer the second example is only referred to as **STRINGS** in java.

You are able to differentiate between a **character** and a **string**. But computer can't tell the difference between character and string.

To resolve this **problem**, java came up with a **solution**.

Well what you under?



The solution is: 

Put the characters in single quotes → **'C'**

Put the string in double quotes → **"CODE"**

When the computer encounters single quotes, it treats it as character. Similarly, when the computer encounters double quotes, it treats it as strings.

What is a Character?

A Character is a value enclosed within single quotes.

What is a String?

A String is a series of characters enclosed within double quotes.

Strings are treated as **objects** in java.

In java, Strings are further divided into two types:

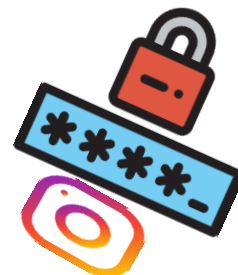
- 1) **Mutable Strings**
- 2) **Immutable Strings.**

As the name itself indicates, **mutable strings** are such strings which are **changeable** and **immutable strings** are such strings which are **non-changeable**.

Let us look at few real time examples to understand this:



Months in a year



Instagram Password

MUTABLE STRINGS



Gmail id



Passport Details.

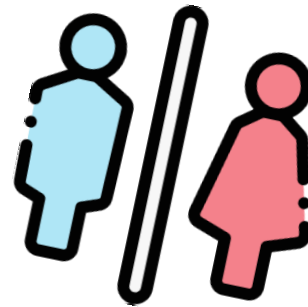
If you look at the examples given above, you may notice they are all changeable and hence they fall under the category of Mutable Strings.

Let us now have a look at non-changeable strings:

*Would
you
like to
guess?*



Earths colour



Gender

IMMUTABLE STRINGS



Date-of-birth



Flags colour Combination

Similarly, if you look at the examples given above, you may notice they are all non-changeable and hence they fall under the category of Immutable Strings.

Let us now learn **Immutable strings** in detail.



Definition: *Immutable strings are such strings that are unmodifiable or unchangeable. They are Created using **String Class**.*

Different ways of creating Immutable strings:

- 1) `char name[] = { 'j' , 'a' , 'v' , 'a' }`
`String s = new String(name);`
- 2) `String s = "java";`
- 3) `String s = new String ("java");`

After getting to know different ways of creating strings, one must now learn different ways of comparing strings.

Different ways of comparing strings:

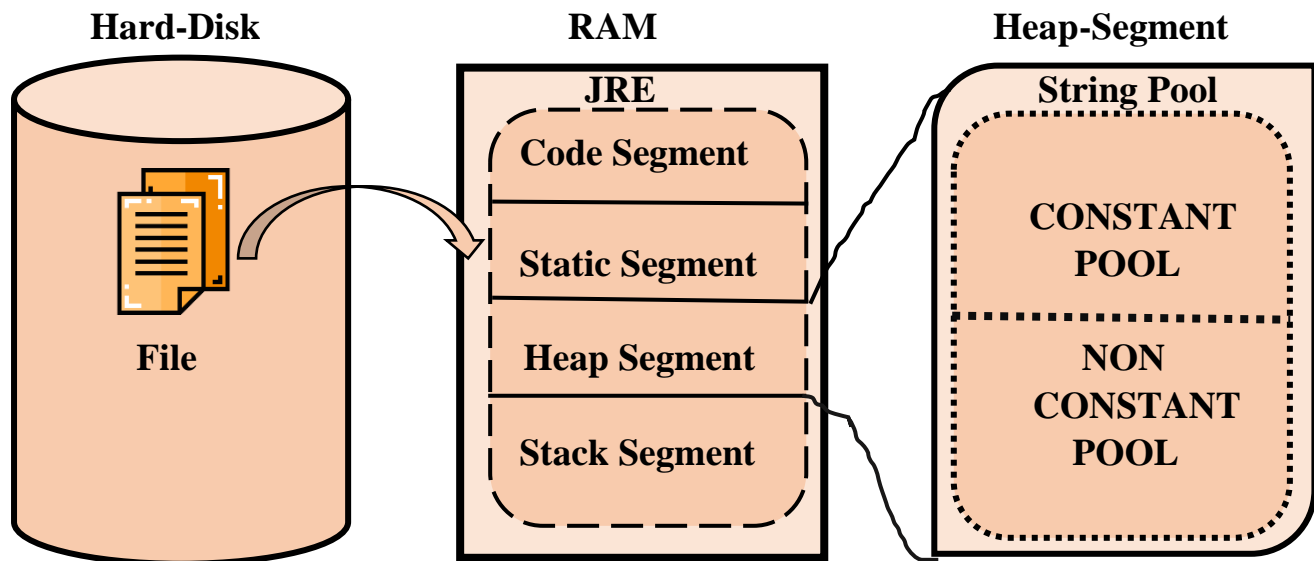
- 1) `==` → String references are compared.
- 2) `equals()` → String values/literals are compared.
- 3) `compareTo()` → Strings are compared character by character.
- 4) `equalsIgnoreCase()` → Strings are compared by ignoring the case.

