1. **Basics**
2. **String Class.**
3. **String Buffer Class.**
4. **Diff between String class and String Buffer class.**
5. **String Builder.**

**BASICS**

**1. Definition1:**  Sequence of characters placed inside a double quote(“ “) is called as string.

**2. String Handling:-** performing different operations on string data is called **String Handling.** The different operations those we can perform on string data are comparing chars, retrieving chars, changing chars case, replacing chars, etc …

Before performing operation on String, we must store string in java program. We have 4 diferent ways to store string in java program.

2.1. Using char[] object.

2.2. Using String class object.

2.3. Using StringBuffer class object.

2.4. Using StringBuilder class object.

JVM creates String literal in two ways:

1.By assigning string literal directly.

Syntax:

String var-name=”String Literal”;

Example:

String stuName=”sukumar”;

When JVM see opening double quote and closing double quote in program, JVM consider sequence of characters between “ and “ and JVM creates **String object** for storting that string data.

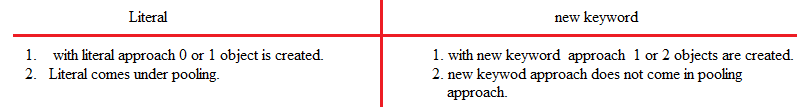
2.By using String,StringBuffer or StringBuilder class constructor.

Example: String s1=new String(“sukumar”);

StringBuffer s1=new StringBuffer(“sukumar”);

String Builder s1= new StringBuilder(“sukumar”);

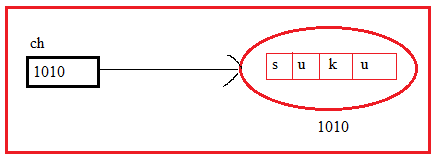
Diff between Literal approach and new with constructor() approach.



**3.char[]:-** Actually, String is stored in char[] object. Inside String,StringBuffer and StringBuilder object also, the string is stored in char[] type variable. The variable name is “Value”.

Example:-

Char [] name={‘s’,’u’,’k’,’u’};



Drawbacks:-

1. Programmer has to type more characters to store a string.It has readability problem.

Ex:- To store “suku”.

Programmer has to type two singlequotes for each character or string.

It is also difficult to read.

1. Size/length problem:- After creating array with string , we can’t append new string to it.
2. In-Built mehod problem:- The array class does not have methods to perform some operations.

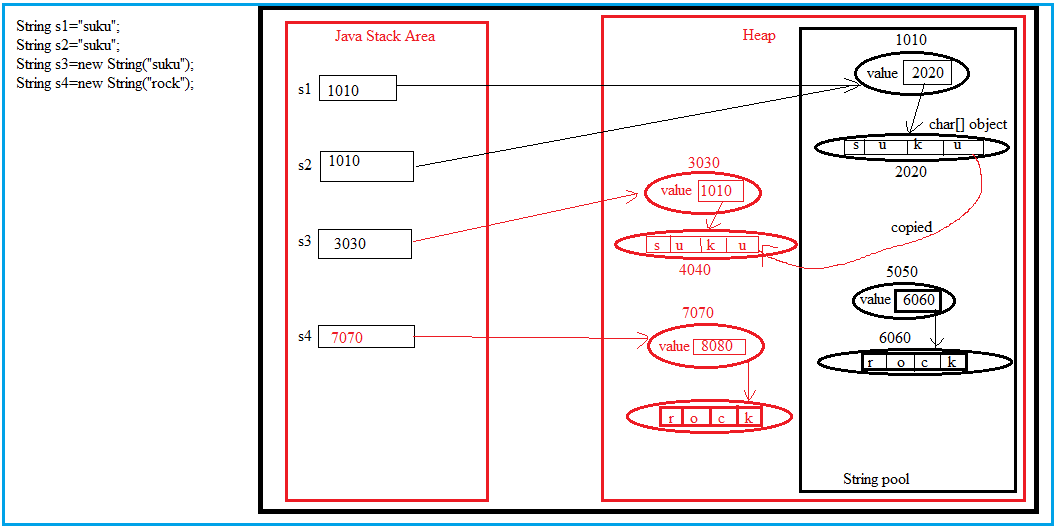
**To overcome these problems, String class has been developed**.

**4.String pool/String Constant pool/String Intern pool**:-

Creating a number of strings may increase the cost and memory too which may reduce the performance also. That’ why JVM introduced String pool concept. The JVM performs some steps during the initialization of string literals that increase the performance and decrease the memor load. To decrease the number of string objects created in the JVM the String class keeps pool of strings.

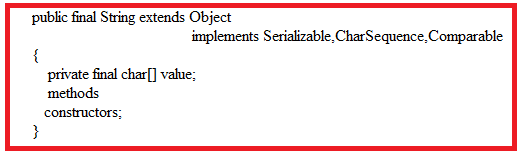
It is Storage area in java heap. It is initially empty. It is privately maintained by “String” class.

When we create a string literal, the JVM first check that literal in the String pool. If the literal is already present in the pool, it returns a refrence to the pooled instance. If the literal is not present I the pool, a new String object takes place in the String pool.



**STRING CLASS**

**1.String:-** The String class definition is in **java.lang** pacakge.

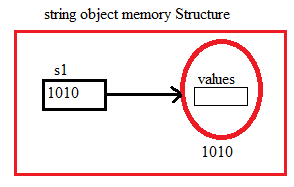
****

**1.1.Constructors**:-

1. String():- It creates empty string object.

Example:

String s1=new String();



1. String(String value)
2. String(StringBuffer sb)//creates new String object with given StringBuffer object content.
3. String(StringBuilder sb1)//creates new String object with given String Builder object content.
4. String(Char [] ch) //creates String object with given char array values.
5. String(char[] ch,int offset, int count) // creates new String object with given count number of characters from the given offset in the char[] object. Here offset is the starting index from which characters must be copied.
6. String(byte[] b) // Creates new String object by copying given byte[] numbers by converting them into their ASCII characters.
7. String(byte[] b, int offset, int count) // Creates new String object with given count number of bytes from given offset in the byte[]. All byte are stored in their ASCII character form.

Note:- we can not pass null as argument directly to constructor,

It leads **to CE:ambiguous error**.

Ex:- String s11=new String(null); // CE: ambiguous error.

