SmartCliff
CAREER MOBILITY SOLUTIONS

SDE Readiness Training

Empowering Tomorrow's Innovators

Module I

Java Software Development: Effective Problem Solving

SmartCliff
Career Mobility Solutions

Strings in Java

Learning Level: Basic & Easy

DATE : 15.02.2025

Introduction

- String is a **group of Characters.**
- It is an **object** of type String. The **String class** represents **character strings**.
- All string literals in Java programs, such as "abc", are implemented as **instances of this class.**
- Strings are **Immutable** which means a **constant** and cannot be changed once created.

Note: An object pointed to by a string variable can still be changed.

Example:

String s1 = "Rajesh"

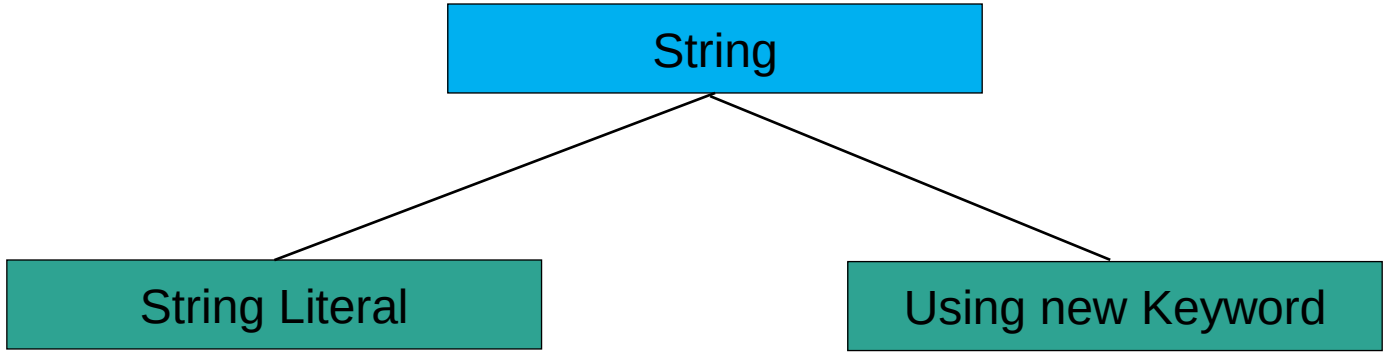
s1 = "Kumar" is still allowed only that s1 is now pointing to another location/object.

Introduction

- To get **changeable strings**, we can use **StringBuffer** and **StringBuilder** classes.
- These classes are available in **java.lang package**.

Creating Strings

- **There are two ways to create a string in Java:**



- **String literal :**

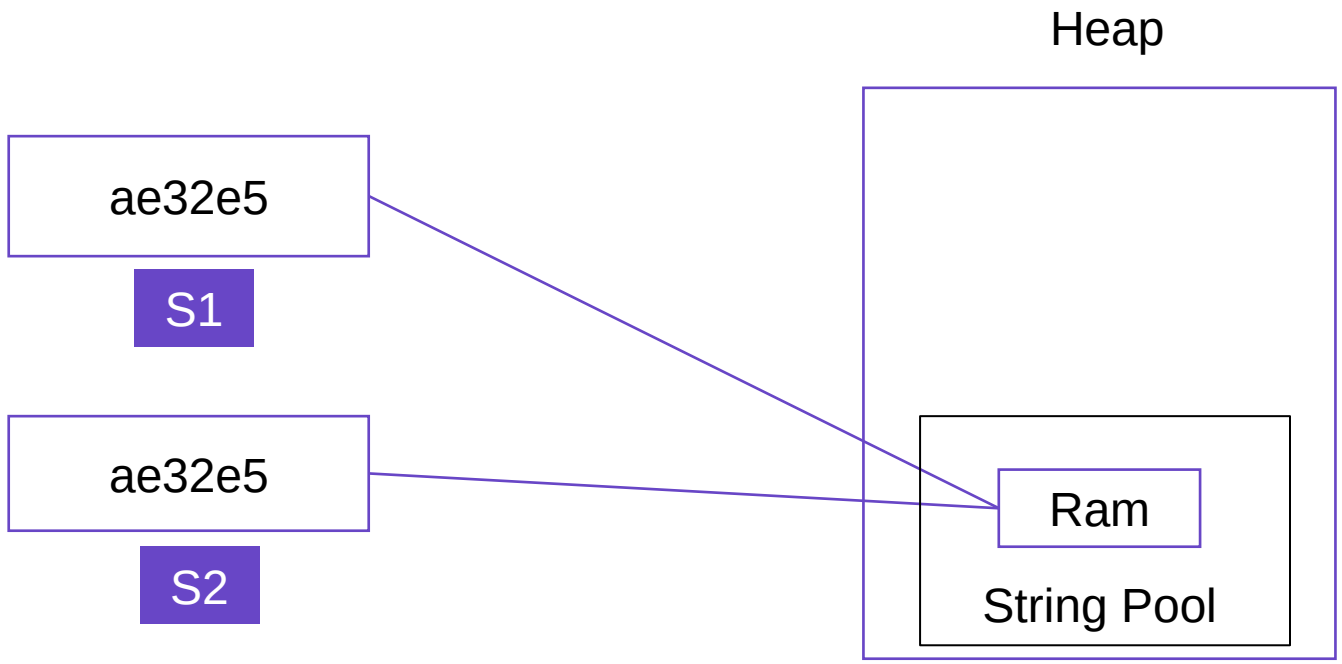
  String str="Ram"; **//char str[]={'R', 'a', 'm'};**

