ECAP615

Programming in Java

Harjinder Kaur

Assistant Professor

Learning Outcomes

After this lecture, you will be able to

- learn the basic concept of Strings,
- understand the ways of creating Strings,
- use of different String functions.

Strings

- Basically, string is a sequence of characters but it's not a primitive type.
- When we create a string in java, it actually creates an object of type String
- String is immutable object which means that it cannot be changed once it is created.

Strings

- String is the only class where operator overloading is supported in java.
- We can concat two strings using + operator. For example "a"+"b"="ab".
- Java provides two useful classes for String manipulation – StringBuffer and StringBuilder.

Strings

- Strings in Java are Objects that are backed internally by a char array.
- Since arrays are immutable(cannot grow), Strings are immutable as well.
- Whenever a change to a String is made, an entirely new String is created.
- A String variable contains a collection of characters surrounded by double quotes.

Strings

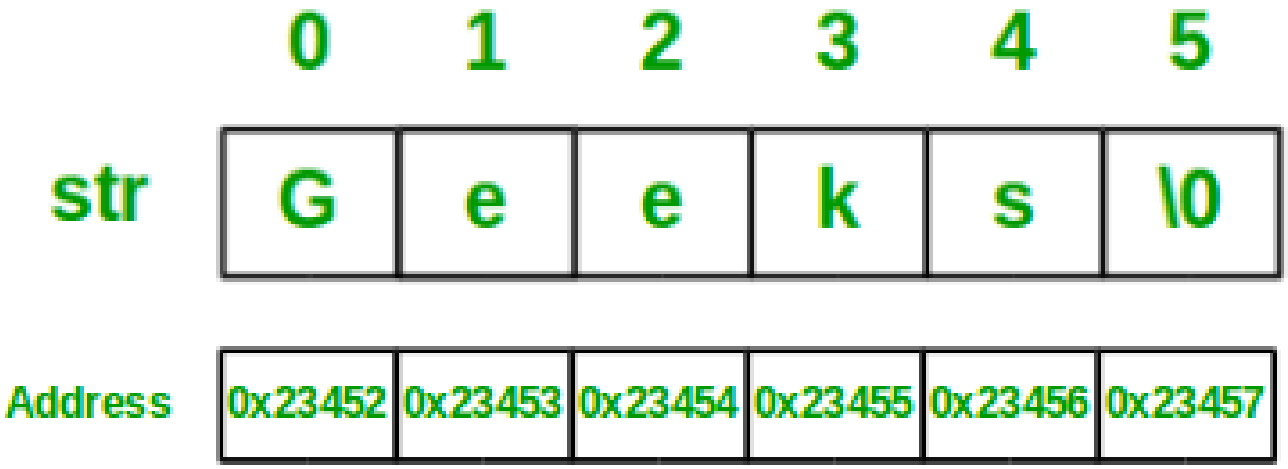
Syntax:

<String_Type> <string_variable> = "<sequence_of_string>";

Example:

String str = "Geeks";

Memory Representation of String



Memory Allocation to Strings

- JVM divides the allocated memory to a Java program into two parts.
  - Stack
  - heap.
- Stack is used for execution purpose and heap is used for storage purpose.
- In that heap memory, JVM allocates some memory specially meant for string literals.
- This part of the heap memory is called String Constant Pool.

Memory Allocation to Strings

- Whenever you create a string object using string literal, that object is stored in the string constant pool .
- Whenever you create a string object using new keyword, such object is stored in the heap memory.

Memory Allocation to Strings

Example

- String s1 = "abc";
- String s2 = "xyz";
- String s3 = "123";
- String s4 = "A";

