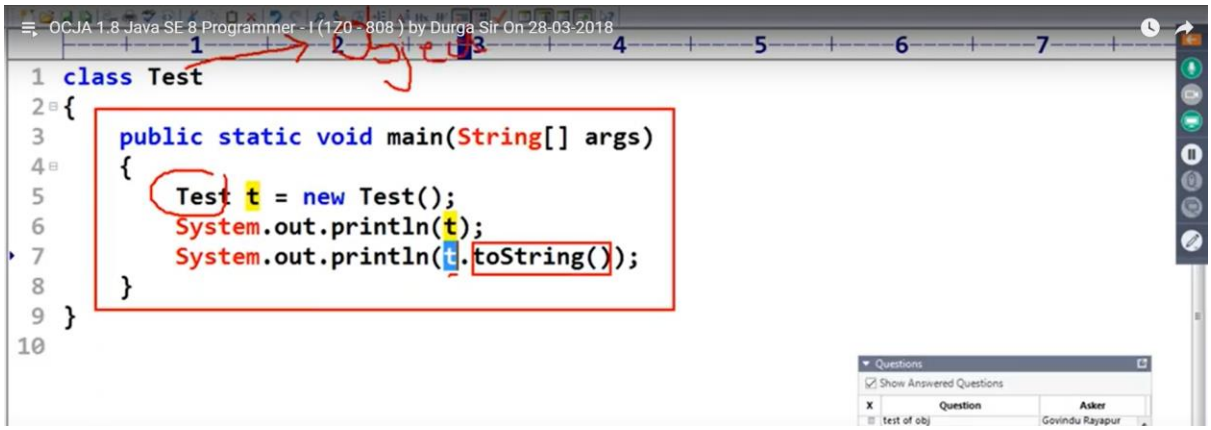
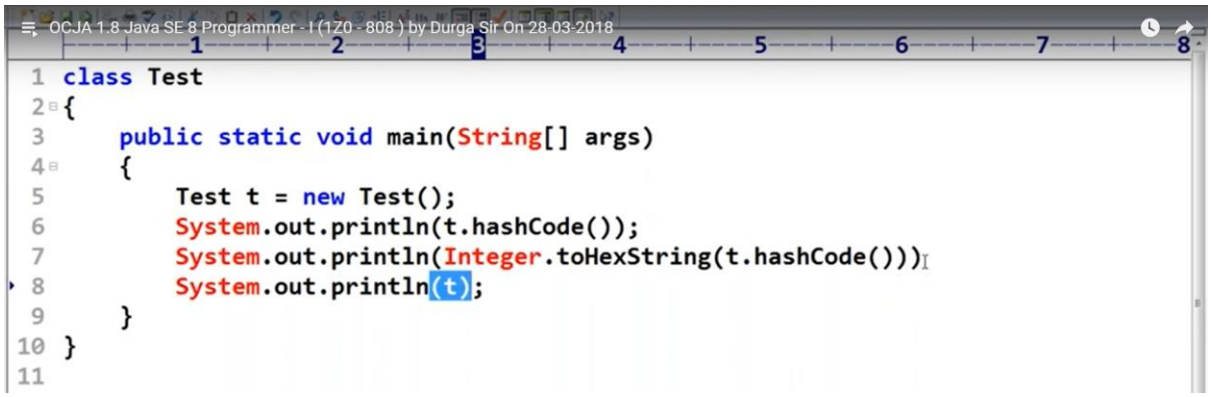
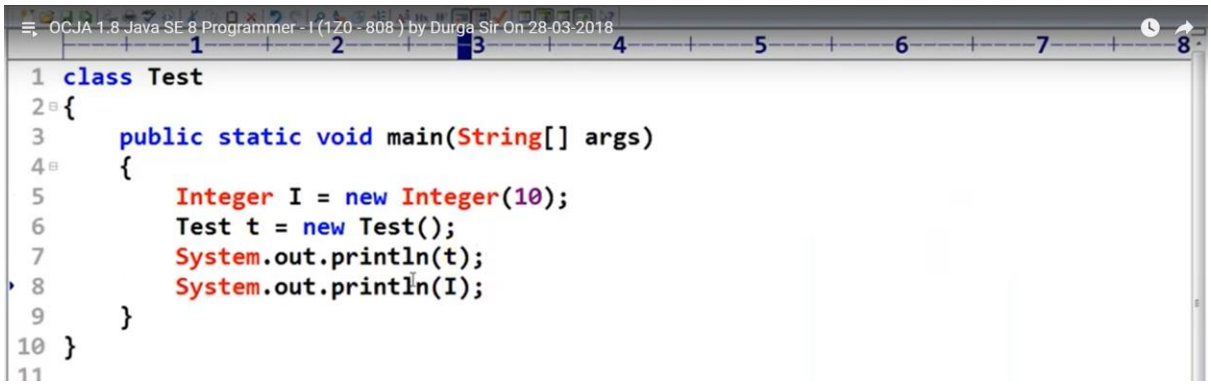


Syout(t) will call t.toString() method. If the Test Class does not have a toString Method it will call the parent class toString method.

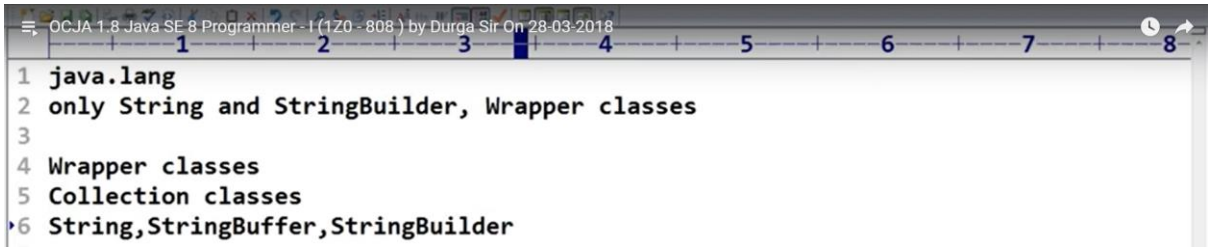
Which will print the “class name” followed by “@” followed by hexadecimal string representation of the HashCode .