- **Using new Keyword:**

  String str=new String("Ram");

String Literals

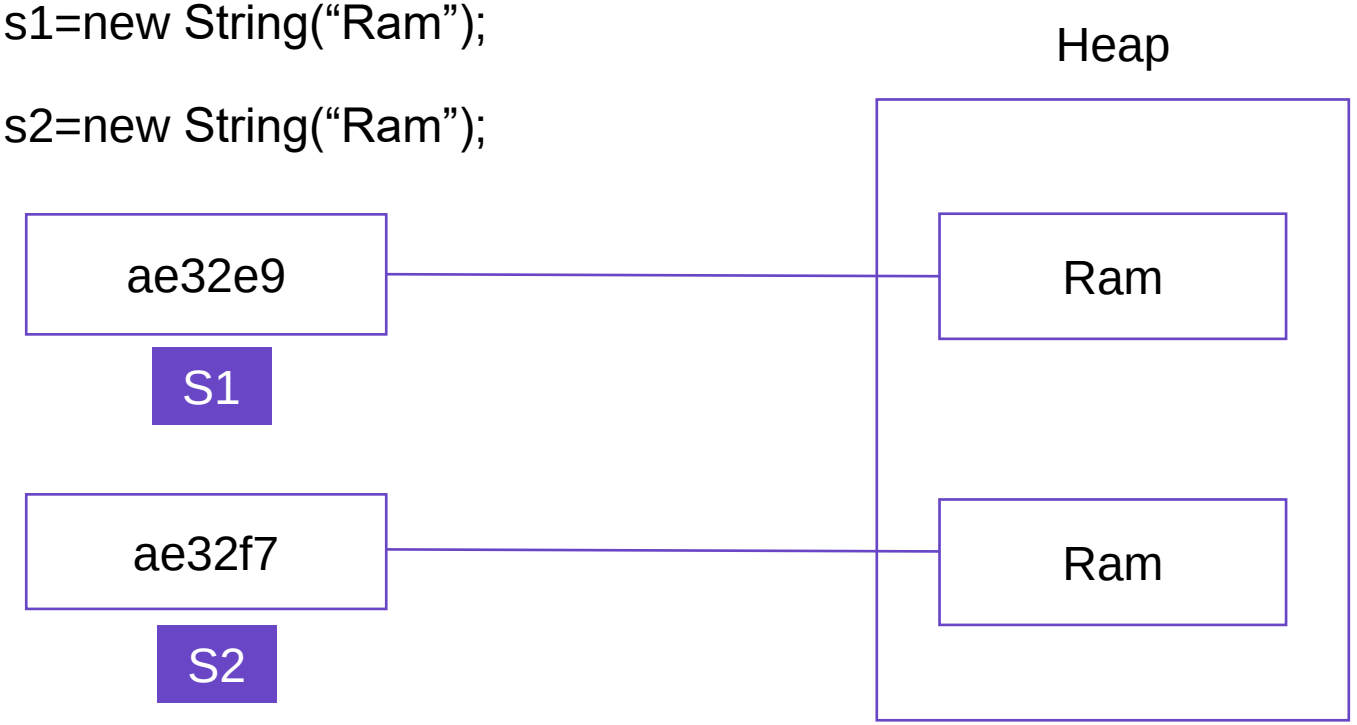
- String s1=“Ram”;
- String s2=“Ram”;



- In the above example, When s1 is interpreted, JVM checks the String Pool for the String “Ram”. Since, it is **not present, a new** object is created.
- In the case of s2 creation, JVM **does not create a new** String “Ram”, instead a **reference** to “Ram” is made to s2.