Example:-

Class StringDemo{

public static void main(String args[])

{

String s1="sumanvitha";

char[]s2={'s','u','k','u','m','a','r'};

String s3=new String(s1);

String s4=new String(s2);

String s5=new String(s2,0,4);

String s6=s1;

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

System.out.println(s4);

System.out.println(s5);

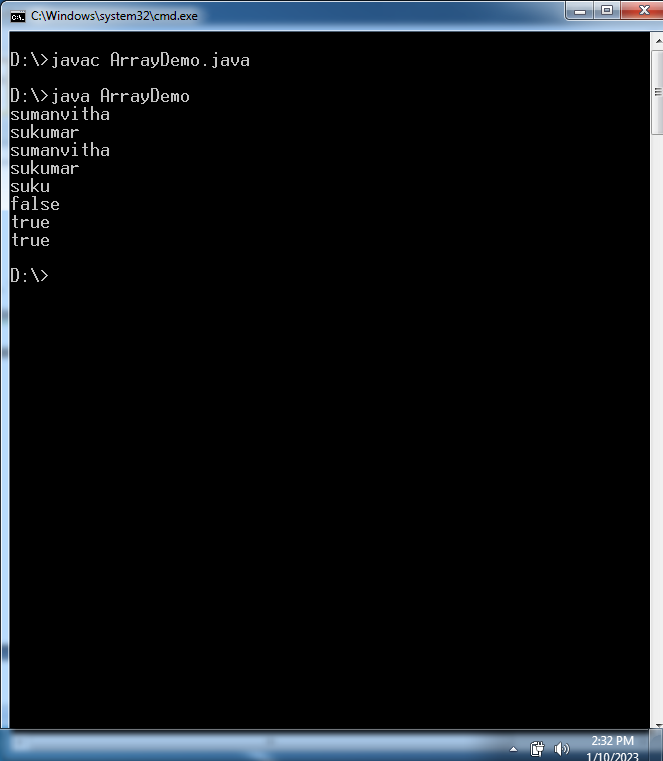
System.out.println(s1==s3);

System.out.println(s1.equals(s3));

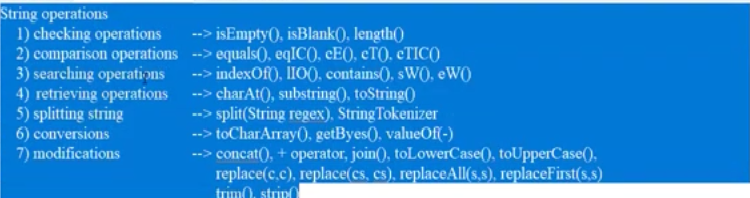
System.out.println(s1==s6);

}

}



**1.2.Methods:-**

****

**1.isEmpty():-**

Syntax:

Public Boolean isEmpty();

If input string is empty then this method returns true otherwise this method returns false.

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1=" ";

String s2=null;

String s3="suku";

String s4=new String("");

String s5=new String("sumanvitha");

String s6=new String(" ");

String s7="";

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

//System.out.println(s2.isEmpty());

System.out.println(s3.isEmpty());

System.out.println(s4.isEmpty());

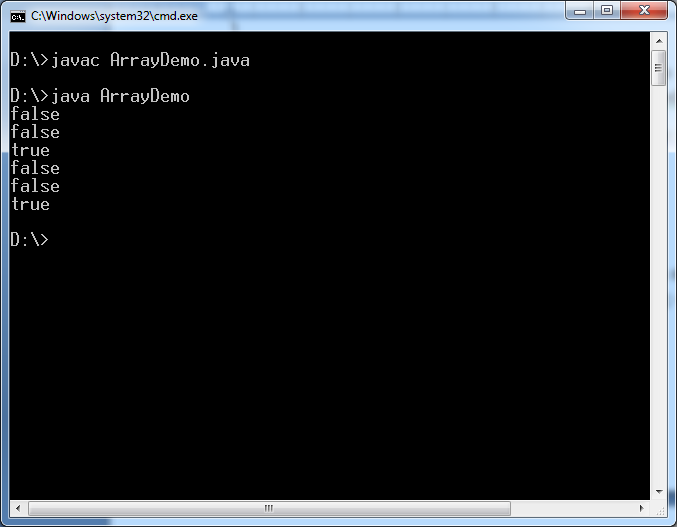
System.out.println(s5.isEmpty());

System.out.println(s6.isEmpty());

System.out.println(s7.isEmpty());

}

}

****

**2. length:-**

Syntax:

Public int length();

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1=" ";

String s2=null;

String s3="suku";

String s4=new String("");

String s5=new String("sumanvitha");

String s6=new String(" ");

String s7="";

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

//System.out.println(s2.length());

System.out.println(s3.length());

System.out.println(s4.length());

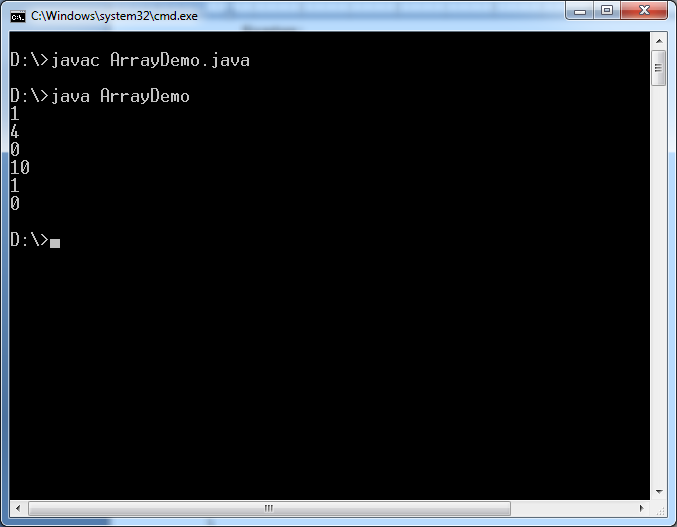
System.out.println(s5.length());

System.out.println(s6.length());

System.out.println(s7.length());

}

}

****

**3.print String object content.**

class ArrayDemo{

public static void main(String args[])

{

String s1="";

String s2=" ";

String s3=null;

String s4="sukumar";

String s5;

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

System.out.println(s4);

//System.out.println(s5); //It leads to CE: variable might not have been initialized.

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

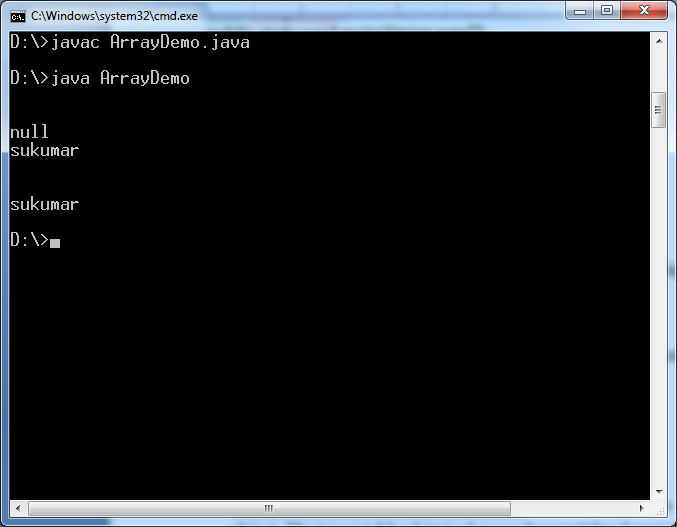
System.out.println(s2.toString());

//System.out.println(s3.toString()); // It leads to CE: Null pointer Exception.