The hashcode is converted to HexaDecimal below

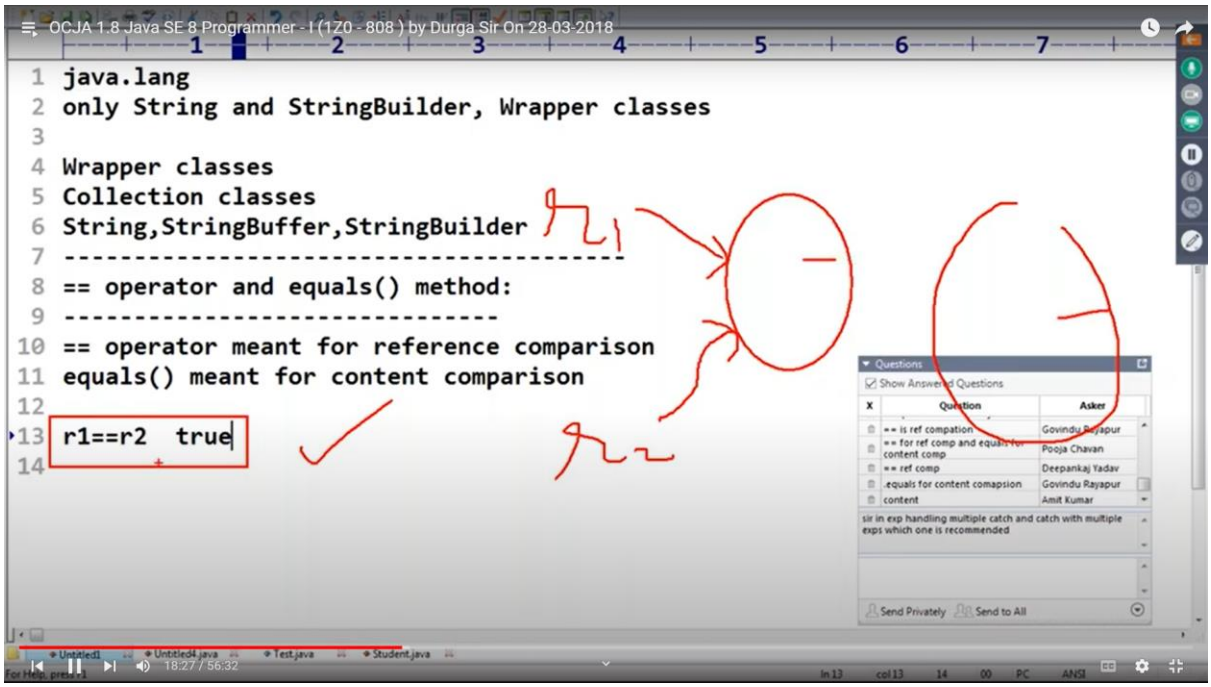


Integer class has overriden the toString method to return the content



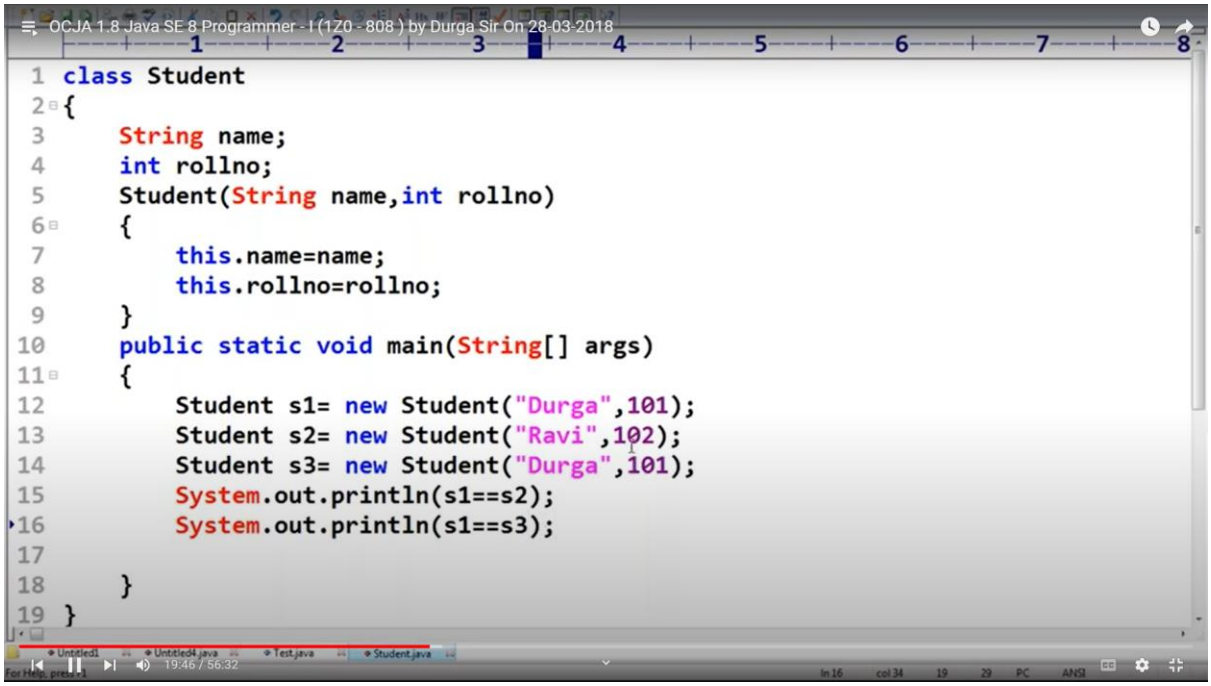
In all String, Wrapper, collection classes , StringBuffer and String Builder the toString method is overridden to return the actual value