Creating Strings using new keyword

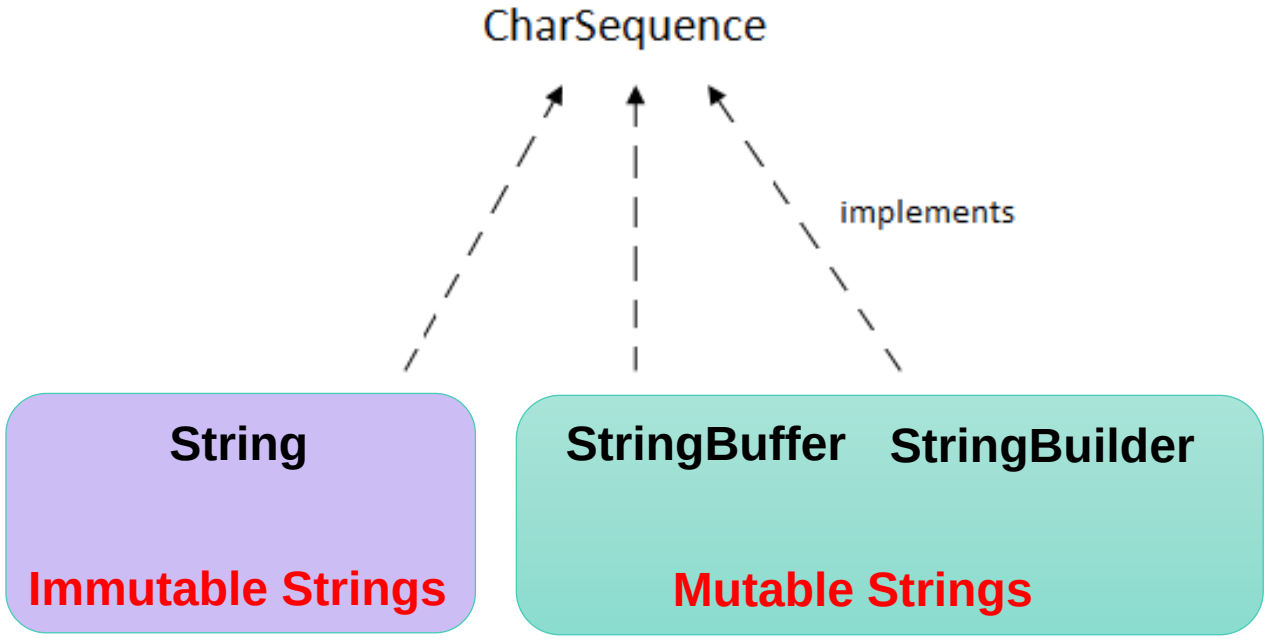
- String s1=new String(“Ram”);
- String s2=new String(“Ram”);



- In the above example, We used a **new keyword** for creating strings each time, so a new String object will be created for each call.

Java String Class Hierarchy

- The **CharSequence** interface is used to represent the sequence of characters.
- **String, StringBuffer, and StringBuilder** classes implement it. It means we can create strings in Java by using these three classes



String Methods

- **Commonly used String methods:**

Method	Description
length()	Returns the length of a specified string
concat()	Appends a string to the end of another string
charAt()	Returns the character value at a specified index within a string
indexOf()	Find the index of the first occurrence of a specified character or substring within a string
substring()	Returns a new string which is the substring of a specified string
toLowerCase()	Converts a string to lower case letters
toUpperCase()	Converts a string to upper case letters
trim()	Removes whitespace from both ends of a string
split()	Split a string into an array of substrings based on a specified delimiter or a pattern

String Methods

- **Commonly used String methods:**

Method	Description
startsWith()	Checks whether a string starts with specified characters
endsWith()	Checks whether a string ends with the specified character(s)
compareTo()	Compare two strings lexicographically. Lexicographical comparison means comparing strings based on the Unicode values of individual characters
contains()	Returns true if and only if the string contains the specified sequence of char values.
repalce()	Replace all occurrences of a specified character or substring within a string with another character or substring
replaceAll()	Replace all substrings that match a specified pattern with another string
toCharArray()	Converts this string to a new character array