System.out.println(s4.toString());

}

}

****

**4.charAT:-** This method returns a char value at the given index number.

Syntax:

Public char charAt(int index);

Where index number starts from 0 and goes to n-1, n is length of index. If index is > n or <0 , then it returns **StringIndexOutOfBoundsException.**

Example:-

class ArrayDemo{

public static void main(String args[])

{

String s1="";

String s2=" ";

String s3=null;

String s4="sukumar";

String s5;

//System.out.println(s1.charAt(0)); //

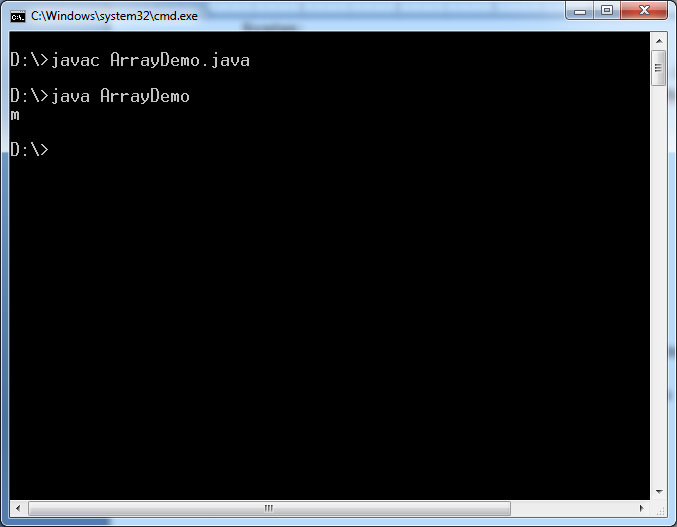
//System.out.println(s4.charAt(-1));// above and this statement leads to CE: StringIndexOutofboundException.

//System.out.println(s3.charAt(0)); This statement leads to CE:Null pointer Exception.

System.out.println(s4.charAt(4));

}

}



**5.Contains:-** This method searches String for sequence of characters. It returns true if the sequence of char values is found otherwise it returns false.

Syntax:

Public Boolean contains(String s);

Example:-

class ArrayDemo{

public static void main(String args[])

{

String s1="";

String s2=" ";

String s3=null;

String s4="sukumar";

String s5;

System.out.println(s1.contains(""));

System.out.println(s1.contains(" "));

System.out.println(s2.contains(" "));

System.out.println(s2.contains("a"));

//System.out.println(s3.contains("s"));

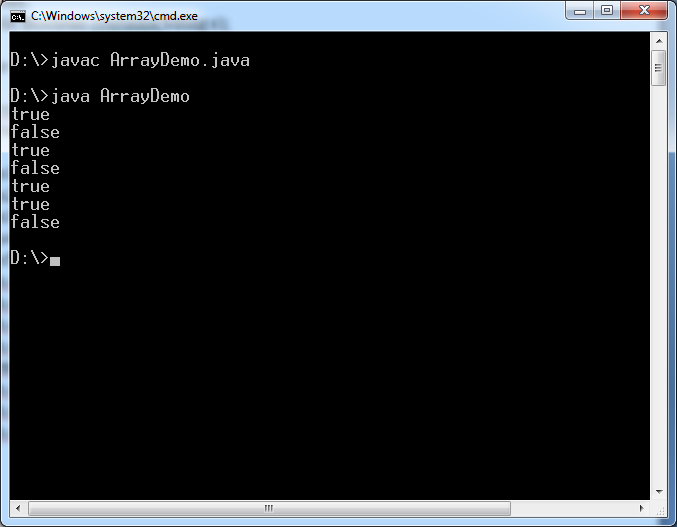
System.out.println(s4.contains("su"));

System.out.println(s4.contains("r"));

System.out.println(s4.contains("x"));

}

}



**6.**

**a. public boolean startsWith(String s)**

**b.public boolean endsWith(String s)**

**Example:**

class ArrayDemo{

public static void main(String args[])

{

String s1="";

String s2=" ";

String s3=null;

String s4="ab cd ef gh";

System.out.println(s1.startsWith(""));

System.out.println(s1.startsWith(" "));

System.out.println(s2.startsWith(""));

System.out.println(s2.startsWith(" "));

System.out.println(s1.endsWith(""));

System.out.println(s1.endsWith(" "));

System.out.println(s2.endsWith(""));

System.out.println(s2.endsWith(" "));

//System.out.println(s3.contains("s"));

System.out.println(s4.startsWith("ab"));

System.out.println(s4.startsWith("ab cd"));

System.out.println(s4.startsWith("abcd"));

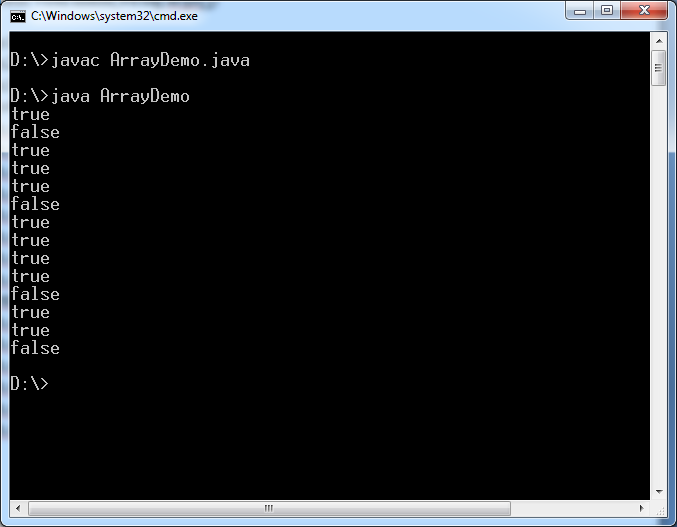
System.out.println(s4.endsWith("gh"));

System.out.println(s4.endsWith(" gh"));

System.out.println(s4.endsWith("fgh"));

}

}



**6.subString:-**

Syntax:-1

Public String substring(int startindex);

In specified string, from specified index to last the values are copied into new String. It returns copied string which is substring of specified string.

Public String substring(int startindex, int endindex);

Where startindex-endindex= n

In specified string, from **startindex and n no.of**  values are copied into new String. It returns copied string which is substring of specified string.