Equals method



Equals method is used to compare the content

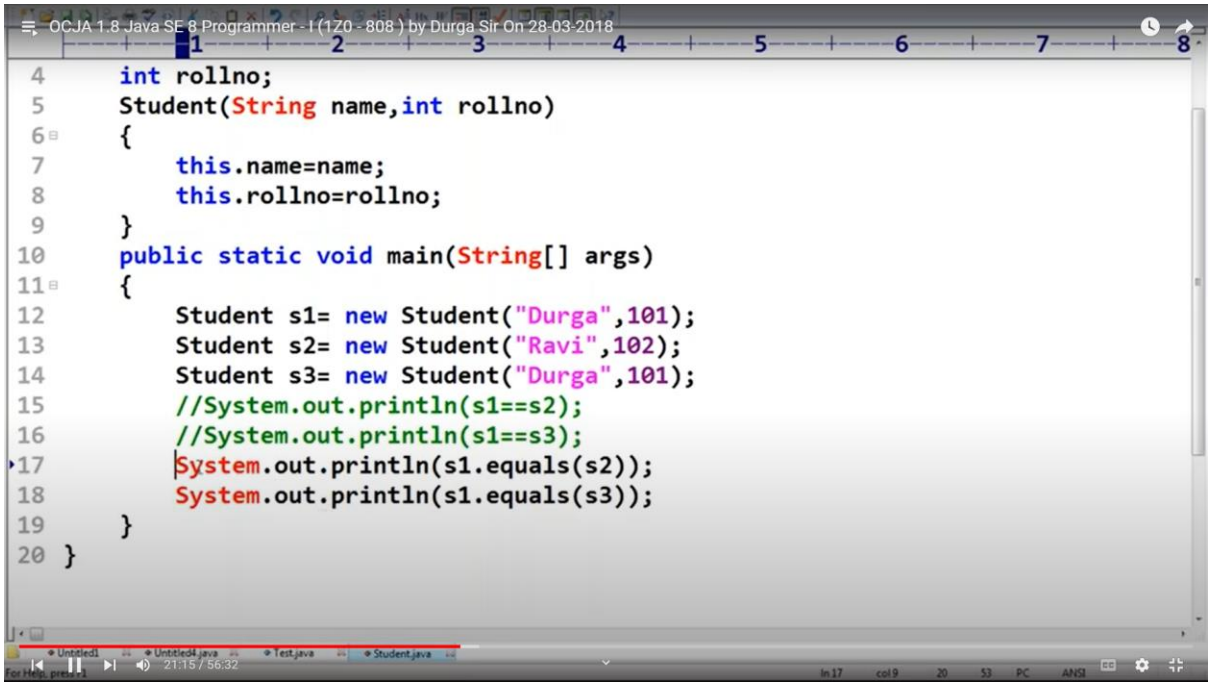
== method to compare the ref



False

False

Both are diff objects used new operator

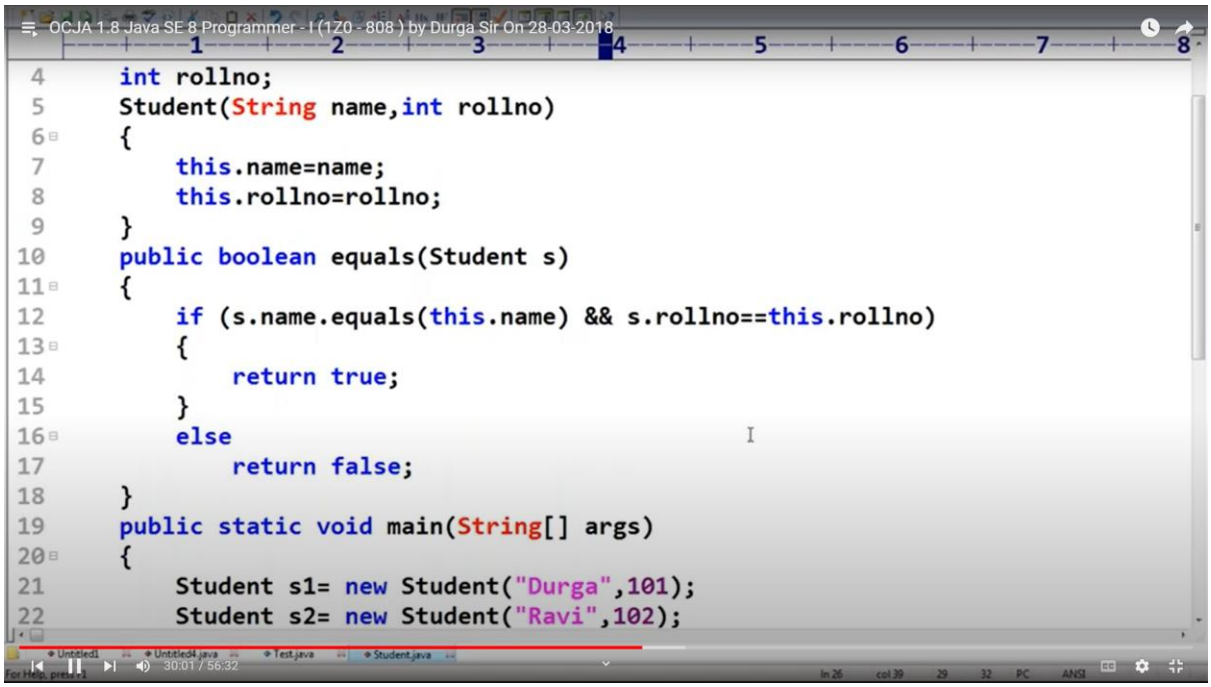


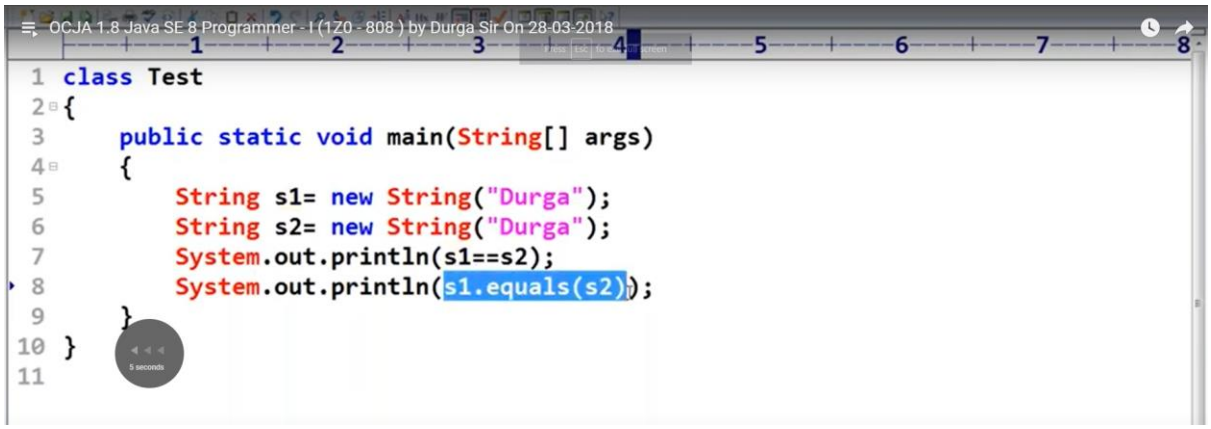
False

True

The class does not contain equals method so parent class “Object class” equals method is used

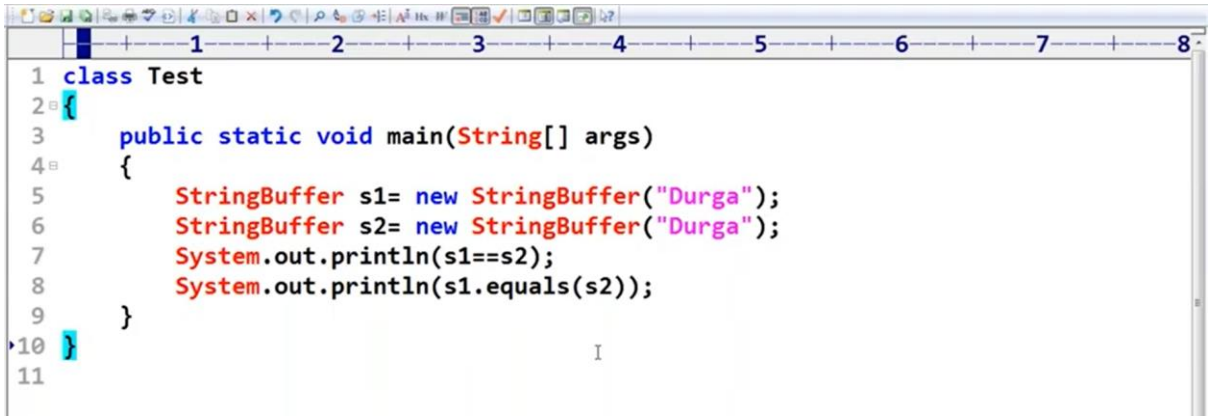
Overriding equals method now Student class equals method will be called






```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
false
true
```

String class – overrides equals method to perform content comparison. Object class .equals method is used for Ref Comparison



```
class Test
{
    public static void main(String[] args)
    {
        StringBuffer s1= new StringBuffer("Durga");
        StringBuffer s2= new StringBuffer("Durga");
        System.out.println(s1==s2);
        System.out.println(s1.equals(s2));
    }
}
```

False

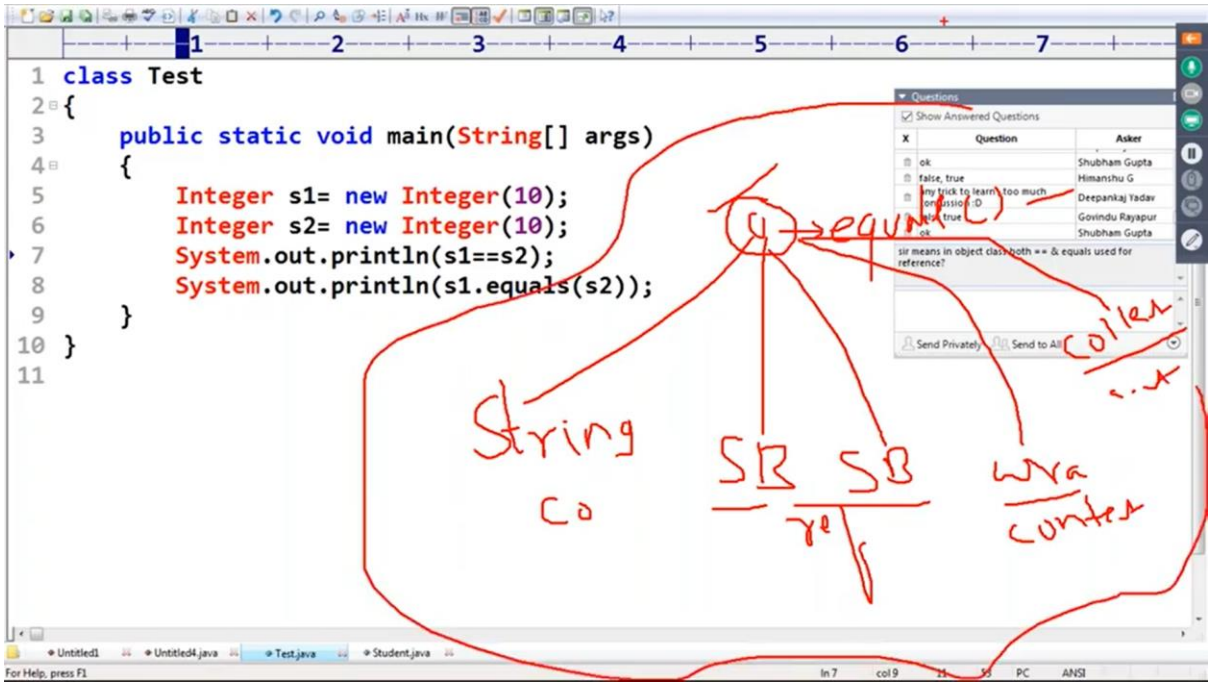
False

In StringBuffer or StringBuilder .equals method are not overridden to perform content comparison

All wrapper classes, collections and String equals method is overridden to perform content comparison

OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 28-03-2018

```
class Test
{
    public static void main(String[] args)
    {
        Integer s1= new Integer(10);
        Integer s2= new Integer(10);
        System.out.println(s1==s2);
        System.out.println(s1.equals(s2));
    }
}
```



StringBuilder and StringBuffer -> calls the equals method of the object class and does ref comparison

CE

```
class Test
{
    public static void main(String[] args)
    {
        String s= new String("Durga");
        StringBuffer sb= new StringBuffer("Durga");
        System.out.println(s==sb);
        System.out.println(s.equals(sb));
    }
}
```

When using == operator

Btw two object types there should be some relationship between arg type either child ->Parent