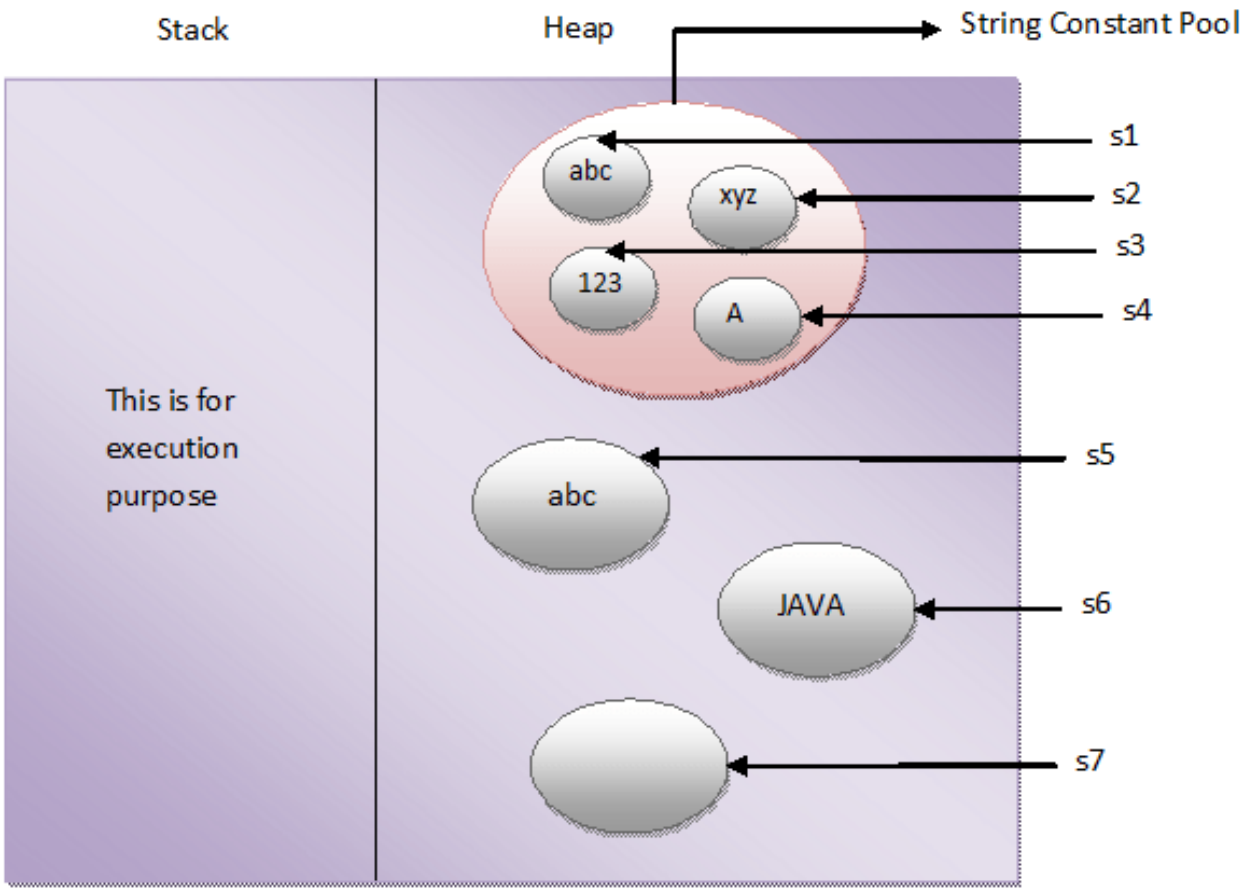
will be stored in the String Constant Pool.

Memory Allocation to Strings

When you create string objects using new keyword like below, they will be stored in the heap memory.

- String s5 = new String("abc");
- char[] c = {'J', 'A', 'V', 'A'};
- String s6 = new String(c);
- String s7 = new String(new StringBuffer());

String data in String Constant Pool



Different Ways of Creating Strings

The following are the ways to create a string object:

1. Using string literal.
2. Using new keyword.

Using string literal

- This is the most common way of creating string.
- In this case a string literal is enclosed with double quotes.