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1="java programming language";

System.out.println(s1.substring(5));

System.out.println(s1.substring(5,10));

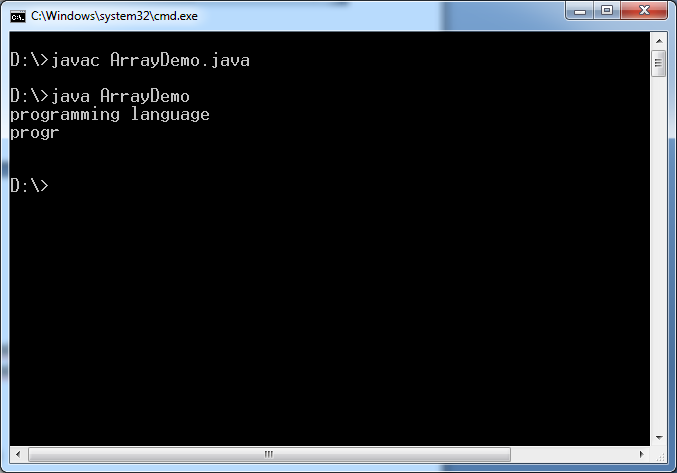
//System.out.println(s1.substring(-1));

System.out.println(s1.substring(6,6));

//System.out.println(s1.substring(-1,10));

}

}



**7.concat:-**  The concat() method appends(concatenate) a string to end another string.

Syntax:

Public String concat(String s);

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1="java programming";

String s2=s1.concat("Language");

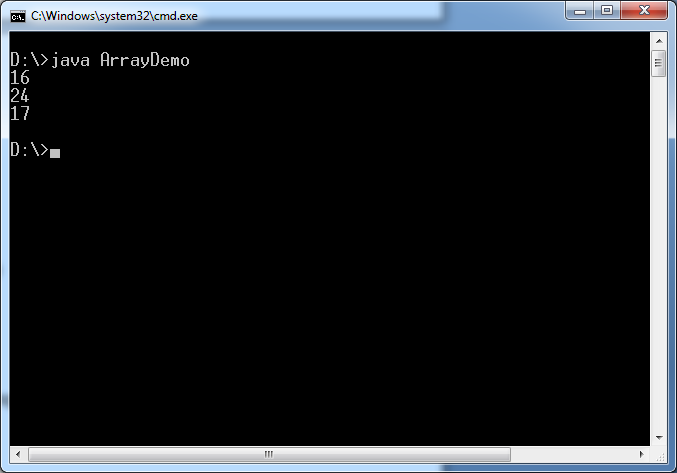
System.out.println(s1.length());

System.out.println(s2.length());

System.out.println(s1.concat(" ").length());

}

}



**8.toUpperCase:-** This method converts a string to uppercase letters.

**Syntax:**

Public String toUpperCase();

**9.toLowerCase:-** This method converts a string to lowercase letters.

Syntax:

Public String toLoweCase();

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1="A.SukuMaR";

String s2=s1.toUpperCase();

String s3=s1.toLowerCase();

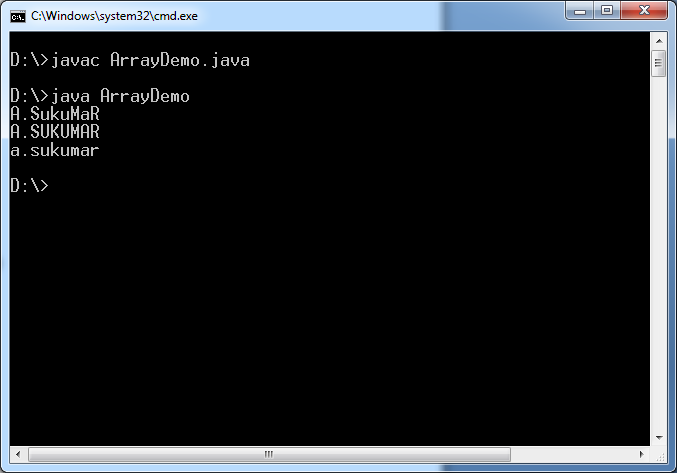
System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}

}



**10.replace:-** The replace() method searches a string for a specified character/string, and returns a new string where the specified character[s] are replace.

The replace **is overloaded** **method** in String class.

Syntax:

Public String replace(char ch);

Public String replace(String str);

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1="A.SukuMaR";

String s2=s1.replace('a','A');

String s3=s1.replace("suku","sv");

String s4=s1.replace("Suku","sv");

String s5=s1.replace("Suku","sukuMCA");

System.out.println(s1);

System.out.println(s2);

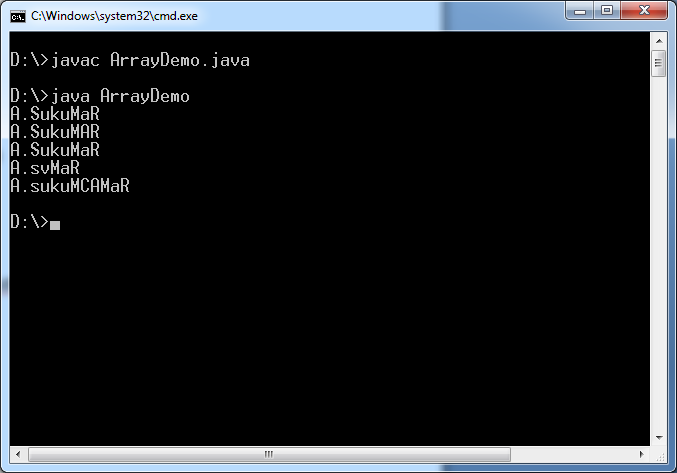
System.out.println(s3);

System.out.println(s4);

System.out.println(s5);

}

}



**11. trim:-** This method removes trailing and leading spaces.

Syntax:

Public String trim();

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1=" A.SukuMaR ";

String s2=" A.Sukumar";

String s3="A.Sukumar ";

String s4=s1.trim();

String s5=s2.trim();

String s6=s3.trim();

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

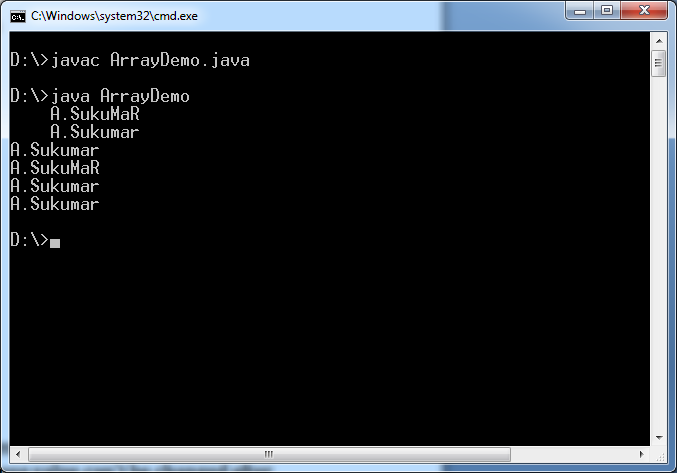
System.out.println(s4);

System.out.println(s5);

System.out.println(s6);

}

}



**12.join:-**

It is available java8 onwards.

This method append “delimiter” to every word except last word .

Syntax:

Public static String join (“delimiter”, CharSequence cs,…);

Example:-

class ArrayDemo{

public static void main(String args[])

{

System.out.println(String.join("--", "sukumar"));

System.out.println(String.join("--", "sukumar","veena"));

System.out.println(String.join("--", "suku","veena","sulamaha"));

String s1=new String("rock");

System.out.println(String.join("--",s1);

System.out.println(String.join("--",s1,s1);

String s2=s1.join("--",s1);

System.out.println(s2);

}

}

Output:

-----------

Sukumar

Sukumar—veena

Suku—veena—sulamaha

Rock

Rock – rock

Rock.