Or parent -> child

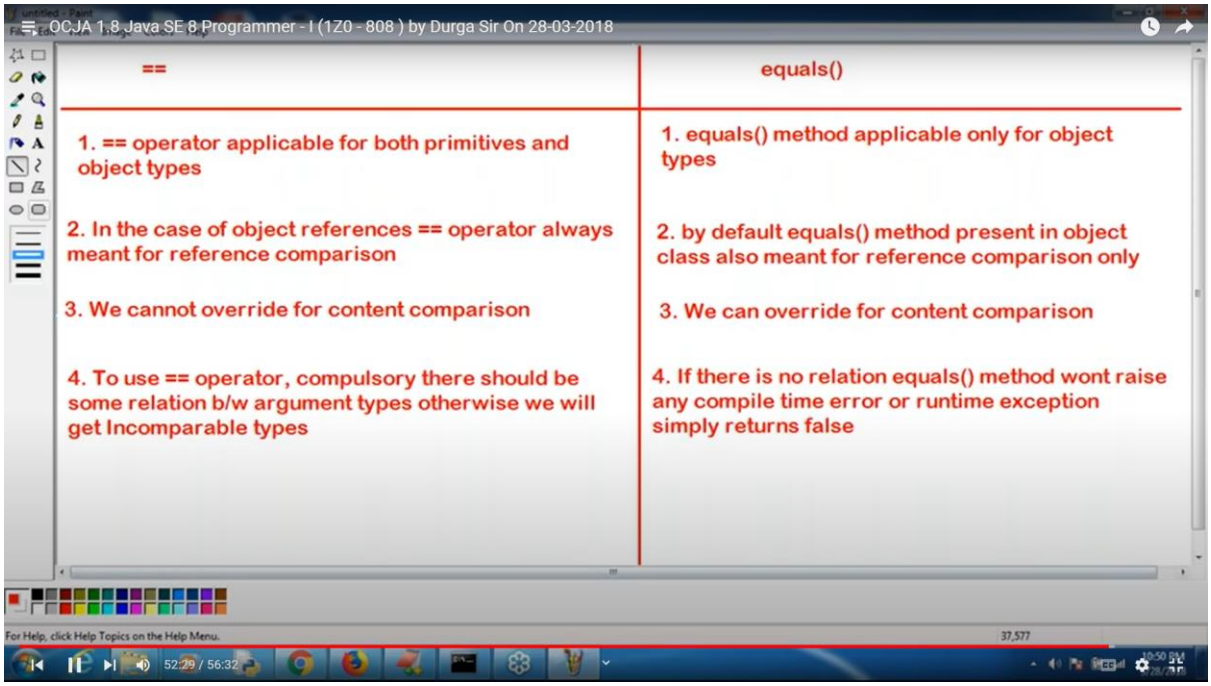
```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          String s= new String("Durga");
6          StringBuffer sb= new StringBuffer("Durga");
7          System.out.println(s==sb);
8          System.out.println(s.equals(sb));
9      }
10 }
11
```

```
C:\Windows\system32\cmd.exe
D:\durgaclasses>javac Test.java
Test.java:7: error: incomparable types: String and StringBuffer
                System.out.println(s==sb);
                                   ^
1 error
```

.equals method will not rise CE even if there are no relationship. Either true or false

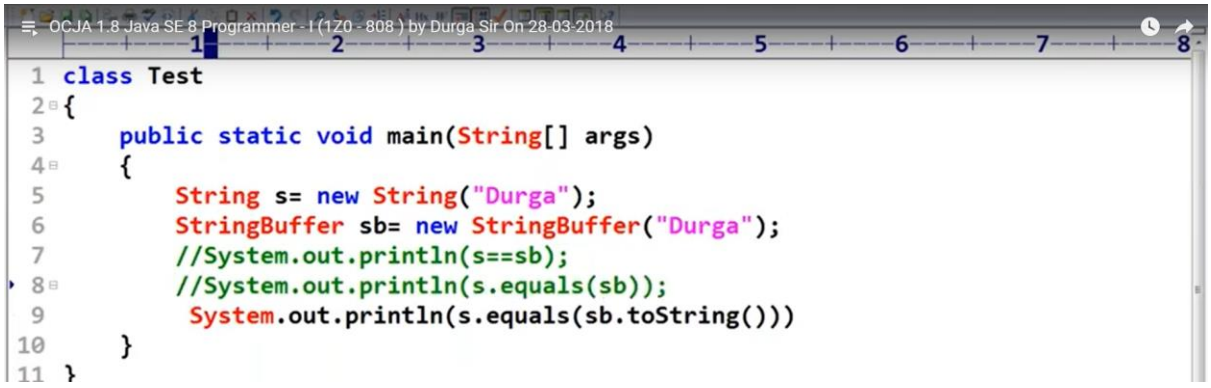
False -> Output

```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          String s= new String("Durga");
6          StringBuffer sb= new StringBuffer("Durga");
7          //System.out.println(s==sb);
8          System.out.println(s.equals(sb));
9      }
10 }
11
```



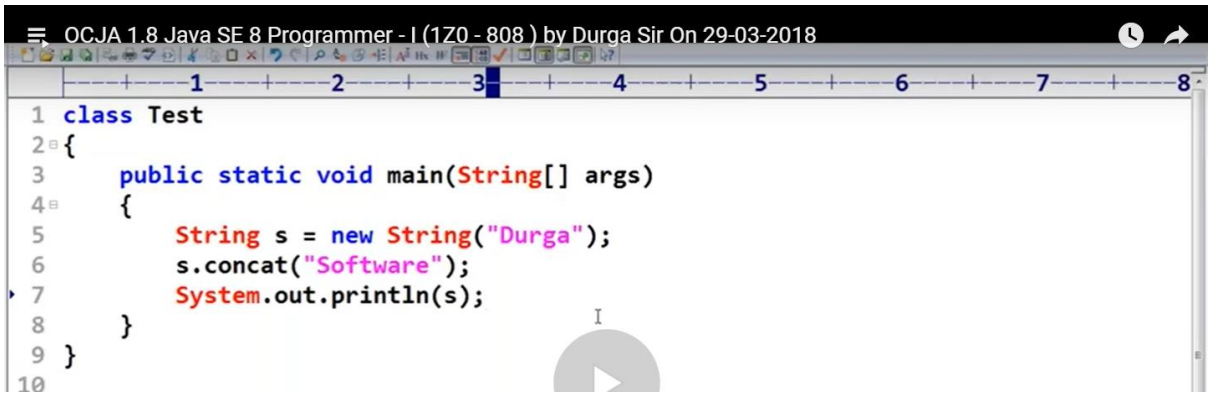
Sb.toString-> return String only.

So comparison of String content -> True




```
class Test
{
    public static void main(String[] args)
    {
        String s= new String("Durga");
        StringBuffer sb= new StringBuffer("Durga");
        String s1=sb.toString();
        System.out.println(s1);
        System.out.println(s.equals(s1));
    }
}
```

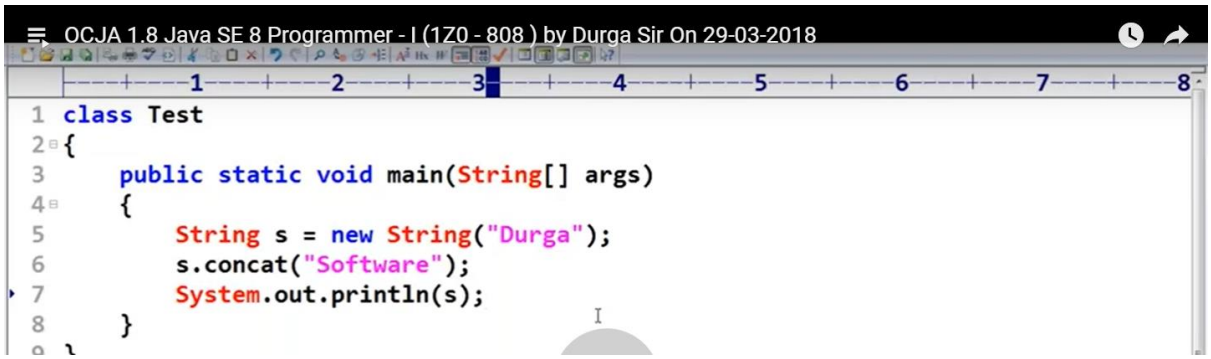
```
case-1:
-------
String immutable
StringBuffer ==>mutable
```