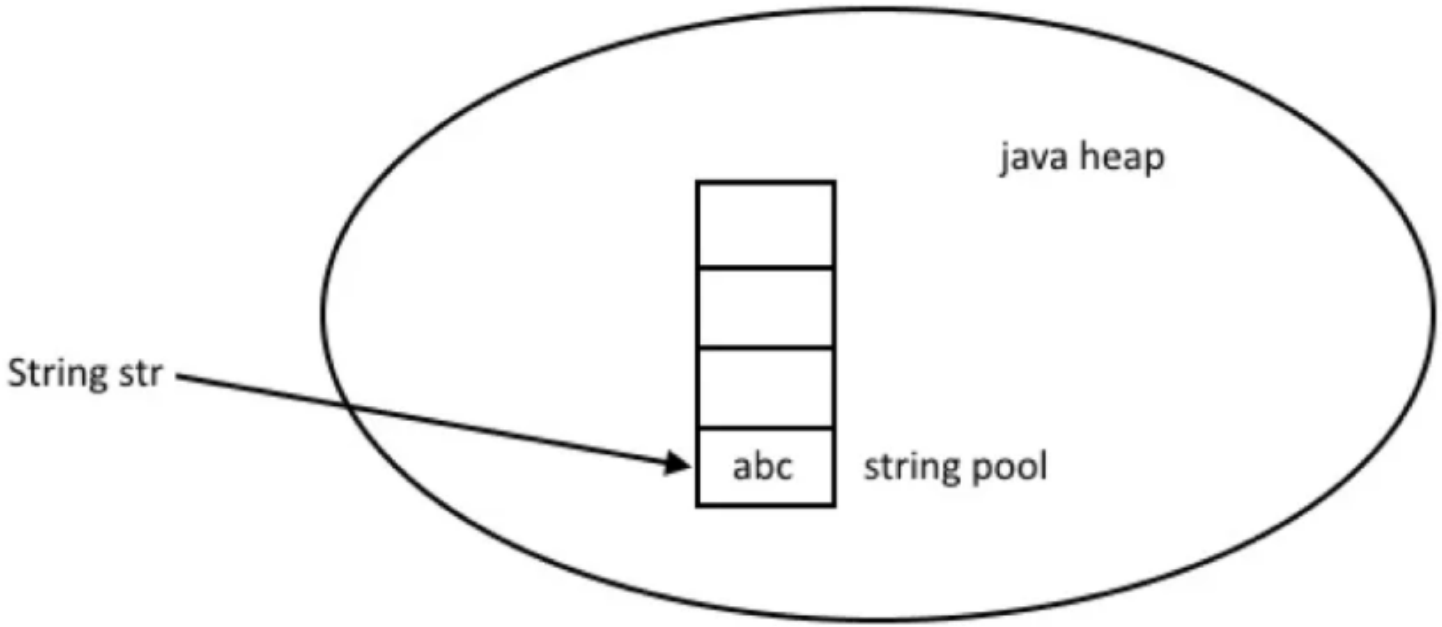
Syntax:

String var_name=“value”

Example:

String str = "abc";

Internal Working



Using new keyword

- We can create String object using new operator, just like any normal java class.
- There are several constructors available in String class to get String from char array, byte array, StringBuffer and StringBuilder.

Using new keyword

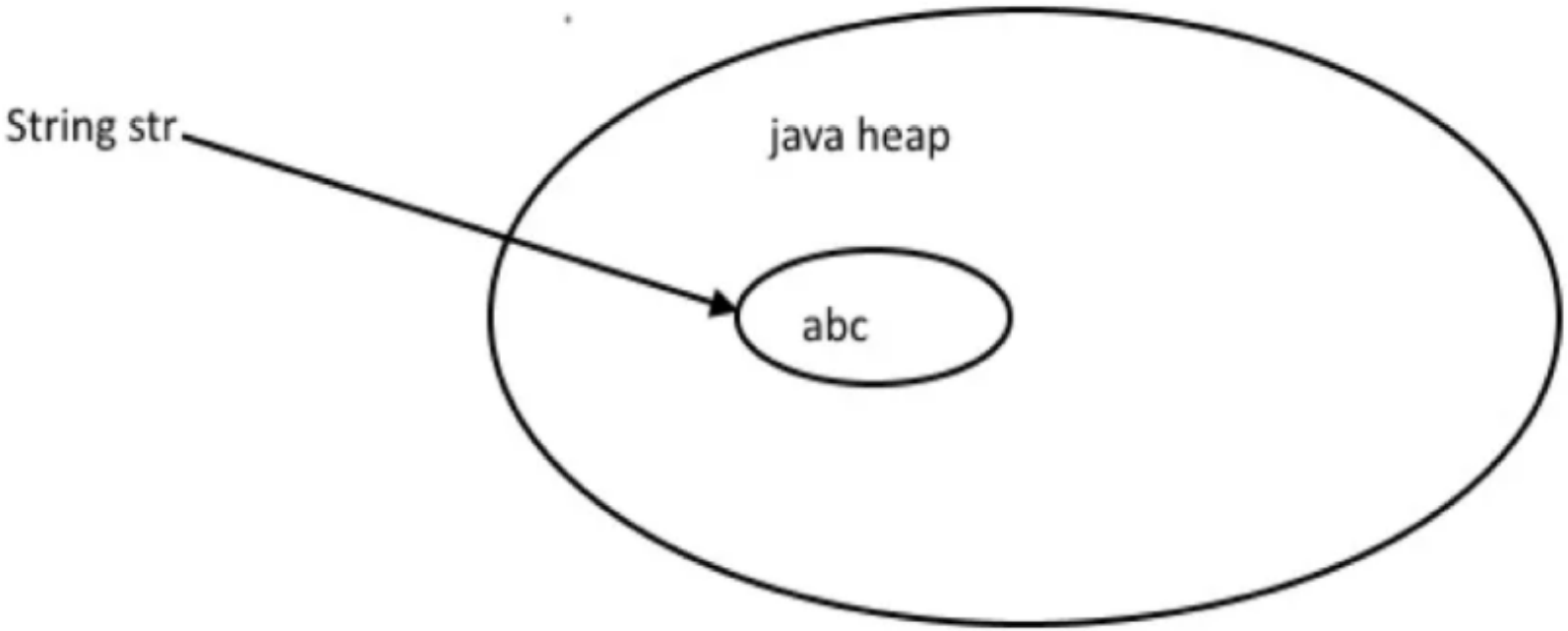
Example:

```
String str  =  new String("abc");

char[ ] a = {'a', 'b', 'c'};

String str2  =  new String(a);
```