Before we start writing our first code on strings, let us understand few other concepts which one should be aware of while coding strings.

We have already discussed hard-disk, ram, and microprocessor. Now you may wonder why am i introducing them here and how is this related to strings.



Consider the diagram shown below:



We know microprocessor is the most important device on our computer and there are two other devices which play a crucial role when it comes to storing the data which are hard-disk and ram. The file is stored on hard-disk but if it has to execute it should be placed on ram. Also, we have learned, when the file is placed on ram, the entire region is not allocated for its execution instead, one region of ram is allocated for this program file to execute which is only referred to as **JRE**.

JRE is further divided into four segments:

- 1) Code Segment
- 2) Static Segment
- 3) Heap Segment
- 4) Stack Segment.

Out of all those segments, **Strings** are allocated memory on **Heap Segment**.

Heap Segment further consists of two pools:

- 1)Constant pool
- 2)Non-Constant Pool.

To understand strings from memory perspective, one must know the features of these pools which is shown below:

CONSTANT POOL

Strings that are created **without using new** keyword are allocated memory on this pool.

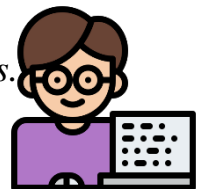
Duplicates are not allowed.

Non-CONSTANT POOL

Strings that are created **using new** keyword are allocated memory on this pool.

Duplicates are allowed.

The wait is over, you now have sufficient knowledge to start coding strings.



Example 1)

```
class Demo
{
    public static void main(String[] args)
    {
        String s1 = "JAVA";
        String s2 = "JAVA";
        if(s1 == s2)
        {
            System.out.println("References are equal");
        }
        else
        {
            System.out.println("References are not equal");
        }
    }
}
```

Output: References are equal.

Explanation: In the above code we made use of **“==”** operator to compare two strings. We know **“==”** operator compares two strings based on the references. Both the strings are created without using new keyword and if you recollect, they will now be allocated memory on constant pool where duplicates are not allowed. Hence only one copy of string literal is created and both the references will point to the same string which means both s1 and s2 will have same address as only one string object is created. Thus, the output references are equal.

Example 2)

```
class Demo
{
    public static void main(String[] args)
    {
        String s1 = "JAVA";
        String s2 = "JAVA";
        if(s1.equals(s2) == true)
        {
            System.out.println("String values are equal");
        }
        else
        {
            System.out.println("String values are not equal");
        }
    }
}
```

Output: String values are equal.

Explanation: In the above code we made use of "equals ()" to compare two strings. We know equals() compares two strings based on the values. Both the strings are created without using new keyword and if you recollect, they will now be allocated memory on constant pool where duplicates are not allowed. Hence only one copy of string literal is created. Since both the strings are same, we get the output as String values are equal.

Example 3)

```
class Demo
{
    public static void main(String[] args)
    {
        String s1 = "JAVA";
        String s2 = new String("JAVA");
        if(s1 == s2)
        {
            System.out.println("References are equal");
        }
        else
        {
            System.out.println("References are not equal");
        }
    }
}
```


Output: References are not equal.

Explanation: In the above code we made use of “==” operator to compare two strings. We know “==” operator compares two strings based on the references. String s1 is created without using new keyword so it gets allocated memory on constant pool and String s2 is created using new keyword so it gets allocated memory on non-constant pool. So now, two different string objects are created with two different addresses. S1 and s2 are the references pointing to two string objects hence they will now consist of different addresses and thus we get the output as references are not equal.



HE UNDERSTOOD. DID YOU?