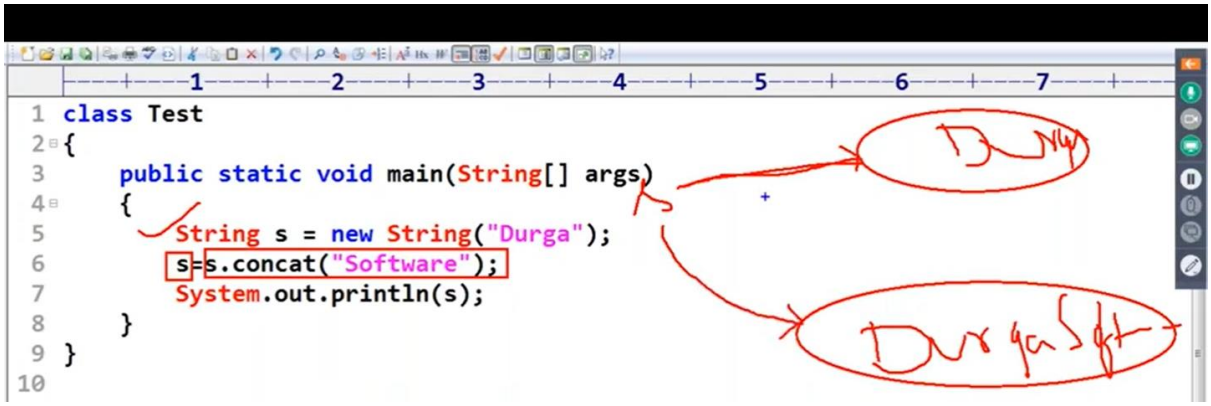
One string object is created , we are not allowed to change its content. If u to try to change a new object will be created and the original object is unaffected

:

Line6- creates a new object but is not assigned to any variable

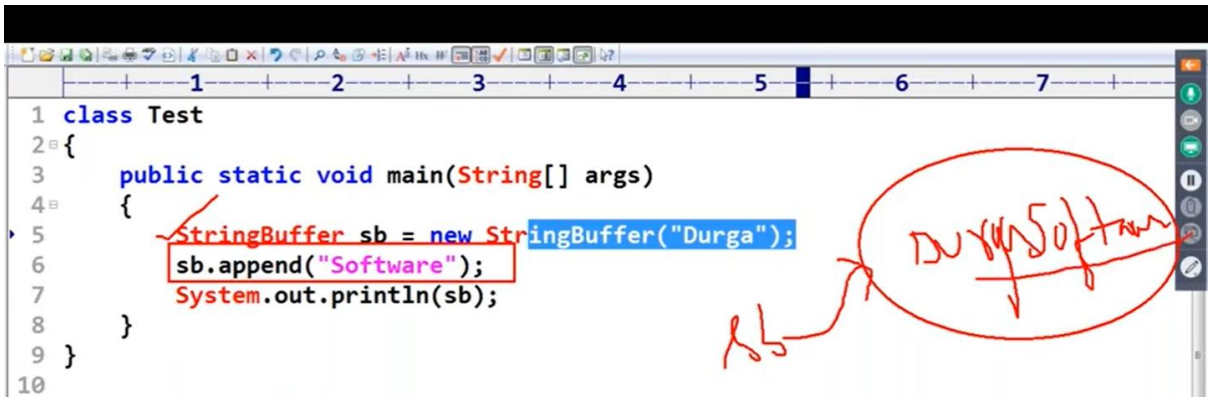


A new object is created and assigned to ref variable s.



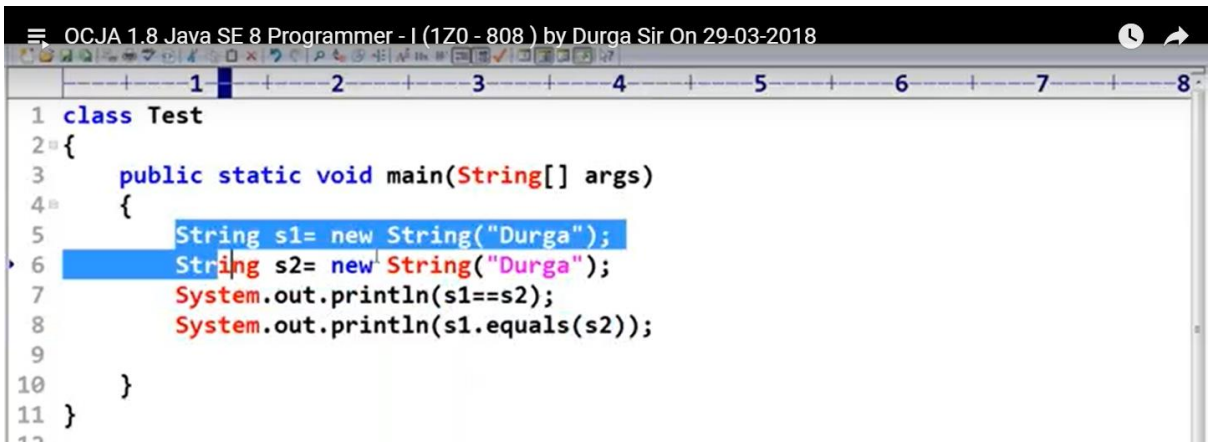
String buffer and Builder same –

Since they are mutable- the content of the string can be modified



False

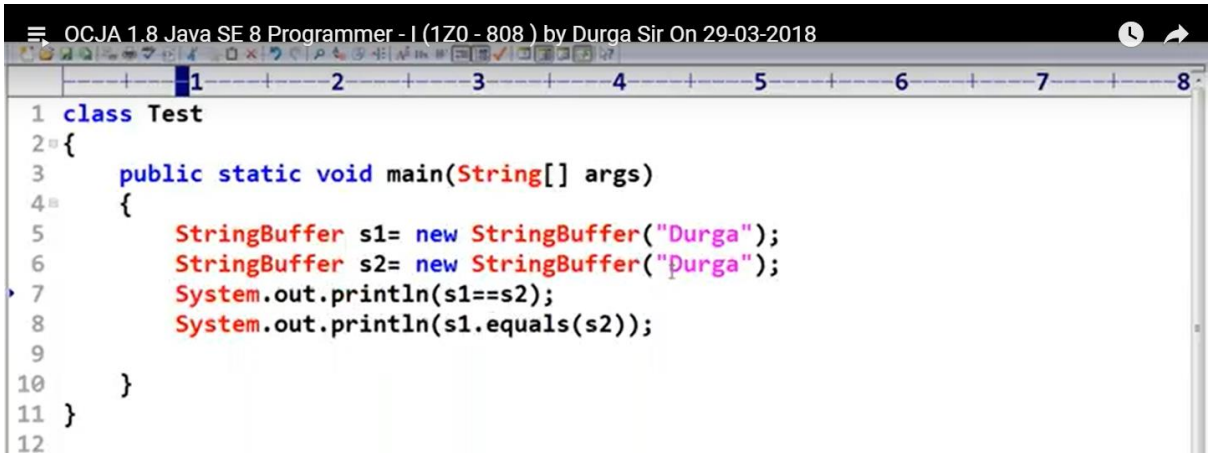
True



False

False

StringBuffer and Builder == and equals method both are not overridden hence object class equals method will be called performing ref comparison

