

# INTRODUCTION

- **Strings**

1. String fundamentals
2. Creating String
  - a. Using string literal.
  - b. Using new keyword.
3. String constructors

## **4.String methods**

- a. Length method
  - b. Searching character in a String : `indexOf()` , `lastIndexOf()`
  - c. Searching substring
  - d. String comparison
  - e. Modifying String
  - f. Extracting characters
5. String Buffer and String Builder

# 1. String fundamentals

- Package: *java.lang.String*
- String is a *final* class
  - means no class can *extend* it.
- String is *immutable object*.
  - means once created and initialized, *cannot be changed*.
  - is a constant

# 1. String fundamentals

- String literal pool
  - Java maintains special memory called: “String literal pool”
  - *is a pool of unique Strings: avoid duplicate*



## 2. Creating String

- String literal
- Using *new* keyword

## 2. Creating String

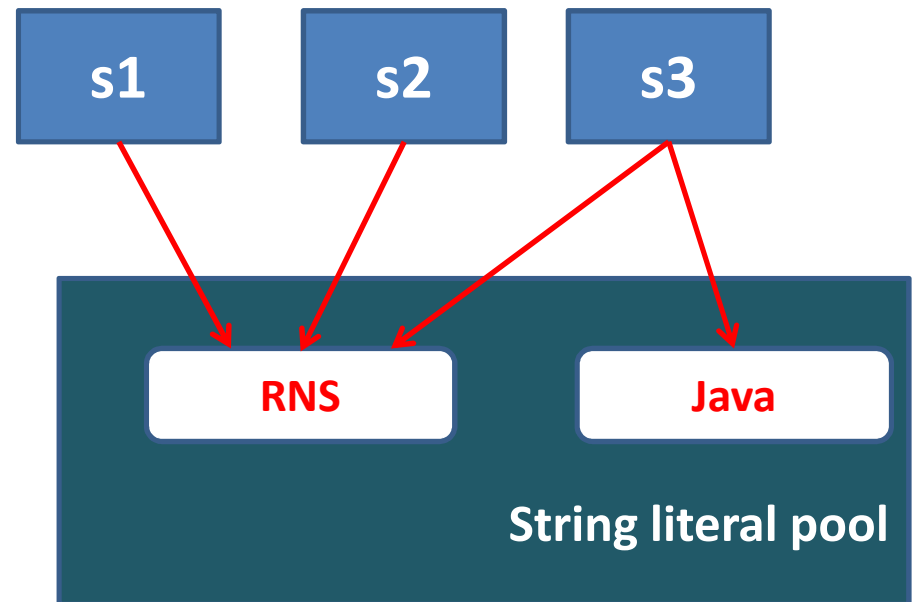
- Using String literal:

String **s1** = “RNS” ;

String **s2** = “RNS” ;

String **s3** = “RNS” ;

**s3** = “Java” ;



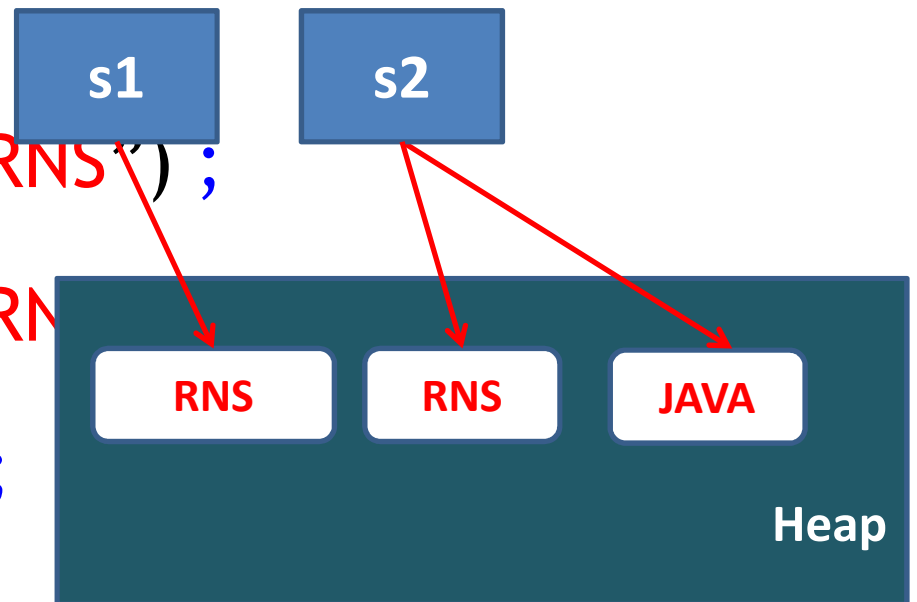
## 2. Creating String

- Using *new* :

```
String s1 = new String("RNS") ;
```

```
String s2 = new String("RNS") ;
```

```
s2 = new String("JAVA") ;
```