**13. split:-** It is overloaded method.

Syntax:1

Public String[] split(“delimiter”);

This method cuts specified string using delimiter as small parts. Each part becomes string. This method places them in string array. It returns the array reference.

Syntax:2

Public String[] split(“delimiter”, int no.of parts)

This method cuts specified string using delimiter as specified no.of parts. Each part becomes string. This method places them in string array. It returns the that array reference.

Example:

class ArrayDemo{

public static void main(String args[])

{

String s1= new String("Atmakuru Sukumar MCA");

String s2= "C:/jdk-19/sample";

String [] s3= s1.split(" ");

String [] s4= s1.split("");

String [] s6= s1.split(" ",2) ;

for(int i=0;i< s3.length;i++)

{

System.out.println(s3[i]);

}

for(int i=0;i< s4.length;i++)

{

System.out.print(s4[i]);

}

String [] s5=s2.split("/");

System.out.println();

for(int i=0;i<s5.length; i++)

{

System.out.println(s5[i]);

}

for(int i=0;i<s6.length; i++)

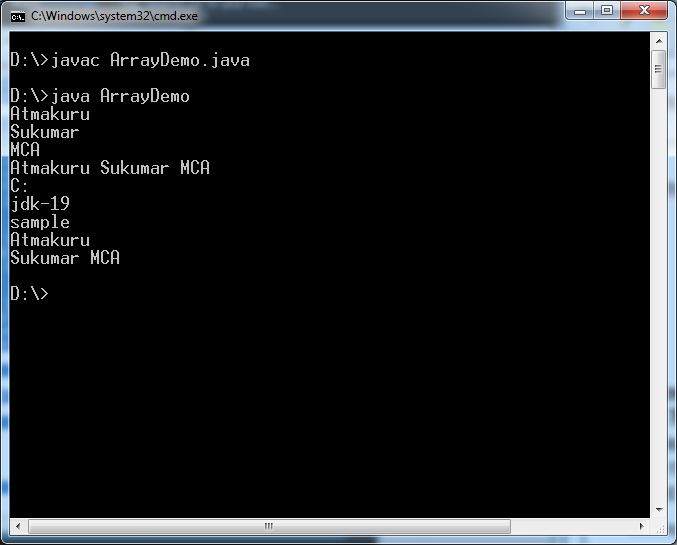
{

System.out.println(s6[i]);

}

}

}



**14.intern():-**

Syntax:

Public String intern();

This method is useful when we create String using new and String class constructor.

This method searches the **String Constant pool** for String object with same content. If such string object is available, then that String object reference is returned instead of creating the new String object in heap .

Otherwise , one String object is created in heap and another string object is created in String constant pool.

Example:

class ArrayDemo

{

public static void main(String args[]){

String s1="sukumar";

String s2="sukumar";

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

String s3=new String("sukumar").intern();

String s4=new String("rock");

String s5=s4.intern();

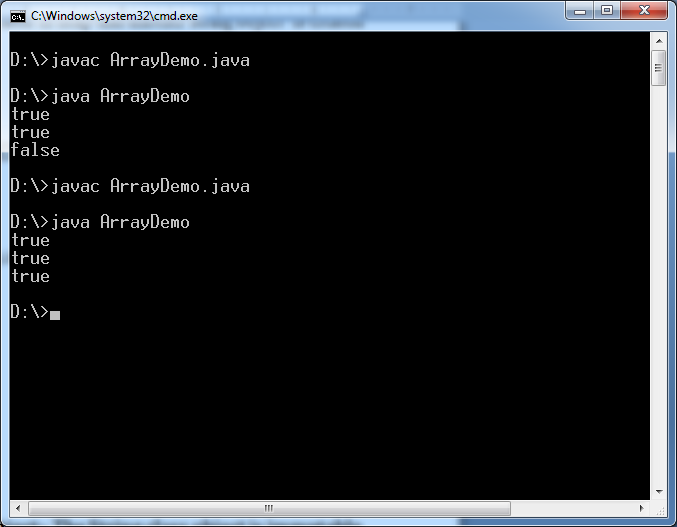
String s6=s4.intern();

System.out.println(s3==s2);

System.out.println(s6==s5);

}

}

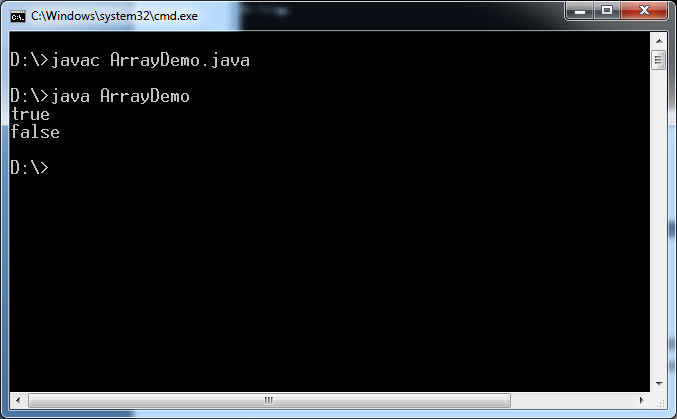


**15.equals:-** This method compare one String object content with another String object content. If both string object contents are same, it returns true. Otherwise it returns false.

Syntax:-

Public Boolean equals(String s);

Example:



16.toCharArray:- This method converts String into charArray.

Syntax:

Public char[] toCharArray()

Example:-

class ArrayDemo

{

public static void main(String args[])

{

String s1="sukumar";

char [] c1= s1.toCharArray();

for(int i=0;i<c1.length;i++)

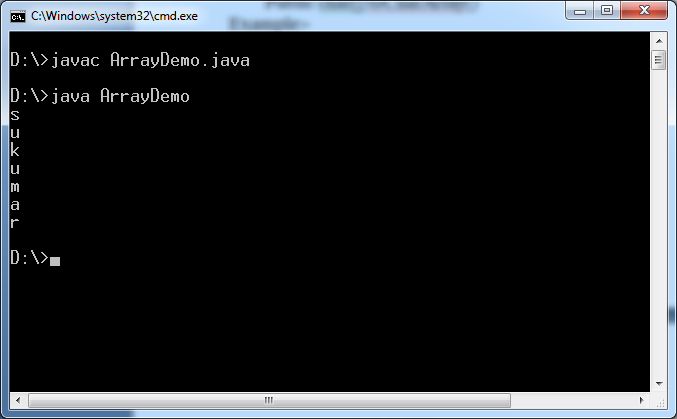
{

System.out.println(c1[i]);

}

}

}



# 1.3.Drawback in String class object:- The String class object is immutable object. The immutable object whose value can’t be changed after initialization.

**(i.**e) once , string is placed in String class object, we can not change it. Therefore string class object is immutable object.

**How is it possible?**

The Actual string is stored in char Array. This Array reference is stored in “value” which is variable in String class. It is final variable. That’why once string is placed in String class object, Changing string is not possible.

Example: String s1=”String”;

S.o.p(s1.upperCase());

Now upperCase goes to s1 object. It read string from s1. It converts string into uppercase . Converted string is kept in new String class object. But s1 value will not be modified.