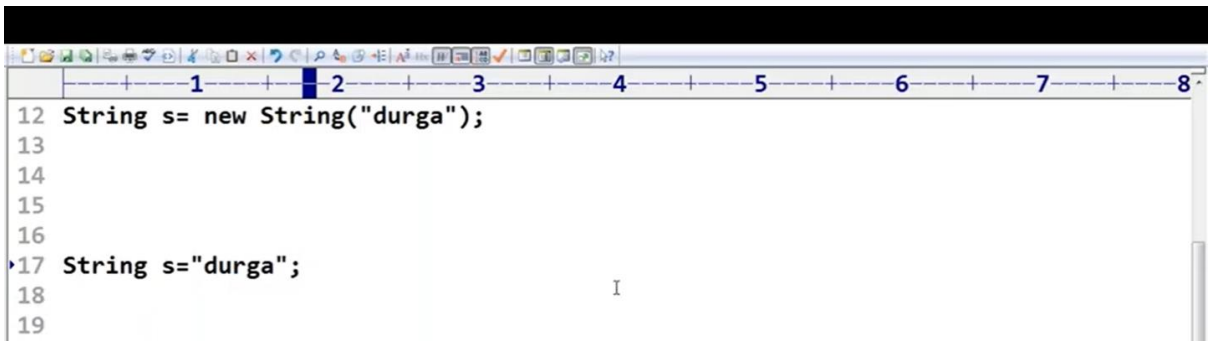


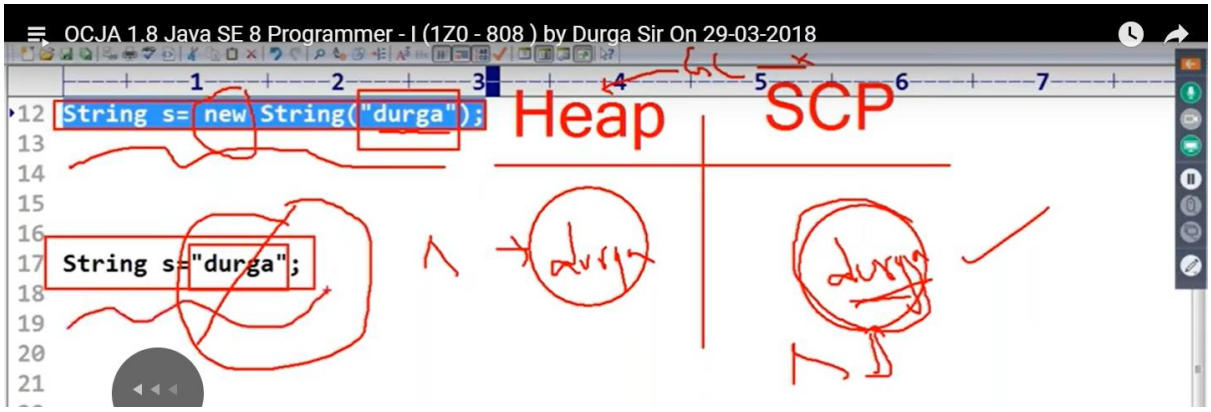
In this scenario, two objects will be created

12- Heap area new object will be created. Its constant , and another object gets created in SCP area.

S will be pointed to Heap Area only. The object in SCP has no ref but still will not be eligible for garbage collection and will be used in future

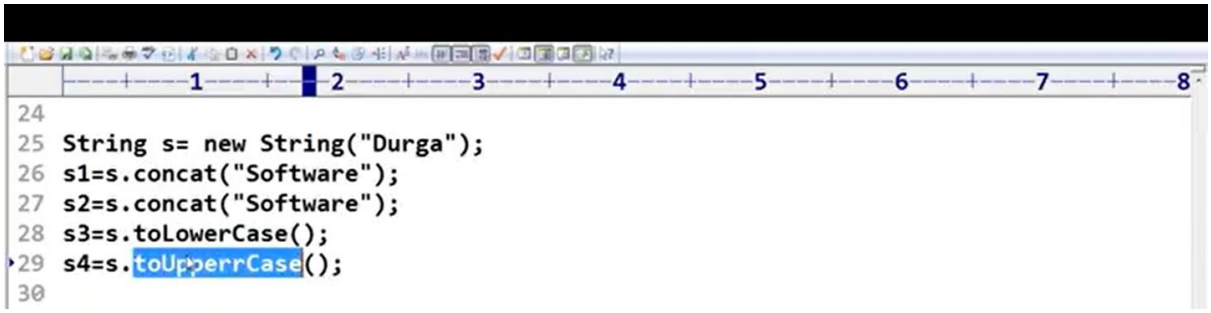
17-> when no new operator is used. JVM checks if there is an existing object with value “durga”, if its there then will create a variable with ref to the existing object. Else a new object will be created





Restarting JVM will be refreshed and SCP area will be refreshed. If you think this would cause memory problem

Any run time operation on String at run time like method call a new object will be created in Heap Area only and not in SCP area. SCP contains only String Constants

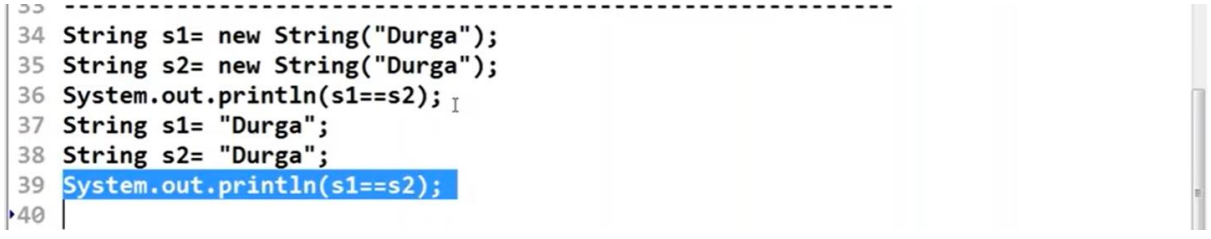


```
24
25 String s= new String("Durga");
26 s1=s.concat("Software");
27 s2=s.concat("Software");
28 s3=s.toLowerCase();
29 s4=s.toUpperCase();
30
```

Line 37-> Durga will be created in SCP

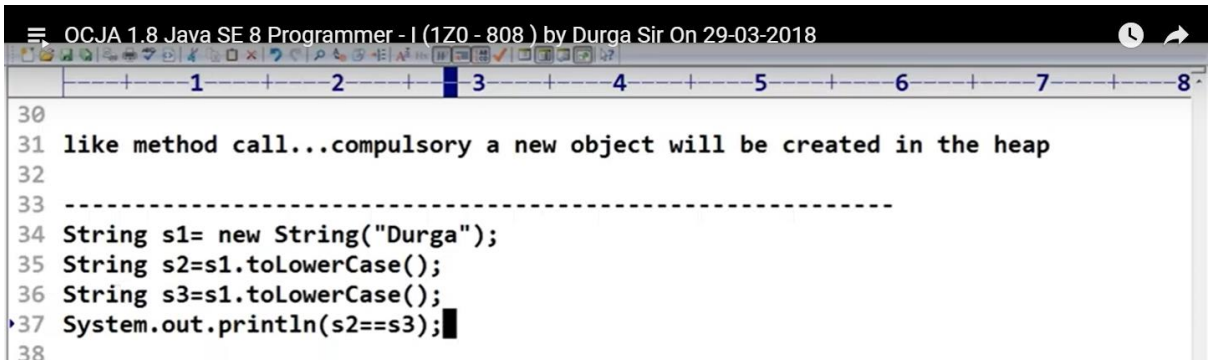
Line 38-> Since Durga is already there s2 will point to Durga ref, hence both will have the same ref

True



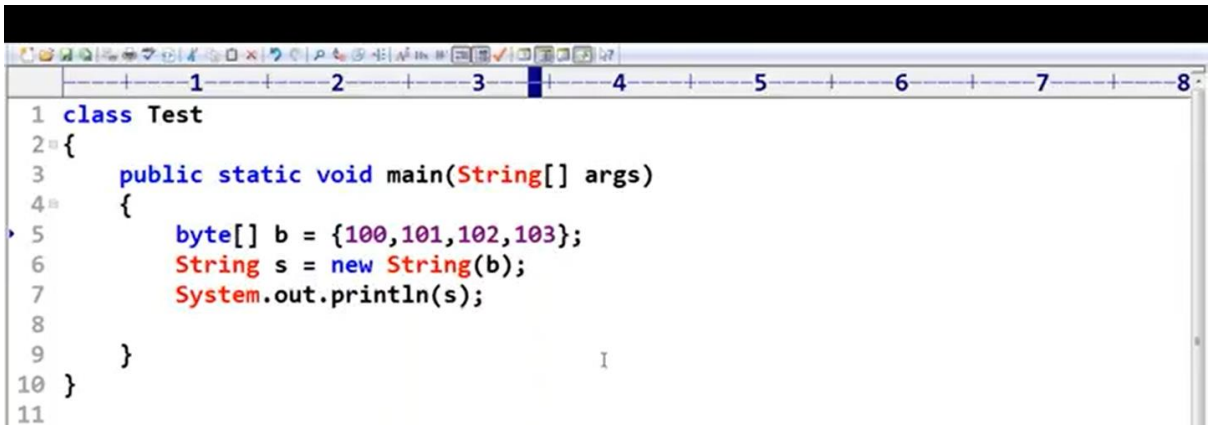
```
34 String s1= new String("Durga");
35 String s2= new String("Durga");
36 System.out.println(s1==s2);
37 String s1= "Durga";
38 String s2= "Durga";
39 System.out.println(s1==s2);
40
```