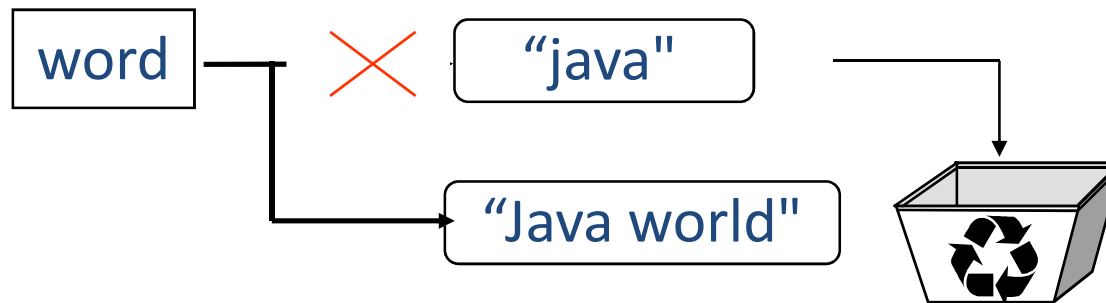
The *new* keyword always creates new string object in Heap.

# Disadvantages of Immutability

Less efficient —

- creates a new string even for small changes.
- and throw away the old one

```
String word = "Java";  
word = word + " world";
```



# INTRODUCTION

- **Strings**

1. String fundamentals
2. Creating String
  - a. Using string literal.
  - b. Using new keyword.
3. String constructors

## **4.String methods**

- a. Length method
  - b. Searching character in a String : `indexOf()` , `lastIndexOf()`
  - c. Searching substring
  - d. String comparison
  - e. Modifying String
  - f. Extracting characters
5. String Buffer and String Builder



# Forms of Constructors

- Using Constructors: *Different forms*
  1. Creating Empty string:
    - *String ( )*
  2. Creating string with other string object
    - *String ( String strObj )*
  3. Creating String from character array
    - *String ( char charArr[] )*
    - *String ( char charArr[], int startIndex, int numChars)*
  4. Creating String from character array
    - *String ( byte ascii[] )*
    - *String ( byte ascii[], int startIndex, int numChars)*

# Forms of Constructors

- Examples:

1. Creating Empty string:

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

2. Creating string with other string object

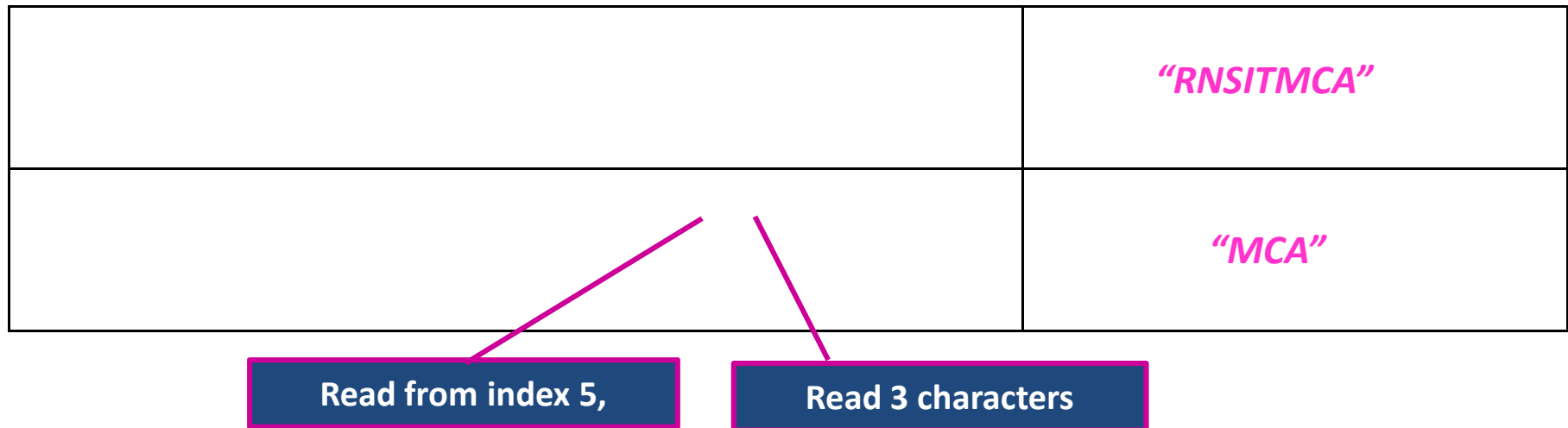
```
String strOb = new String("Apple");
```

```
String newStr = new String(strOb);
```