Q. Why is Strict class object given as immutable?

A.

Reason1: Thourgh network when we send data/values , They goes as String objects.

Uname:”XXX”

Pwd:”YYY”

IPAddress:”200/123.54.0”

Assume: If String class object is not immutable, while string traveling over network Hackers may change it and he store it in same object. Therefore we get rong result even we entered correct values.

That’why string class object is given as “immutable”.

Reason2: To Connect DB, we invoke a getConnection () method. To that , 3 arguments are passed.

1.url of DB.

2.Uname

3. pwd.

These values are placed in String class objects.

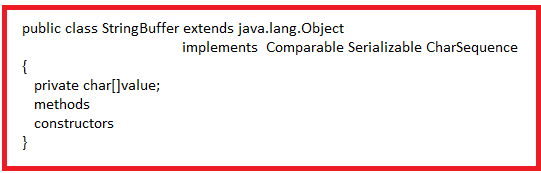
Assume: If String class object is not immutable then somebody may change the url and store it in same object. It leads to connecting to rong db.

That’s why String object is given as immutable.

**1.4. This object can be used in single threaded application and multithreaded application.**

**STRING BUFFER CLASS**

**1.StringBuffer:-** This class is available in java.lang. package.

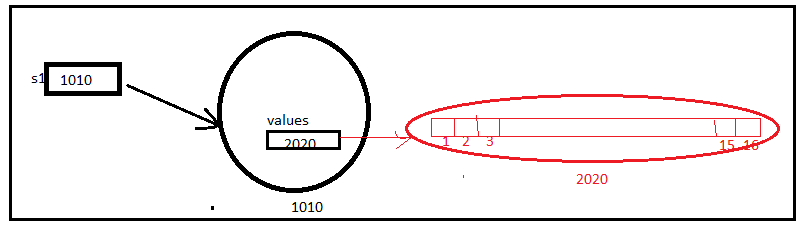


**1.1.Constructors:-**

**a.public StringBuffer():**  It creates empty StringBuffer object with default capacity 16.

Example:-

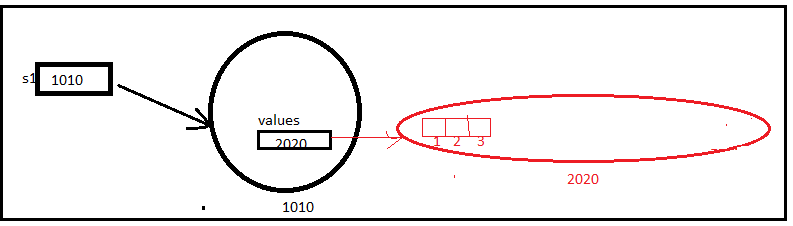
StringBuffer s1=new StringBuffer();



**b.public StringBuffer(int Capacity):** It creates empty String Buffer object with given capacity .

Example:-

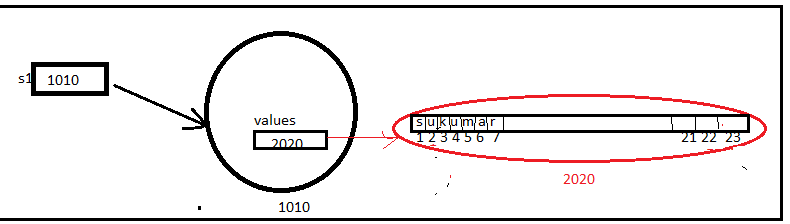
StringBuffer s1=new StringBuffer(3);



**C.public StringBuffer(String s):** It creates StringBuffer object with given String object content. It performs string copy from String object to String buffer object. The default capacity is **16+s.length**.

Example:

StringBuffer s1=new StringBuffer(“sukumar”);



**Note:-1** we should not pass “null” to StringBuffer constructor.

It leads to CE: **NullPointerException**.

Example:

StringBuffer b1=new StringBuffer(null); // It Leads to CE: NullPointerException.

**Note:-2** The StringBuffer object is only created using “new” and “ StringBuffer constructor”.

Example:-

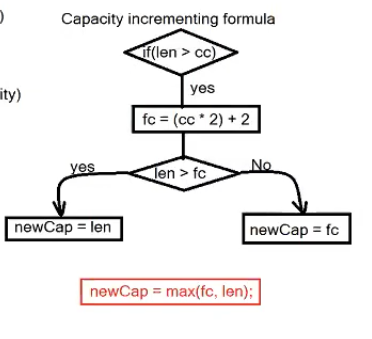
1.StringBuffer b1=new StringBuffer();//valid

2.StringBuffer b2=new StringBuffer(3);//valid

3.StringBuffer b3=new StringBuffer(“rock”); //valid

**4.StringBuffer b4=”sukumar**”; **//It is invalid.**

**1.2StringBuffer capacity calculation Formula:**

****

**1.2. The following operations , we can perform on StringBuffer content**.

**1**.**setCharAt**:- This method replace char at specified index with given character..

Syntax:-

Public synchronized StringBuffer setCharAt(int index, char ch);

**2.reverse:-** This method reverse the character sequence.

Syntax:-

Public synchronized StringBuffer reverse();

**Example**:

class ArrayDemo

{

public static void main(String args[])

{

StringBuffer s1=new StringBuffer("sukumar");

System.out.println("Before Replacing:"+s1);

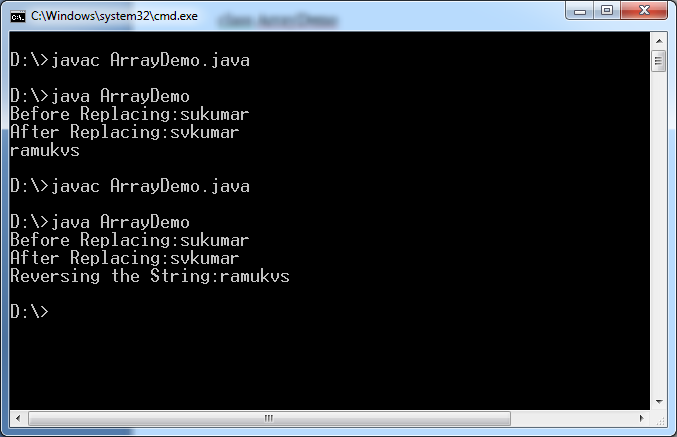
s1.setCharAt(1,'v');

System.out.println("After Replacing:"+s1);

System.out.println("Reversing the String:"+s1.reverse());

}

}



**3.Delete:-**

Syntax:1

**Public synchronized StringBuffer deleteCharAt(int index);**

It deletes the character from specified position in StringBuffer object content .