Internal Working Using new



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 );

   }

}
```

String Methods

A String in Java is actually an object, which contain methods that can perform certain operations on strings which are as follows:

- String Length.
- String Concatenation.
- Finding a Character in a String.
- To convert the case of a String.
- Comparison of Strings.
- contains

String Length

The accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object.

String Length

Example:

```
public class Main {
  public static void main(String[] args) {
    String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    System.out.println("The length of the txt string is: " +
txt.length());
  }
}
```

Output: 26

String Concatenation

- Strings are more commonly concatenated with the + operator
- The + operator can be used between strings to combine them.

Example:

"Hello," + " world" + "!“

results in –

"Hello, world!"

Example

```
public class StringDemo {

   public static void main(String args[]) {

      String string1 = "saw I was ";

      System.out.println("Dot " + string1 + "Tod");

   }

}
```

Output:

```
Dot saw I was Tod
```

String Concatenation

You can also use the concat() method with string literals for concatenation.

Syntax:

string1.concat(string2);

This returns a new string that is string1 with string2 added to it at the end.

Example:

"My name is ".concat("Zara");

Finding a Character in a String

The indexOf() method returns the index of the first occurrence of a specified text in a string

Example:

```
public class Main {
  public static void main(String[] args) {
    String txt = "Please locate where 'locate' occurs!";
    System.out.println(txt.indexOf("locate"));
  }
}
```

Output: 7

To convert the case of a String

- toUpperCase() and toLowerCase() are used to convert the case of a string.

Example:

```
String txt = "Hello World";

System.out.println(txt.toUpperCase());    // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase());    // Outputs "hello world"
```

Comparison of Strings

- String class provides equals() and equalsIgnoreCase() methods to compare two strings.
- These methods compare the value of string to check if two strings are equal or not.
- It returns true if two strings are equal and false if not.

Example

```
public class StringEqualExample

{

 public static void main(String[] args)

{ //creating two string object

 String s1 = "abc";

String s2 = "abc";

String s3 = "def";

String s4 = "ABC";
```

Example

```
System.out.println(s1.equals(s2));//true

System.out.println(s2.equals(s3));//false

System.out.println(s1.equals(s4));//false;

System.out.println(s1.equalsIgnoreCase(s4));//true
} }
```

Comparison of Strings

- String class implements Comparable interface, which provides compareTo() and compareToIgnoreCase() methods and it compares two strings lexicographically.
- Both strings are converted into Unicode value for comparison and return an integer value which can be greater than, less than or equal to zero.
- If strings are equal then it returns zero or else it returns either greater or less than zero.

Example

```
public class StringCompareToExample {

public static void main(String[] args)

{

String a1 = "abc";

String a2 = "abc";

String a3 = "def";

String a4 = "ABC";
```

Example

```
System.out.println(a1.compareTo(a2));//0

System.out.println(a2.compareTo(a3));//less than 0

System.out.println(a1.compareTo(a4));//greater than 0

System.out.println(a1.compareToIgnoreCase(a4));//0

} }
```

Contains

- Java String contains() methods checks if string contains specified sequence of character or not.
- This method returns true if string contains specified sequence of character, else returns false.

Example

```
public class StringContainsExample

{

 public static void main(String[] args)

 {

String s = "Hello World";

System.out.println(s.contains("W"));//true

System.out.println(s.contains("X"));//false

 } }
```


That’s all for now…