## Creating String from character array

### 3. Sample program to Create String from character array

```
char [] charArray = {'R', 'N', 'S', 'I', 'T', 'M', 'C', 'A'};
```



# INTRODUCTION

- **Strings**

1. String fundamentals
2. Creating String
  - a. Using string literal.
  - b. Using new keyword.
3. String constructors

## **4.String methods**

- a. Length method
  - b. String searching
  - c. String comparison
  - d. Modifying String
  - e. Extracting characters
5. String Buffer and String Builder

## 4.a) length Methods

- Length method

- returns the number of characters contained in the string object.

- Syntax:

```
int    String.length ()
```

- Example: *find the return value*

```
int len = "Google".length();
```

```
String s1 = "Google";
```

```
len = s1.length();
```

## 4.a) String searching Methods

□ Serching in a string:

<code>int <i>indexOf</i> (   <i>searchKey</i>, [<i>startIndex</i>] )</code>	returns the position of <i>first occurrence</i> of a <i>searchKey</i> in a target string. Returns <b>-1</b> if <i>searchKey</i> not found.
<code>int <i>lastIndexOf</i> (   <i>searchKey</i>, [<i>startIndex</i>] )</code>	returns the position of <i>last occurrence</i> of a <i>searchKey</i> in a target string. Returns <b>-1</b> if <i>searchKey</i> not found.
<code>String <i>substring</i> ( <i>i</i>, <i>k</i> )</code>	returns substring from position <b><i>i</i> to <i>k-1</i></b> , where <b><i>i</i></b> and <b><i>k</i></b> are the start and end Index
<code>boolean <i>contains</i>(String)</code>	method returns true if this string contains s, else false.

# Example..

0	2	5	9	13	19
---	---	---	---	----	----

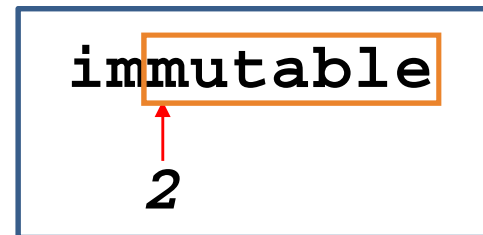
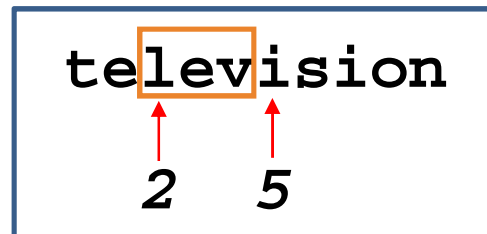
String **name** = "Principal.RNS@gmail.com";

	Returns:
<b>name.indexOf</b> ('i');	2
<b>name.indexOf</b> ('i', 3);	5
<b>name.indexOf</b> ("RNS");	10
<b>name.indexOf</b> ('@');	13
<b>name.indexOf</b> ("Bob");	-1
<b>Name.IndexOf</b> ('.');	9
<b>name.lastIndexOf</b> ('.');	19

## 4.b) Obtaining a string: subString

- Example..

	Returns:
<code>"television".substring (2,5);</code>	<code>"lev"</code>
<code>"immutable".substring (2);</code>	<code>"mutable"</code>
<code>"RNS".substring (9);</code>	<code>"</code> (empty string)





## 4.a) String searching Methods: subString

- Example..

```
String campus = "Student@RNS.Campus.connect";
```

```
String college = "rns";
```

```
if( campus.contains(college))
```

```
    System.out.println ("Student belongs to RNS college");
```