Line 35 and 36 are run time method call will create a new object in HEAP so the answer is FALSE

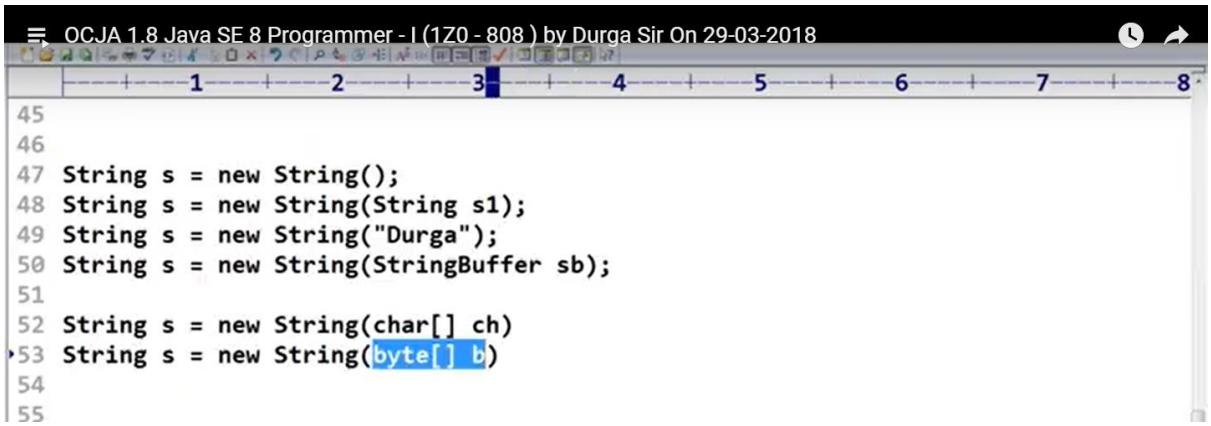


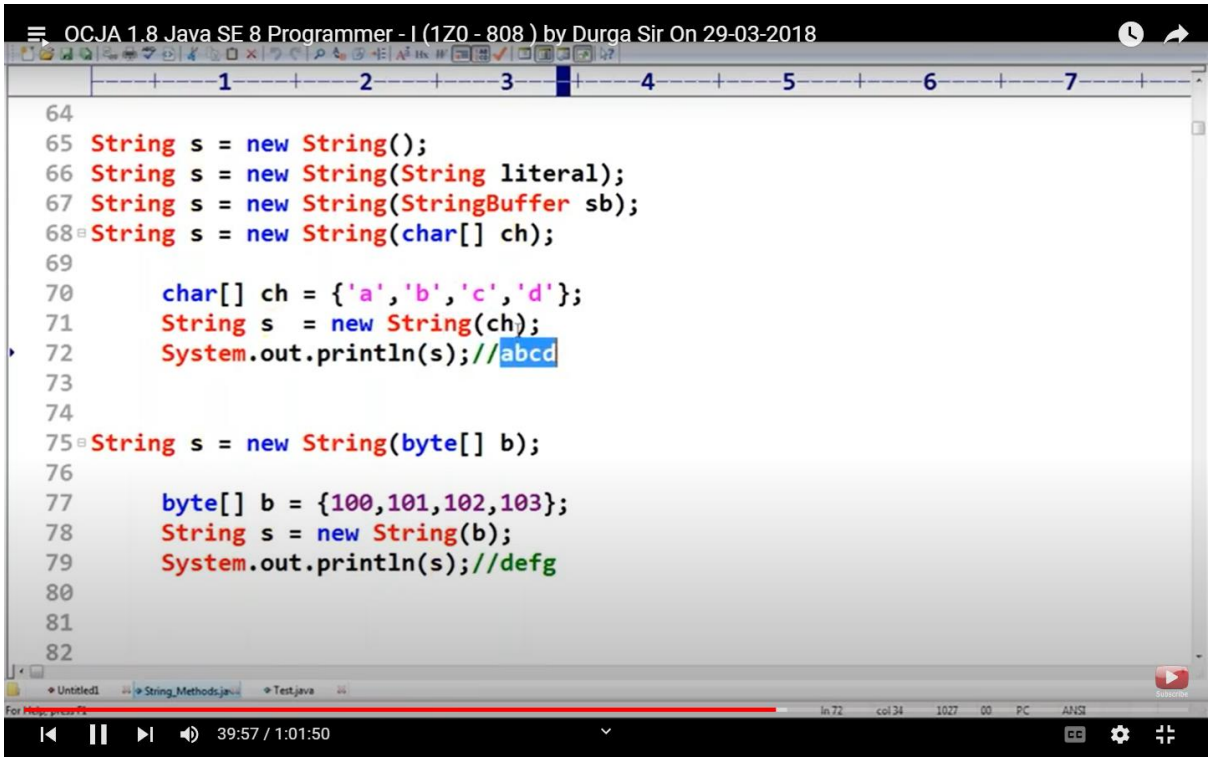
Unicode character of 100,101 -> d e f g

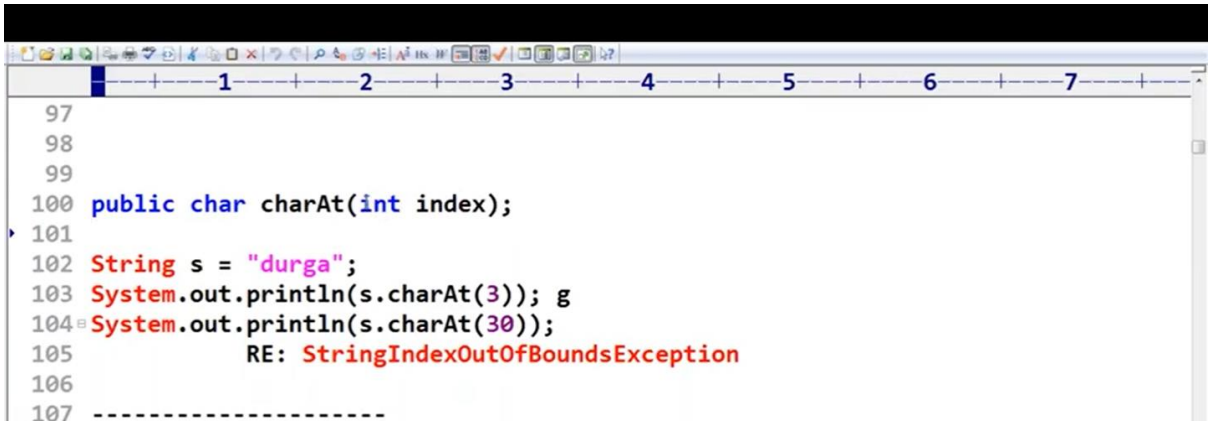
97- A 98-B 99 -C 100-D



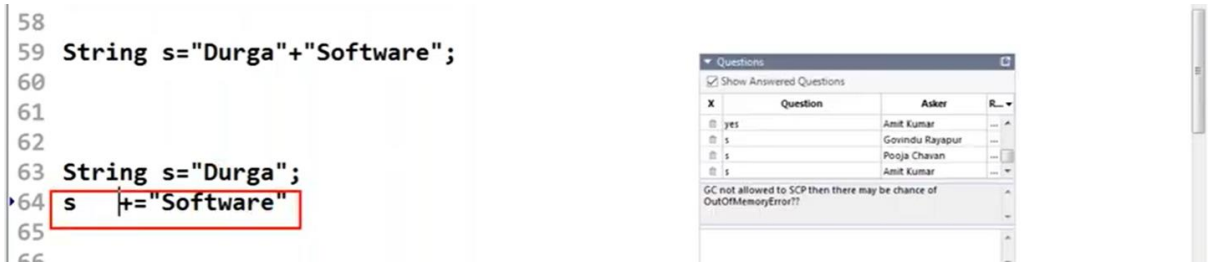
Create string using the below ways

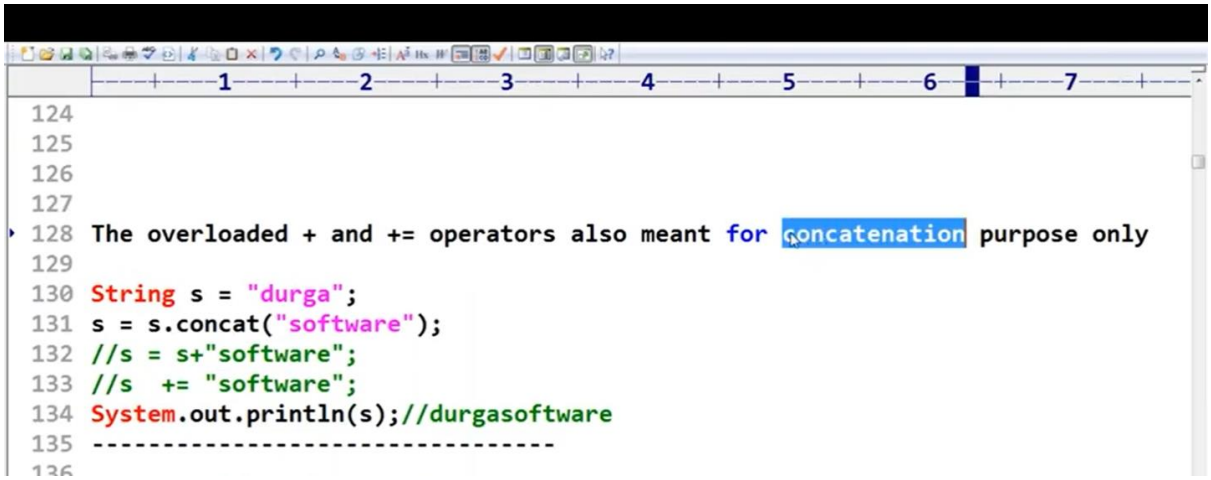






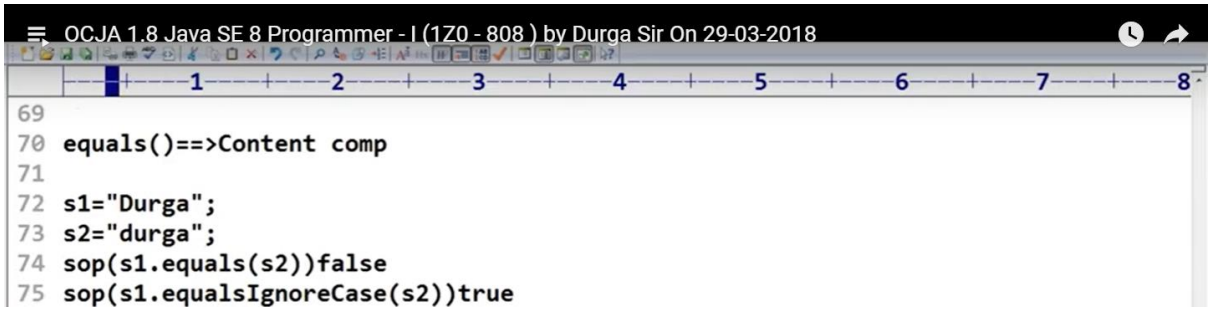