Syntax:2

**Public synchronized StrinigBuffer delete(int start ,int end);**

**Example:**

class ArrayDemo

{

public static void main(String args[])

{

StringBuffer s1=new StringBuffer("sukumar Atmakuru");

System.out.println("Before Delete:"+s1);

s1.deleteCharAt(0);

System.out.println("After Deleting char at 0th position:"+s1);

s1.deleteCharAt(0);

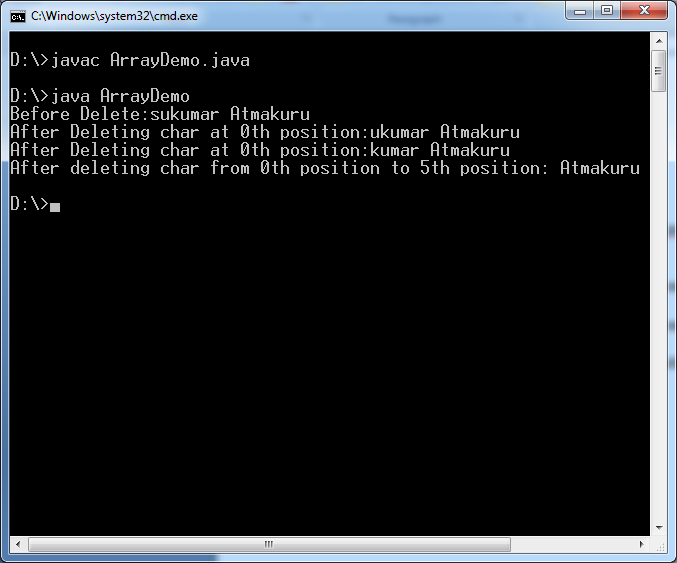
System.out.println("After Deleting char at 0th position:"+s1);

s1.delete(0,5);

System.out.println("After deleting char from 0th position to 5th position:"+s1);

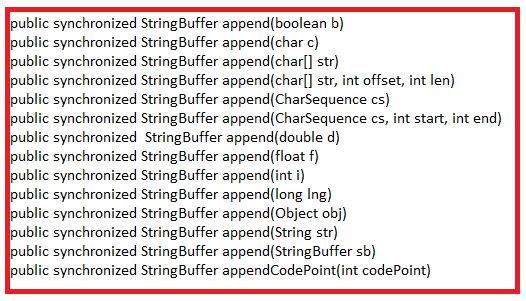
}

}



**4.append:-**  It is overloaded method. While appending , JVM may uses capacity calculation formula.

Syntax:



Example:-

class ArrayDemo

{

public static void main(String args[])

{

StringBuffer s1=new StringBuffer();

System.out.println("SB Capacity:"+ s1.capacity());

System.out.println("SB Length:"+s1.length());

System.out.println("String Buffer Object Content:"+ s1);

StringBuffer s2=new StringBuffer(1);

System.out.println("SB Capacity:"+ s2.capacity());

System.out.println("SB Length:"+s2.length());

System.out.println("String Buffer Object Content:"+ s2);

StringBuffer s3=new StringBuffer("abc");

System.out.println("SB Capacity:"+ s3.capacity());

System.out.println("SB Length:"+s3.length());

System.out.println("String Buffer Object Content:"+ s3);

StringBuffer s4=new StringBuffer(3);

s4.append("abc");

System.out.println("SB Capacity:"+ s4.capacity());

System.out.println("SB Length:"+s4.length());

System.out.println("String Buffer Object Content:"+ s4);

StringBuffer s5=new StringBuffer(3);

s5.append("abcd");// capacity calculation(flowchart) procedure is followed by JVM

System.out.println("SB Capacity:"+ s5.capacity());

System.out.println("SB Length:"+s5.length());

System.out.println("String Buffer Object Content:"+ s5);

StringBuffer s6=new StringBuffer(3);

s6.append("Atmakuru Sukumar veena sula suma suha");// capacity calculation(flowchart) procedure is followed by JVM

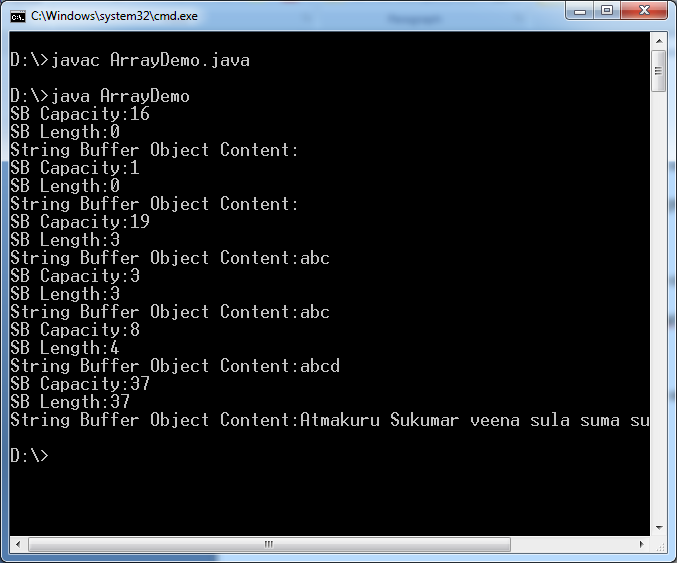
System.out.println("SB Capacity:"+ s6.capacity());

System.out.println("SB Length:"+s6.length());

System.out.println("String Buffer Object Content:"+ s6);

}

}



Note:-

StringBuffer b1= new StringBuffer((String null)); // It is valid.

**5.insert:-** This command inserts a character or sequence of characters at specified place in StringBuffer object content. While inserting , JVM may use capacity calculation formula. This is overloaded method.

By using insert method, we can also do appending operation.

Syntax:

Public synchronized StringBuffer insert(int index, Char c);

Public synchronized StringBuffer insert(int index, CharSequence ch);

Where index should be >=0 and index should be <= length.

Example:

class ArrayDemo

{

public static void main(String args[])

{

StringBuffer s1= new StringBuffer(3);

System.out.println("Capacity:"+s1.capacity());

System.out.println("length:"+s1.length());

System.out.println(s1);

s1.insert(0,'s');

System.out.println("Capacity:"+s1.capacity());

System.out.println("length:"+s1.length());

System.out.println(s1);

s1.insert(1,'v');

System.out.println("Capacity:"+s1.capacity());

System.out.println("length:"+s1.length());

System.out.println(s1);

s1.insert(2,"suma");

System.out.println("Capacity:"+s1.capacity());

System.out.println("length:"+s1.length());

System.out.println(s1);

s1.insert(3,"sulakshmi suhanvitha");

System.out.println("Capacity:"+s1.capacity());

System.out.println("length:"+s1.length());

System.out.println(s1);

s1.insert(s1.length(),"MCA");// Here appending is done.