## 4.c) String Comparison -

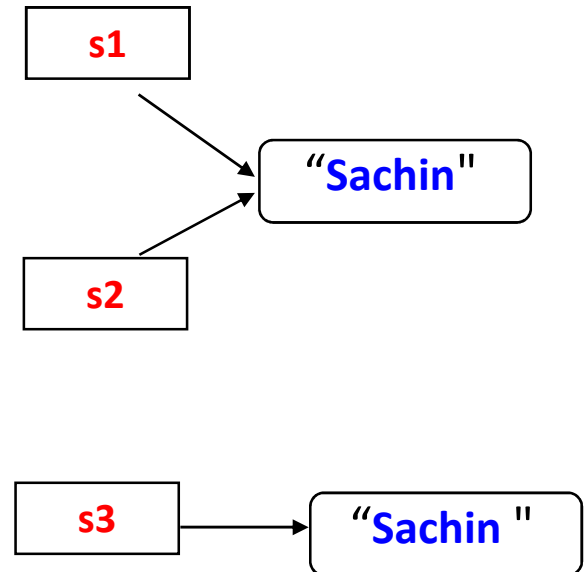
1. *Comparing with “==” operator*
2. *equals() and equalsIgnoreCase()*
3. *compareTo()*
4. *regionMatches()*
5. *startsWith() and endsWith()*

# Comparing with ==

✓ The '==' operator compares references not values.

- Example:

```
class CompareDemo1 {  
    public static void main(String args[]) {  
  
        String s1 = "Sachin";  
        String s2 = "Sachin";  
        String s3 = new String("Sachin");  
  
        if(s1==s2)  
            System.out.println ( "Both points to same object" );  
  
        if(s1==s3)  
            System.out.println ( "Both points to different object" );  
  
    }  
}
```



# *equals()* *and* *equalsIgnoreCase()*

- Syntax:

<i>boolean equals ( Object )</i>	▪ compares two strings for equality
<i>boolean equalsIgnoreCase ( Object )</i>	▪ <i>both</i> returns <i>true</i> if both the strings are same , <i>false</i> otherwise.

# equals() and equalsIgnoreCase()

- Example

returns

---

`"rnsmotors".equals( "rnsmotors" );` → *true*

`"rnsmotors".equals( "RNSmotors" );` → *false*

`"rnsmotors".equalsIgnoreCase( "RNSmotors" );` → *true*

# *Example..*

- Example: write program
  - Read 2 emails from user
  - Display domain of both emails
  - Check if two emails belong to same domain or not
- Say
  - email1 = raj@gmail.com and
  - email2 = usha@rediffmail.com
- Output:
  - Domain of 1st email is : gmail
  - Domain of 2nd email is : rediffmail
  - Result: Domain not matching

---

```
public class DomainDemo {  
    public static void main(String [] a) throws Exception {  
  
        BufferedReader input = new BufferedReader( new InputStreamReader (System.in) );  
        String email_1, email_2, domain1, domain2;  
        int beginIndex, endIndex;  
  
        // a) input the 2 emails  
        System.out.print("Enter first email : ");  
        email_1 = input.readLine();  
  
        System.out.print("Enter Second email : ");  
        email_2 = input.readLine();  
  
        // b) extract the domain  
        beginIndex = email_1.indexOf("@") + 1;  
        endIndex = email_1.lastIndexOf(".");  
  
        domain1 = email_1.substring(beginIndex, endIndex);  
        System.out.println("Domain of first email : " + domain1);  
    }  
}
```

# Example..

```
beginIndex = email_2.indexOf("@") + 1;  
endIndex = email_2.lastIndexOf(".");
```

```
domain2 = email_2.substring(beginIndex, endIndex);  
System.out.println("Domain of second email : " + domain2);
```

```
// c) compare both domain
```

```
if (domain1.equals(domain2))
```

```
    System.out.print("Both are brothers ");
```

```
else
```

```
    System.out.print("Both are Enemies ");
```

```
}
```

```
}
```



# *compareTo()*

- Syntax:

<pre>int word1.compareTo (word2)</pre>	<p>Returns the “<i>difference</i>” [word1 - word2]</p> <p>If the “<i>difference</i>” is</p> <ul style="list-style-type: none"><li>✓ <b>Negative</b> - ( word1 comes before word2 ),</li><li>✓ <b>Zero</b> - ( word1 and word2 are equal )</li><li>✓ <b>Positive</b> - ( word1 comes after word2 ).</li></ul> <p>Often used in conditional statements.</p>
----------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# *compareTo()*

- Example..

```
public class StringDemo {  
    public static void main(String[] args) {  
        String str1 = "RNS", str2 = "PQR";  
        if ( str1.compareTo(str2) == 0 )  
            System.out.println ( "Strings are EQUAL" );  
        if ( str1.compareTo(str2) > 0 )  
            System.out.println ( "str1 is greater than str2" );  
        else  
            System.out.println ( "str1 is less than str2" );  
    }  
}
```