String Methods

- **Commonly used String methods:**

Method	Description
equals()	Compares two strings. Returns true if the strings are equal, and false if not
equalsIgnoreCase()	Compares two strings, ignoring case considerations
matches()	Finds whether a string matches a specified pattern
valueOf()	Convert different types of data into their string representation
toString()	Returns a string representation of the object
For More Methods: ***https://docs.oracle.com/javase/8/docs/api/java/lang/String.html***	

String Methods: length

- **length()** function is used to find the **number of character** values (including space, ‘\n’) in the string.

```
String s1="\nHai all";
System.out.println("S1 Length is “+s1.length());
String s2="\nHai Ravi";
System.out.println("S2 Length is "+s2.length());
```

Output:
S1 Length is 8
S2 Length is 9

String Methods: Concat

- **Two strings** can be joined together using **concat operation**.
- Methods like **concat()** and **+ Operator** is used to combine 2 strings.

```
//Using + Operator

 String str="Always"+" Smile";
   System.out.println(str);
```

Output:
Always Smile

- **Note:** After a string literal, **all the +** will be treated as **string concatenation operator.**

```
class StringConcatenation{
   public static void main(String args[]){
      String str=20+30+"Sachin"+20+30;
      System.out.println(str); }
   }
}
```

Output:
50Sachin2030

String Methods: concat()

```
//Using concat() method

String s1="I";
String s2 = " Will";
String s3 = s1.concat(s2);
System.out.println("\ns1: "+s1+"\ns2: "+s2+"\ns3: "+s3);
```

Output:
s1: I
s2: Will
s3: I Will

String Methods: charAt()

- Retrieve the character at a **specific index** within a given string.
- **public char charAt(int index) - index:** Position of the character to be retrieved. **Indexing** starts from **0**, so the first character in the string is at index 0, the second character at index 1, and so on.
- Method throws **StringIndexOutOfBoundsException** if the index provided is **negative or greater than or equal to the length** of the string

```
public class CharAtDemo {
        public static void main(String[] args) {
            String str = "Discoverable";
            System. out.println(“Character at index 5: “ + str.charAt(5)); // Retrieves the character 'v' at index 5
            System. out.println("Character at index 11: " + str.charAt(11)); // Retrieves the character 'e' at index 11
        }
}
```

Output:

Character at index 5: v

Character at index 11: e

String Methods: indexOf()

- Find the index of the **first occurrence of a specified character or substring** within a string
- **public int indexOf(int ch):** Returns the **index** of the **first occurrence** of the specified character within the given string or **-1** if the character does not occur.
- **public int indexOf(String str):** Returns the **index** of the **first occurrence** of the specified substring within the given string or **-1** if the substring does not occur

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

        String str = "Hello, World!";
         System.out.println("Index of 'o': " + str.indexOf('o')); // Returns the index 7
        System.out.println("Index of 'lo, W': " + str.indexOf("lo, W")); // Returns the index 3
        System.out.println("Index of 'Java': " + str.indexOf("Java")); // Returns -1
    }
}
```

Output:

```
Index of 'o': 4
Index of 'lo, W': 3
Index of 'Java': -1
```

String Methods: substring()

- A **part of string** is called substring.
- In case of substring **startIndex** is **inclusive** and **endIndex** is **exclusive.**
- **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
- **public String substring(int startIndex, int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex & just before the endIndex

```
class StringSubstring{
     public static void main(String []args){
       String s = "Education";
       System.out.println(s.substring(6));  //Substring with start index alone
       System.out.println(s.substring(3,6));  //Substring with start index and end index
   }
}
```

Output:

ion

cat

String Methods: toUpperCase() and toLowerCase()

- **toUpperCase()** method converts all the letters in the String into uppercase letter
- **toLowerCase()** method converts all the letters in the String into lowercase letter.

```
class StringSubstring{
    public static void main(String []args){
        String s = "DisCoVEr";
        System.out.println(s.toUpperCase());
        System.out.println(s.toLowerCase());
   }
}
```

Output:
DISCOVER
discover

String Methods: trim()

- It **eliminates white spaces** before and after the String.

```
class StringTrim{
    public static void main(String []args){
        String s = "   Concentrate   ";
       System.out.println(s.trim());Output:
   }
}
```

Output:
Concentrate //printed without space (before & after)

String Methods: split()

- Split a string into an **array of substrings** based on a given **Regular Expression.**
- **public String[] split(String regex) – regex:** The delimiter that specifies where to split the string. It can be a regular expression or a delimiter.

