STRINGS

Definition:

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

How to create String object?
There are two ways to create a String object:

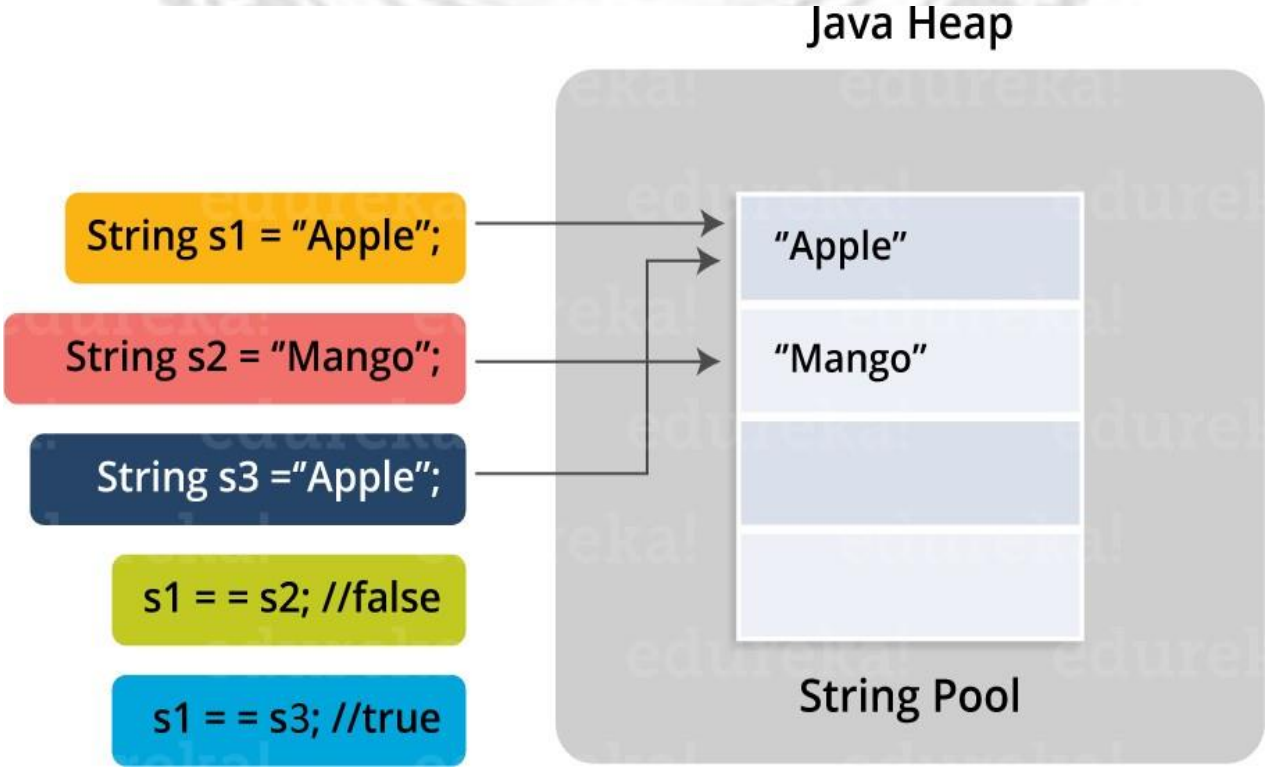
1. **By string literal**: Java String literal is created by using double quotes. For Example: String s="Welcome";

2. **By new keyword**: Java String is created by using a keyword "new".
   For example: String s=new String("Welcome");
   It creates two objects (in String pool and in heap) and one reference variable where the variable 's' will refer to the object in the heap.

Java String Pool:

Java String pool refers to collection of Strings which are stored in heap memory.
In this, whenever a new object is created,

1) String pool first checks whether the object is already present in the pool or not.
2) If it is present, then same reference is returned to the variable
3) else new object will be created in the String pool and the respective reference will be returned.



Refer to the diagrammatic representation for better understanding: In the above image, two Strings are created using literal i.e "Apple" and "Mango". Now, when third String is created with the value "Apple", instead of creating a new object, the already present

object reference is returned.

Example: Creating Strings

```
public class StringExample
{
public static void main(String args[])
{
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}
}
```

Output:
java
strings
example

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means un-modifiable or unchangeable.

- ✓ Once string object is created its data or state can't be changed but a new string object is created.
- ✓ Let's try to understand the immutability concept by the example given below:

```
class Simple{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends the string at the end
   System.out.println(s);//will print Sachin because strings are immutable objec
ts
 }
}
```

Output: Sachin

4.10: METHODS

Methods of String class in Java

java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting strings etc.

Important methods of String class.