## 4. String Comparison -

```
boolean regionMatches ( int startIndex, String str2, int str2StartIndex, int length)
```

```
boolean regionMatches ( boolean ignoreCase, int startIndex, String str2, int  
str2StartIndex, int length)
```

- ❑ *compares a specific region of one string with another specific region of another string.*
- ❑ *Second form ignore the case.*

# Methods — Changing Case

- *Example:*

*returns*

---

<code>"campus.rns.in".regionMatches(7, "rnsit", 0,3);</code>	<code>// true</code>
<code>"campus.rns.in".regionMatches(7, "RNSIT", 0,3);</code>	<code>// false</code>
<code>"campus.rns.in".regionMatches(true, 7, "RNSIT", 0,3);</code>	<code>// true</code>

## 4. String Comparison -

### ❑ *Example.*

```
String s1 = new String("principal@RNSIT.ac.in");
```

```
String s2 = new String("rnsit");
```

```
String s3 = new String("RNSIT");
```

```
int startIndex = s1.indexOf("@") + 1;
```

```
System.out.print ("Found RNSIT : " );
```

```
System.out.println (s1.regionMatches(startIndex, s2, 0, 3));
```

```
System.out.print ("Found rnsit: " );
```

```
System.out.println (s1.regionMatches(startIndex, s3, 0, 3));
```

```
System.out.print ("Found rnsit: " );
```

```
System.out.println (s1.regionMatches(true, startIndex, s3, 0, 3));
```

Found RNSIT: *true*

Found rnsit: *false*

Found rnsit: *true*

## 4. String Comparison -

- Syntax

<i>boolean</i> <b>startsWith</b> ( <i>String str</i> )	<i>determines whether a given String <b>begins</b> with a specified string.</i>
<i>boolean</i> <b>endsWith</b> ( <i>String str</i> )	<i>determines whether a given String <b>ends</b> with a specified string.</i>

*If string is found returns **true** otherwise **false***

# *compareTo()*

- Example..

```
public class StringDemo {  
    public static void main(String[] args) {  
        String str1 = "Welcome to RNSIT, MCA ";  
        boolean val = str1.startsWith("Welcome") ;  
        System.out.println ( "Does string starts with welcome : " + val );  
        val = str1.endsWith("MCA") ;  
        System.out.println ( "Does string ends with MCA : " + val );  
    }  
}
```

# INTRODUCTION

- **Strings**

1. String fundamentals
2. Creating String
  - a. Using string literal.
  - b. Using new keyword.
3. String constructors

## **4.String methods**

- a. Length method
  - b. String Searching
  - c. String comparison
  - d. Modifying String
  - e. Extracting characters
5. String Buffer and String Builder



# Methods — to modify a string

String <b>replace</b> ( oldCh, newCh )	<i>returns a new string formed from word1 by replacing all occurrences of oldCh with newCh</i>
String <b>trim</b> ();	<i>returns a new string after removing white space at both ends of word1. does not affect whites space in the middle</i>
String <b>toUpperCase</b> (); String <b>toLowerCase</b> ();	<i>returns a new string after converting all characters to upper ( or lower) case</i>
String <b>concat</b> ( String )	<i>appends one String to the end of another.</i>
Concatenating with +	

*All method returns a new string resulting from old String,  
as String can not be modified*

# Methods — replace

- Example:

```
String word1 = "rare";  
String word2 = word1.replace('r', 'd');  
  
// word2 is "dade", but word1 is still "rare"
```

# Methods — trim

Example:

```
String word1 = " Hi John " ;
```

```
String word2 = word1.trim();
```

```
//word2 is "Hi John" - no spaces on either end
```

```
//word1 is still " Hi John " - with spaces
```

# Methods — Changing Case

- *Example:*

	<i>returns</i>
String word1 = "HeLLo";	
String word2 = word1.toUpperCase();	// "HELLO"
String word3 = word1.toLowerCase();	// "hello"
System.out.println ( word1 );	// "HeLLo"

## 3.d) Concatenating using '+' operator

1. *concatenating two strings with '+'*

*Statements*

*returns*

---

```
String s1 = "RNS" + "MCA" ;
```

```
// "RNSMCA"
```

```
System.out.println( "RNS" + "Motors" );
```

```
// prints "RNSMotors"
```

## 3.d) Concatenating using '+' operator

### 2. Concatenating strings with 'other data types'

*Statements*

*returns*

---

```
String s1 = "Bring " + 4 + " Apples" ;    // s1 = "Bring 4 Apples"
```

```
System.out.println("Sum is: " + 2 + 2 );    // prints "Sum is 22"
```

```
System.out.println("Sum is: " + (2 + 2) );    // prints "Sum is 4"
```

# Obtaining character within a String

- `charAt( int index)`
  - extract and return a single character from a String,
- `getBytes()`
  - returns the resultant byte array
- `toCharArray()`
  - returns the resultant character array
- `getChars (int sourceStart, int sourceEnd, char target[ ], int targetStart)`
  - Reads set of characters from source and copies to destination.