System.out.println("Capacity:"+s1.capacity());

System.out.println("length:"+s1.length());

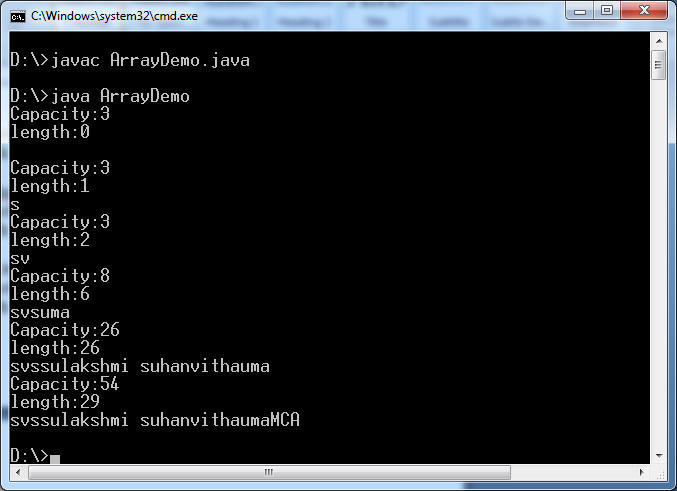
System.out.println(s1);

s1.insert(s1.length(),(String)null);

//s1.insert(s1.length(),(char[])null);

}

}



**6. capacity:-** It is number. The StringBuffer object can hold this number of elements.

Syntax:

Public int capacity();

**7.length:-** It is also number. This number of characters are currently present in stringBuffer object.

Syntax:

Public int length();

**8.trimToSize:-** It reduces Stringbuffer to its size.

Syntax:

Public Synchronized void trimToSize()

Example:-

StringBuffer b1=new StringBuffer(“abc”);

System.out.println(b1.capacity());//19

System.out.println(b1.size()); // 3

B1.trimTosize();

System.out.println(b1.capacity()); //3

**9.ensureCapacity:**

Syntax:

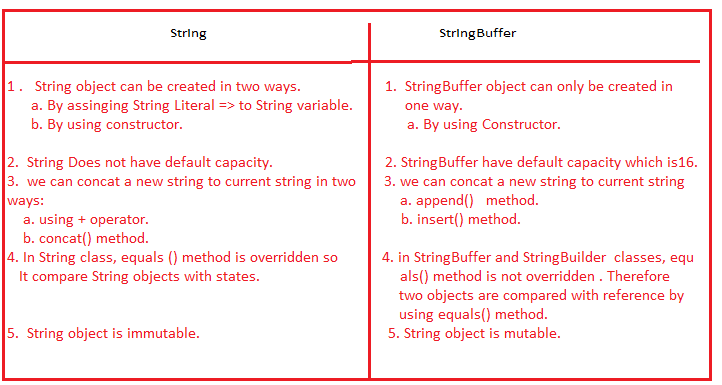
Public void ensureCapacity(int minimumCapacity);

**10.setLength:**

Syntax:

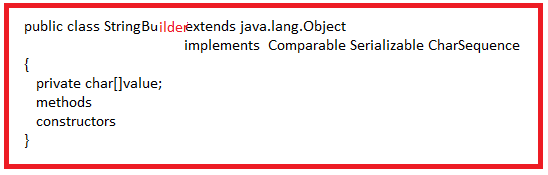
Public void setLength(int newLength);

**DIFFERENCE BETWEEN STRING CLASS AND STRING BUILDER CLASS**

****

**STRING BUILDER CLASS**

**1.String Builder**:- This class is available in java.lang. package.



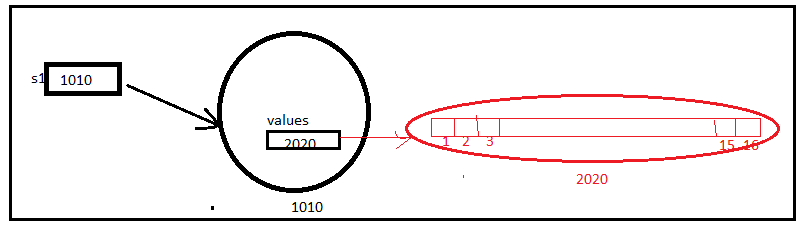
The StringBuilder object is mutable object and It is to be used in SingleThread application Because It ‘s methods are not synchronized methods.

**1.1.Constructors:-**

**a.public StringBuilder():**  It creates empty StringBuilder object with default capacity 16.

Example:-

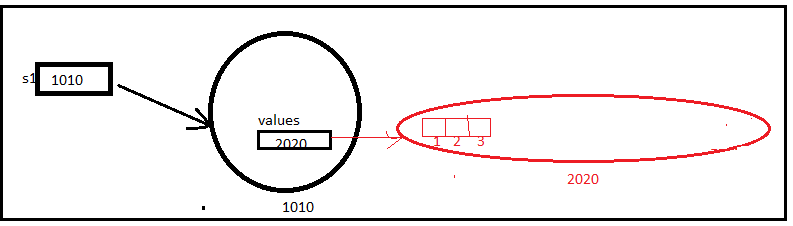
StringBuilder s1=new StringBuilder();



**b.public StringBuilder(int Capacity):** It creates empty String Buffer object with given capacity .

Example:-

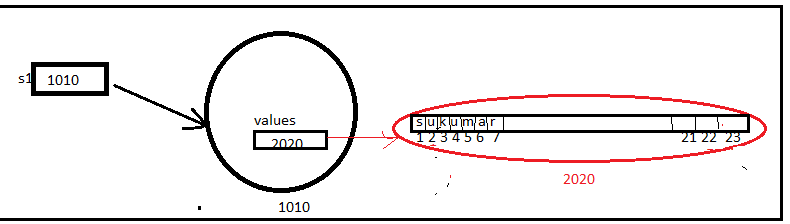
StringBuilder s1=new StringBuilder(3);



**C.public StringBuilder(String s):** It creates StringBuilder object with given String object content. It performs string copy from String object to String builder object. The default capacity is **16+s.length**.

Example:

StringBuilder s1=new StringBuilder(“sukumar”);



**Note:-1** we should not pass “null” to StringBuilder constructor.

It leads to CE: **NullPointerException**.

Example:

StringBuilder b1=new StringBuilder(null); // It Leads to CE: NullPointerException.

**Note:-2** The StringBuilderobject is only created using “new” and “ StringBuilder constructor”.

Example:-

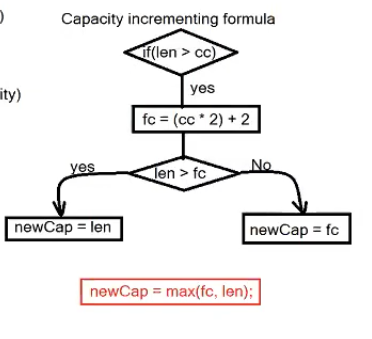
1.StringBuilder b1=new StringBuilder();//valid

2.StringBuilder b2=new StringBuilder(3);//valid

3.StringBuilder b3=new StringBuilder(“rock”); //valid

**4.StringBuilder b4=”sukumar**”; **//It is invalid.**

**1.2StringBuffer capacity calculation Formula:**

****

**1.2.Methods:**

a.setCharAt(int index,char c);

b. reverse()

c.deleteCharAt(int index);

d. delete(int start,int end)

e. insert(int index,XXX ch)

f.append(XXX ch).

g. capacity()

h. length();

i.trimToSize()

j.setLength(int new length)

k.ensureCapacity(int minimumcapacity).