```
public class SplitExample {
    public static void main(String[] args) {
        String str = "apple,banana,orange";
        String[] fruits = str.split(",");       // Splitting the string using comma as the delimiter
        for (String fruit: fruits) {             // Displaying the substrings
            System.out.println(fruit);
        }

        String sentence = "Brown fox jumps";
        String[] words = sentence.split("\\s+"); // Splitting by whitespaces (regular expression pattern)
            for (String word: words) {              // Displaying the substrings
            System.out.println(word);
        }
    }
}
```

Output:

```
apple
banana
orange
brown
fox
jumps
```

String Methods: startsWith() and endsWith()

- **startsWith()** – Tests if this string starts with the specified prefix.
- **endsWith()** - Tests if this string ends with the specified suffix.

```
class SubString{
    public static void main(String []args){
        String s = "DisCoVEr";
       System.out.println(s.startsWith("Dis"));
       System.out.println(s.endsWith("er"));
  }
}
```

Output:

true

false

String Methods: toCharArray()

toCharArray(): Converts the string into character array .

```
class StringMethods{
    public static void main(String []args){
        String s="Welcome";
        System.out.print("The string is "+s+"\n");
        char[] ch=s.toCharArray();
        for(int i=0;i<ch.length;i++)
            System.out.println(ch[i]);
   }
}
```

Output:

```
The string is Welcome
W
e
l
c
o
m
e
```

String Methods: equals()

- **equals() Method :** It is used to compare the invoking String to the object specified.
- It will return **true**, if the argument is not null and it is String object which contains the **same sequence** of characters as the invoking String.

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

Output:

Strings are equal

String Methods: equalsIgnoreCase()

- **equalsIgnoreCase() Method :** It Compares this String to another String, **ignoring case considerations**.
- Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the **two strings are equal ignoring case**.

```
public class StringTest{
      public static void main(String[] args){
            String s1="Computer";
            String s2="COMPUTER";
            if(s1.equalsIgnoreCase(s2))
                System.out.println("Strings are equal");
            else
                System.out.println("Strings are not equal");
      }
}
```

Output:

Strings are equal

String Methods: ==

Using == operator: It compares the references and not the contents.

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

Output:

Strings are equal

String Methods: ==

Using == operator: It compares the references and not the contents.

```
public class StringTest{
      public static void main(String[] args){
            String s1=“Computer";
            String s2=“Raj";
            if(s1==s2)
                System.out.println("Strings are equal");
            else
                System.out.println("Strings are not equal");
      }
}
```

Output:

Strings are not equal

String Methods: ==

Using == operator: It compares the references and not the contents.

```
public class StringTest{
      public static void main(String[] args){
            String s1=“Computer";
            String s2=“Raj";
            if(s1==s2)
                System.out.println("Strings are equal");
            else
                System.out.println("Strings are not equal");
      }
}
```

Output:

Strings are not equal

String Methods: compareTo

- Compare **two strings lexicographically.**
- **public int compareTo(String anotherstring) - anotherstring:** The string to be compared with the current string.
- The **compareTo()** method compares two strings lexicographically. It returns an **integer value** that indicates the relationship between the two strings:
    - **Returns 0** if the **two strings are equal.**
    - Returns a **negative integer** if the **current string is lexicographically less than the specified string.**
    - Returns a **positive integer** if the **current string is lexicographically greater than the specified string**

String Methods: compareTo

```
public class CompareToDemo {
    public static void main(String[] args) {
        String str1 = “Java";
        String str2 = “Python";
        String str3 = "Java";

        System.out.println("Comparison of str1 and str2: " + str1.compareTo(str2));
        System.out.println("Comparison of str2 and str1: " + str2.compareTo(str1));
        System.out.println("Comparison of str1 and str3: " + str1.compareTo(str3));
    }
}
```

Output:

Comparison of str1 and str2: -6

Comparison of str2 and str1: 6

Comparison of str1 and str3: 0

String Methods: contains

- Method returns **true if and only if the string contains the specified sequence of char values otherwise false**
- **public boolean contains(CharSequence sequence) - sequence:** The sequence of characters to be searched for in the string.

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

        String str = "Java Programming!";

        System.out.println("Contains 'Java': " + str.contains("Java"));
        System.out.println("Contains 'Programs': " + str.contains("Programs"));
    }
}
```

Output:

Contains 'Hello': true

Contains 'Java': false

String Methods: repalce

- Replace **all occurrences of a specified character or substring within a string with another character or substring.**

- **public String replace(char oldChar, char newChar).**

- **public String replace(CharSequence target, CharSequence replacement).**

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

        String originalString = "Java, World!";

        System.out.println("----Character Replacement---");
        System.out.println("Original string: " + originalString);
        System.out.println("Replaced string: " + originalString.replace('a', 'A'));

         System.out.println("----String Replacement---");
        System.out.println("Original string: " + originalString);
        System.out.println("Replaced string: " + originalString.replace("World", "Program"));
    }
}
```

Output:

```
----Character Replacement---
Original string: Java, World!
Replaced string: JAvA, World!
----String Replacement---
Original string: Java, World!
Replaced string: Java, Program!
```

String Methods: repalceAll

- Replace all substrings that **match a specified pattern with another string.**
- **public String replaceAll(char oldChar, char newChar)**

```
public class ReplaceAllDemo {
    public static void main(String[] args) {
        String originalString = "Quick brown fox";

        System.out.println("Original string: " + originalString);
        System.out.println("Replaced string: " + originalString.replaceAll("\\s+", ":"));

        System.out.println("Original string: " + originalString);
        System.out.println("Replaced string: " + originalString.replaceAll("[aeiouAEIOU]", "-"));
    }
}
```

Output:

```
Original string: Quick brown fox
Replaced string: Quick:brown:fox
Original string: Quick brown fox
Replaced string: Q--ck br-wn f-x
```

String Methods: matches

- Finds whether a **string matches a specified pattern.**
- **public boolean matches(String regex) :** Returns **true** if the entire string matches the given regular expression;

  otherwise, it returns **false.**

```
public class MatchesDemo {
    public static void main(String[] args) {
        String str1 = "Java, World!";
        String str2 = "12345";

        System.out.println("String 1 matches: " + str1.matches("Java"));// Returns false
        System.out.println("String 2 matches: " + str1.matches("Java, [A-Za-z]+!")); // Returns true (matches using regex)
        System.out.println("String 3 matches: " + str2.matches("\\d+")); // Returns true (matches digits)
    }
}
```

Output:

```
String 1 matches: false
String 2 matches: true
String 3 matches: true
```

String Methods: valueOf

- Convert **different types of data into their string representation**
- **static String valueOf(boolean b) -** Returns the string representation of the **boolean** argument.
- **static String valueOf(char c) -** Returns the string representation of the **character** argument.
- **static String valueOf(char[] data) -** Returns the string representation of the **character array** argument.
- **static String valueOf(double d) -** Returns the string representation of the **double** argument.
- **static String valueOf(float f) -** Returns the string representation of the **float** argument.
- **static String valueOf(int i) -** Returns the string representation of the **int** argument.
- **static String valueOf(long l) -** Returns the string representation of the **long** argument.
- **static String valueOf(Object obj) -** Returns the string representation of the **object** argument with the helpof the object's **toString** Method.

String Methods: valueOf

```
public class ValueOfDemo {
	public static void main(String[] args) {
		char[] charArray = {'J', 'a', 'v', 'a'};
		Object obj = new Integer(456);

		//Converting primitive values to strings
		System.out.println("Boolean string: " + String.valueOf(true));
		System.out.println("Character string: " + String.valueOf('A'));
		System.out.println("Integer string: " + String.valueOf(123));
		System.out.println("Double string: " + String.valueOf(3.14));

		// Converting character array to a string
		System.out.println("Character array string: " +String.valueOf(charArray));

		// Converting an object to a string
		System.out.println("Object string: " + String.valueOf(obj).toString());
	}
}
```

Output:

```
Boolean string: true
Character string: A
Integer string: 123
Double string: 3.14
Character array string: Java
Object string: 456
```

String Methods: toString()

- The **toString()** method in Java is a method of the **Object class**, and it is used to **return a string representation** of an object.

- **public String toString() :**