## 3.e) Extracting character

- Example:

<i>Statements</i>	<i>returns</i>
<code>char ch = "abzde".charAt(2);</code>	<code>// ch = 'z'</code>
<code>byte [] buf = "Aa".getBytes();</code>	<code>// buf = [ 65 , 97 ]</code>
<code>char [] buf = "RNS".toArray();</code>	<code>// buf = [ 'R' , 'N' , 'S' ]</code>
<code>char[] buf = new char[4];</code>	
<code>"understand".getChars(2, 5, buf, 0);</code>	<code>// buf = [ 'd', 'e', 'r' ]</code>



- String Vs StringBuffer/StringBuilder class

# String Vs StringBuffer/StringBuilder class

Class → Points	String	StringBuffer	StringBuilder
Modifiable	Immutable	Mutable	Mutable
Memory	String Literal pool	Heap	Heap
Thread safe	Yes	Yes	No
Speed	Fast	Slow	Fast
Size	Fixed length	Growable	Growable

## Thread Safe:

does not allow two threads to simultaneously access the same method .

# StringBuffer

- **is mutable** - can change the value of the object .
- The default capacity of the buffer is (16 + initialized string length).
- StringBuffer has the same methods as the StringBuilder
  - *but each method in StringBuffer is synchronized (thread safe)*
  - *Only one thread can use a method at a given time.*

# StringBuffer

- Constructors:

Constructor	Description
<code>StringBuffer()</code>	Reserves room for <b>16 characters</b> default
<code>StringBuffer( int size )</code>	Reserves the size of the object to integer argument passed
<code>StringBuffer( String str )</code>	Sets the initial contents of the object and reserves room for 16 more characters.

# StringBuffer

- Methods to modify buffer

method	Description
length( )	Returns Current length of the string
Capacity( )	Returns total allocated capacity for the object.
ensureCapacity ( int capacity)	If we want to preallocate room for a certain number of characters after StringBuffer has been constructed, we use ensureCapacity().
setLength( int length )	Set the length of the buffer within a object

# StringBuffer

- Other methods

method	Description
Char <b>charAt</b> ( int index)	Returns the character at position specified by 'index' .
void <b>setCharAt</b> ( int index, char ch)	Sets the character at position 'index' with new character 'ch'.
StringBuffer <b>append</b> (String str)	If we want to preallocate room for a certain number of characters after StringBuffer has been constructed, we use <code>ensureCapacity()</code> .

# Array of Strings

- Example:

```
String [] company = { "RNS", "Jindal", "Relience" };
```

```
System.out.println("Elements of array are:");
```

```
for( String name : company )
```

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

# Using String to control switch

- Example:

```
String color = "red";

switch (color) {
case "red":
    System.out.println("Color is Red");
    break;
case "green":
    System.out.println("Color is Green");
    break;
default:
    System.out.println("Color not found");
}
```



# Command line Arguments

```
class CommandLine {  
    public static void main (String [] args)  
  
        System.out.println(" Total Elemnts: " + args.length );  
  
        System.out.println(" Elemnts are: " );  
  
        for( String str : args )  
            System.out.println(str);  
    }  
}
```

### 3.1. Creating String from character array

```
class StringWithCharArray {  
    public static void main (String [] arg) {  
        char [] charArray = { 'H', 'e', 'l', 'l', 'o', 'F', 'u', 'n' };  
  
        System.out.println("Displaying charater Array values:" );  
  
        for ( int i = 0 ; i < charArray.length ; i++)  
            System.out.println(" " + charArray[i] );  
  
        System.out.println("Creating String with char Array:" );  
        String str = new String(charArray);  
  
        System.out.println("Displaying String value:" );  
  
        System.out.println(" String = " + str );  
  
        System.out.println("Another version:" );  
        str = new String(charArray, 6, 3);  
        System.out.println(" String = " + str );  
    }  
}
```