S.No.	Method	Description
1)	public boolean equals(Object anObject)	Compares this string to the specified object.
2)	public boolean equalsIgnoreCase(String another)	Compares this String to another String, ignoring case.
3)	public String concat(String str)	Concatenates the specified string to the end of this string.
4)	public int compareTo(String str)	Compares two strings and returns int
5)	public int compareToIgnoreCase(String str)	Compares two strings, ignoring case differences.
6)	public String substring(int beginIndex)	Returns a new string that is a substring of this string.
7)	public String substring(int beginIndex,int endIndex)	Returns a new string that is a substring of this string.
8)	public String toUpperCase()	Converts all of the characters in this String to upper case
9)	public String toLowerCase()	Converts all of the characters in this String to lower case.
10)	public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
11)	public boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
12)	public boolean endsWith(String suffix)	Tests if this string ends with the specified suffix.
13)	public char charAt(int index)	Returns the char value at the specified index.
14)	public int length()	Returns the length of this string.
15)	public String intern()	Returns a canonical representation for the string object.
16)	public byte[] getBytes()	Converts string into byte array.
17)	public char[] toCharArray()	Converts string into char array.
18)	public static String valueOf(int i)	converts the int into String.
19)	public static String valueOf(long i)	converts the long into String.
20)	public static String valueOf(float i)	converts the float into String.

21)	public static String valueOf(double i)	converts the double into String.
22)	public static String valueOf(boolean i)	converts the boolean into String.
23)	public static String valueOf(char i)	converts the char into String.
24)	public static String valueOf(char[] i)	converts the char array into String.
25)	public static String valueOf(Object obj)	converts the Object into String.
26)	public void replaceAll(String firstString,String secondString)	Changes the firstString with secondString.

- **String comparison in Java**

  - ✓ We can compare two given strings on the basis of content and reference.
  - ✓ It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.
  - ✓ There are three ways to compare String objects:

1. By equals() method
2. By = = operator
3. By compareTo() method

1) By equals() method

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 another){}** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another){}** compares this String to another String, ignoring case.

Example: equals() method

```
class Simple{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   String s4="Saurav";
   System.out.println(s1.equals(s2));//true
   System.out.println(s1.equals(s3));//true
   System.out.println(s1.equals(s4));//false
 }
}
```

Output:

```
true
true
false
```

Example: equalsIgnoreCase(String) method

```
class Simple{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";
   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s3));//true
 }
}
```

Output:

```
false
 true
```

2) By == operator

The = = operator compares references not values.

Example: == operator:

```
class Simple{
 public static void main(String args[])
{
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   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 nonpoo
l)
 }

}
```

Output:

```
true
false
```

3) By compareTo() method:

compareTo() method compares values and returns an int which tells if the values compare less than, equal, or greater than.

Suppose s1 and s2 are two string variables.If:

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

Example: compareTo() method:

```
class Simple{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";
```

```
  System.out.println(s1.compareTo(s2));//0
  System.out.println(s1.compareTo(s3));//1(because s1>s3)
  System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
 }
}
```

Output:

```
0
1
-1
```

- **String Concatenation in Java**

Concating strings form a new string i.e. the combination of multiple strings.

There are two ways to concat string objects:

1. By + (string concatenation) operator
2. By concat() method

1) By + (string concatenation) operator

String concatenation operator is used to add strings.For Example:

```
//Example of string concatenation operator

class Simple{
 public static void main(String args[]){

  String s="Sachin"+" Tendulkar";
  System.out.println(s);//Sachin Tendulkar
 }
}
```

Output: Sachin Tendulkar

The compiler transforms this to:

```
String s=(new StringBuilder()).append("Sachin").append(" Tendulkar).toString();
```

String concatenation is implemented through the StringBuilder(or StringBuffer) class and its append method.

String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also.

Example:

```
class Simple{
 public static void main(String args[]){
  String s=50+30+"Sachin"+40+40;
```

```
  System.out.println(s);//80Sachin4040
 }
}
```

Output: 80Sachin4040

Note: If either operand is a string, the resulting operation will be string concatenation. If both operands are numbers, the operator will perform an addition.

By concat() method

concat() method concatenates the specified string to the end of current string.