```
public class ToStringExample {
        private String name;
        private int age;
        public ToStringExample(String name, int age) {
            this.name = name;
            this.age = age;
        }
       @Override
        public String toString() {
                return "ToStringExample{" +"name='" + name + '\'' +", age=" + age +'}';
         }
        public static void main(String[] args) {
                    ToStringExample person = new ToStringExample("John", 30);
                    System.out.println(person.toString());
        }
}
```

Output:
ToStringExample{name='John', age=30}

StringBuffer and StringBuilder Class

- To create **mutable** (modifiable) String objects, Java **StringBuffer** and **StringBuilder** Class is used.
- The **StringBuffer** and **StringBuilder** class in Java is the same as String class except it is **mutable** i.e. it can be changed.
- The Java **StringBuilder** class is similar to **StringBuffer** class except that it is **non-synchronized.**
- **Create mutable strings:**
- **StringBuffer str = new StringBuffer(“Ram");**
- **StringBuilder str = new StringBuilder(“Raj");**

Note:

Both have **similar type of methods**

StringBuffer and StringBuilder Methods

Commonly used StringBuffer and StringBuilder methods:

Method	Description
append(String s)	It 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.
insert(int offset, String s)	It 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.
replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
reverse()	is used to reverse the string.

StringBuffer and StringBuilder Methods

Method	Description
capacity()	It is used to return the current capacity.
ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
charAt(int index)	It is used to return the character at the specified position.
length()	It is used to return the length of the string i.e. total number of characters.
substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

StringBuffer and StringBuilder Methods: append()

This method **adds** the given argument with the string **at the end.**

```
StringBuffer s1=new StringBuffer("Hello ");  (OR)

StringBuilder s1=new StringBuilder("Hello ");

s1.append("World");

System.out.println(s1);
```

Output:

Hello World

StringBuffer and StringBuilder Methods: insert()

- This method **inserts** the given string at the **given position.**
- This method has **2** arguments.
- The first one is the **postion** & second is the **string** to be inserted.

```
StringBuffer s1=new StringBuffer("I want a job");  (OR)

StringBuilder s1=new StringBuilder("I want a job");

s1.insert(2,"really ");

System.out.println(s1);
```

Output:

I really want a job

StringBuffer and StringBuilder Methods: replace()

- This method replaces the given String **from** the specified beginIndex and **before** the endIndex.
- This method has **3 arguments.**
- The first one is the begining index & second is the end index and third is the string to be replaced.
- Exam;

```
StringBuffer s1=new StringBuffer("Success");  (OR)
StringBuffer s1=new StringBuffer("Success");
s1.replace(2,4,"XXXX");
System.out.println(s1);
```

Output:

SuXXXXess

StringBuffer and StringBuilder Methods: delete()

- This method deletes the String from the specified beginIndex & before the endIndex.
- This method has 2 arguments.
- The first one is the begining index & second is the end index.

```
StringBuffer s1=new StringBuffer("Helping");  (OR)

StringBuffer s1=new StringBuffer("Helping");

s1.delete(1,4);

System.out.println(s1);
```

Output:

Hing

StringBuffer and StringBuilder Methods: reverse()

This method reverses the string.

```
StringBuffer s1=new StringBuffer("TOP");  (OR)

StringBuilder s1=new StringBuilder("TOP");

s1.reverse();

System.out.println(s1);
```

Output:

POT

StringBuffer and StringBuilder Methods: capacity()

- The capacity() method returns the current capacity of the buffer.
- The default capacity of the buffer is 16.
- If the number of character exceeds its current capacity, then, the capacity is recaluclated as (oldcapacity*2)+2
- For example if your current capacity is 16, it will be recalculated as (16*2)+2=34

```
StringBuffer s1=new StringBuffer();  (OR)
StringBuilder s1=new StringBuilder();
System.out.println("s1: "+s1+"  &capacity: "+s1.capacity());
s1.append("Java");
System.out.println("s1: "+s1+"  & capacity: "+s1.capacity());
s1.append(" is my favourite language");
System.out.println("s1: "+s1+"  capacity: "+s1.capacity());
```

Output:

s1: &capacity: 16
s1: Java & capacity: 16
s1: Java is my favourite language
capacity: 34

StringBuffer and StringBuilder Methods: ensureCapacity()

- The **ensureCapacity()** method has the minmum_capacity_required as its argument.
- The default capacity of the **buffer is 16**.
- ensureCapacity() method, **checks** the current capacity with its argument i.e., the minmum_capacity_required. If it is satisfied, it performs nothing. If min>current, capacity is recalculated as **(oldcapacity*2)+2** .
- **For example** if your current capacity is 16, it will be recalculated as **(16*2)+2=34**

StringBuffer and StringBuilder Methods: ensureCapacity()

```
StringBuffer s1=new StringBuffer();      // Creating a string
System.out.println("Initial capacity: "+s1.capacity());
s1.append("typing 16 letter");           //adding 16 characters
System.out.println("capacity after typing 16 letter");
System.out.println("capacity: "+s1.capacity());
System.out.println("Ensuring capacity to 16");
s1.ensureCapacity(16);
System.out.println("capacity: "+s1.capacity());
System.out.println("Ensuring capacity to 17");
s1.ensureCapacity(17);
System.out.println("capacity: "+s1.capacity());
System.out.println("Ensuring capacity to 60");
s1.ensureCapacity(60);
System.out.println("capacity: "+s1.capacity());
```

Output:

Initial capacity: 16
capacity after typing 16 letter
capacity: 16
Ensuring capacity to 16
capacity: 16
Ensuring capacity to 17
capacity: 34
Ensuring capacity to 60
capacity: 70

String Vs String Buffer Vs StringBuilder Class

Comparison Element	String	StringBuffer	String Builder
Storage	String pool / Heap	Heap	Heap
Modifiable	No (Immutable)	Yes (Mutable)	Yes (Mutable)
Thread safe	Yes	Yes	No
Performance	Slow	Fast	Fast

Note: Thread safe means two threads **can't call** the methods simultaneously

String Vs StringBuffer Vs StringBuilder Class

```
/*** This example demonstrates the difference between String, StringBuffer, and StringBuilder class*/
class StringsExample{
        public static void concat1(String s1) {  // Concatenates to String
                s1 = s1 + "Ram";
        }
        // Concatenates to StringBuilder
        public static void concat2(StringBuilder s2) {
                s2.append("Raj");
        }
        // Concatenates to StringBuffer
        public static void concat3(StringBuffer s3)   {
                s3.append("Ravi");
        }
```

String Vs String Buffer Vs StringBuilder Class

```
    public static void main(String[] args) {
        String s1 = "Hello, ";
        concat1(s1); // s1 is not changed
        System.out.println("String: " + s1);

        StringBuilder s2 = new StringBuilder("Hello, ");
        concat2(s2); // s2 is changed
        System.out.println("StringBuilder: " + s2);

        StringBuffer s3 = new StringBuffer("Hello, ");
        concat3(s3); // s3 is changed
        System.out.println("StringBuffer: " + s3);
    }
}
```

Output:

```
String: Hello,
StringBuilder: Hello, Raj
StringBuffer: Hello, Ravi
```

Quiz

1. Which of these class is used to create Immutable strings?

a) StringBuffer Class

b) String Class

c) StringBuilder Class

d) all the above

b) String Class

Quiz

2. Which of these function is used to join two strings

a) concatenation()

b) concat()

c) connect()

d) all the above

b) concat()

Quiz

4. What will be the output:

```
String s1 = "Cat";

String s2 = "Cat";

String s3 = new String("Cat");

System.out.print(s1 == s2);

System.out.print(s1 == s3);
```

a) truefalse

b)truetrue

c) falsetrue

d) falsefalse

a) truefalse

Quiz

5. What will be the output:

```
String s=10+15+15+10+"Example";

System.out.println(s);
```

a) 251510Example

b) 10151510Example

c) 50Example

d) None

c) 50Example

Quiz

6. What will be the output:

```
String s=20+15+”Example”+15+10;

System.out.println(s);
```

a) 35Example35

b) 1015Example1510

c) 35Example1510

d) 1015Example35

c) 35Example1510

Quiz

7. What will be the output:

```
StringBuffer s=new StringBuffer("Entertainment");

s.replace(2,5,"yyyyyy");

System.out.println(s);
```

a) Enyyyyyyyrtainment

b) Enyyyyyyyertainment

c) Enyyyyyyyinment

d) Enyyyyyyytainment

d)Enyyyyyyytainment

Quiz

8. Which of these classes can be call safely by more than one thread simultaneously?

a) StringBuffer Class

b) String Class

c) StringBuilder Class

d) all the above

a) StringBuffer Class

Quiz

9. What will be the output:

```
String s1 = "abc";
String s2 = "def";
System.out.println(s1.compareTo(s2));
```

a) 0

b) true

c) -3

d) false

c) -3

Quiz

10. What will be the output:

```
public class Main {
    public static void main(String[] args) {
        String s1 = null;
        System.out.println(s1); //line 2
        System.out.println(s1.toString()); //line 3
    }
}
```

a) null null

b) null NullPointerException

c) NullPointerException
NullPointerException

d) None

b) null NullPointerException

A little progress each day adds up to big results.

- Satya Nani


THANK YOU
SmartCliff
Career Mobility Solutions