Syntax: public String concat(String another){}

Example of concat(String) method

```
class Simple{
public static void main(String args[]){
  String s1="Sachin ";
  String s2="Tendulkar";
  String s3=s1.concat(s2);
  System.out.println(s3);//Sachin Tendulkar
 }
}
```

Output: Sachin Tendulkar

- **Example Program: Using all the methods of String class**

```
class Simple{
public static void main(String args[])
{
String s="Sachin Tendulkar";
System.out.println("Substring 1: "+s.substring(6));
System.out.println("Substring2: "+s.substring(0,6));
System.out.println("Uppercase: "+s.toUpperCase());
System.out.println("Lowercase: "+s.toLowerCase());
System.out.println("Trim: "+s.trim());
System.out.println("Start With: "+s.startsWith("Sa"));
System.out.println("End with: "+s.endsWith("n"));
System.out.println("Char at Position 0: "+s.charAt(0));
System.out.println("Char at Position 3: "+s.charAt(3))
System.out.println("Length: "+s.length());
String s2=s.intern();
System.out.println("Intern: "+s2);
System.out.println("Replace: "+s.replace('a','q'));
System.out.println("Index 1: "+s.indexOf('I'));
System.out.println("Index 2: "+s.indexOf('I',5));
 }
}
```

```
Substring 1: Tendulkar
Substring2: Sachin
Uppercase: SACHIN TENDULKAR
Lowercase: sachin tendulkar
Trim: Sachin Tendulkar
Start  With:  true
End  with:  false
Char at Position 0: S
Char at Position 3: h
Length: 16
Intern: Sachin Tendulkar
Replace: Sqchin Tendulkqr
Index 1: -1
Index 2: -1
```

4.11: STRING BUFFER CLASS

❖ StringBuffer CLASS

The StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class is same as String except it is mutable i.e. it can be changed.
StringBuffer can be changed dynamically. String buffers are preferred when heavy modification of character strings is involved (appending, inserting, deleting, modifying etc).

Difference between String class and StringBuffer class:

S.No	String Class	StingBuffer Class
1	String objects are constants and immutable(cannot change the content)	StringBuffer objects are not constants and mutable(can change the content)
2	String class supports constant Strings	StringBuffer class supports growable and modifiable strings
3	The methods in the String class are not synchronized	The methods of the StringBuffer class can be synchronized

Important Constructors of StringBuffer class

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16. **Example:** StringBuffer s=**new** StringBuffer();
StringBuffer(String str)	creates a string buffer with the specified string. **Example:** StringBuffer s=**new** StringBuffer("Alice in Wonderland");
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length. **Example:** StringBuffer s=**new** StringBuffer(20);

Important methods of StringBuffer class

Modifier and Type	Method	Description

public synchronized StringBuffer	append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	is used to return the character at the specified position.
public int	length()	is used to return the length of the string i.e.total number of characters.
public String	substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Example:

```
public class StringBufferFunctionsDemo {
        public static void main(String[] args) {
//      Examples of Creation of Strings
        StringBuffer strBuf1 = new StringBuffer("Bobby");
        StringBuffer strBuf2 = new StringBuffer(100); //With capacity 100
        StringBuffer strBuf3 = new StringBuffer(); //Default Capacity 16
        System.out.println("strBuf1 : " + strBuf1);
        System.out.println("strBuf1 capacity : " + strBuf1.capacity());
        System.out.println("strBuf2 capacity : " + strBuf2.capacity());
        System.out.println("strBuf3 capacity : " + strBuf3.capacity());
        System.out.println("strBuf1 length : " + strBuf1.length());
        System.out.println("strBuf1 charAt 2 : " + strBuf1.charAt(2));
```

```
        // A StringIndexOutOfBoundsException is thrown if the index is not valid.
        strBuf1.setCharAt(1, 't');



        System.out.println("strBuf1 after setCharAt 1 to t is : "+ strBuf1);
        System.out.println("strBuf1 toString() is : " + strBuf1.toString());
        strBuf3.append("beginner-java-tutorial");
        System.out.println("strBuf3 when appended with a String : "+ strBuf3.toString());
        strBuf3.insert(1, 'c');
        System.out.println("strBuf3 when c is inserted at 1 : "+ strBuf3.toString());
        strBuf3.delete(1, 'c');
        System.out.println("strBuf3 when c is deleted at 1 : "+ strBuf3.toString());
        strBuf3.reverse();
        System.out.println("Reversed strBuf3 : " + strBuf3);
        strBuf2.setLength(5);
        strBuf2.append("jdbc-tutorial");
        System.out.println("strBuf2 : " + strBuf2);
        //    We can clear a StringBuffer using the following line
        strBuf2.setLength(0);
        System.out.println("strBuf2 when cleared using setLength(0): "+ strBuf2);
        }
}
```

Output:

```
strBuf1 : Bobby strBuf1 capacity : 21 strBuf2 capacity : 100strBuf3
capacity : 16 strBuf1 length : 5 strBuf1 charAt 2 : b
strBuf1 after setCharAt 1 to t is : BtbbystrBuf1 toString() is : Btbby
strBuf3 when appended with a String : beginner-java-tutorialstrBuf3
when c is inserted at 1 : bceginner-java-tutorial strBuf3 when c is
deleted at 1 : b
Reversed strBuf3 : b strBuf2 : jdbc-tutorial
strBuf2 when cleared using setLength(0):
```