+ operator for concatenation.



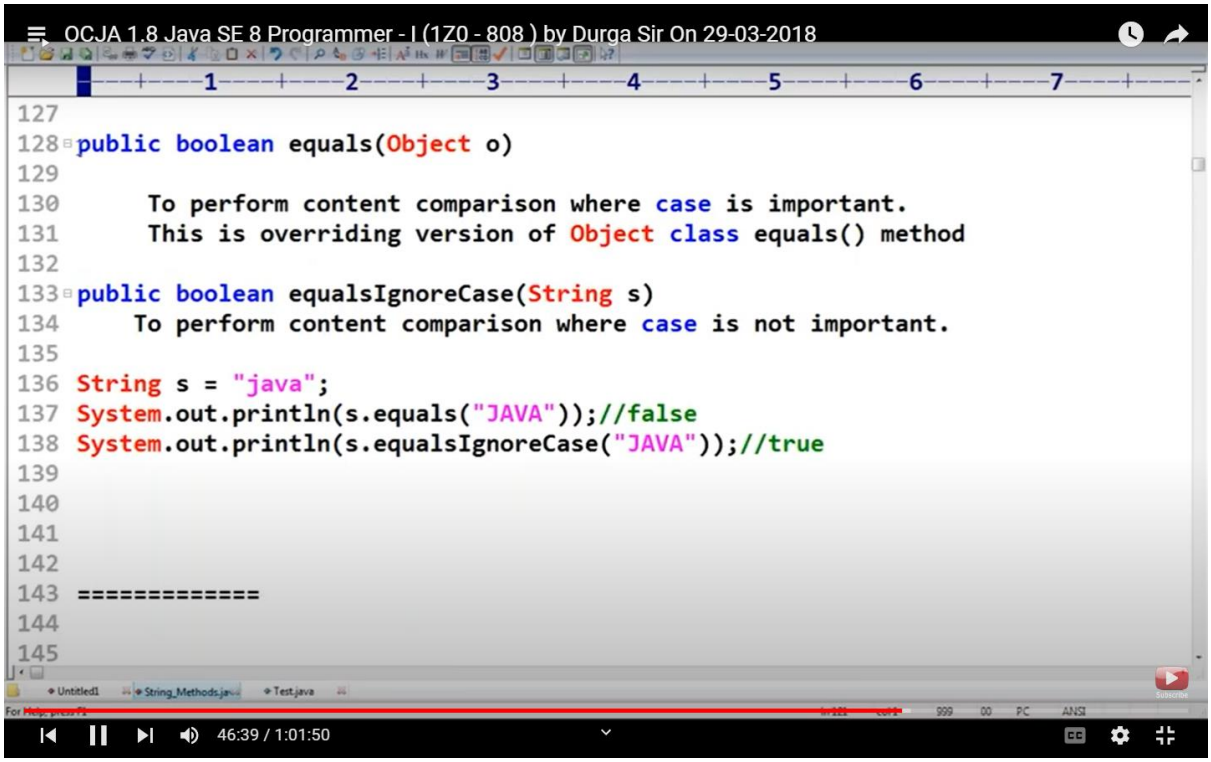


```
124
125
126
127
128 The overloaded + and += operators also meant for concatenation purpose only
129
130 String s = "durga";
131 s = s.concat("software");
132 //s = s+"software";
133 //s  += "software";
134 System.out.println(s);//durgasoftware
135 ---------------------------------
```

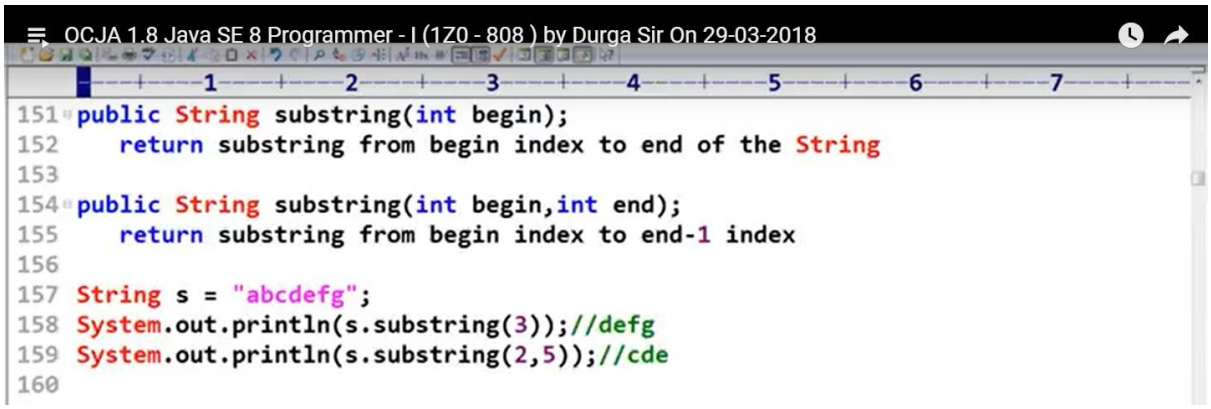
Equals Ignore Case



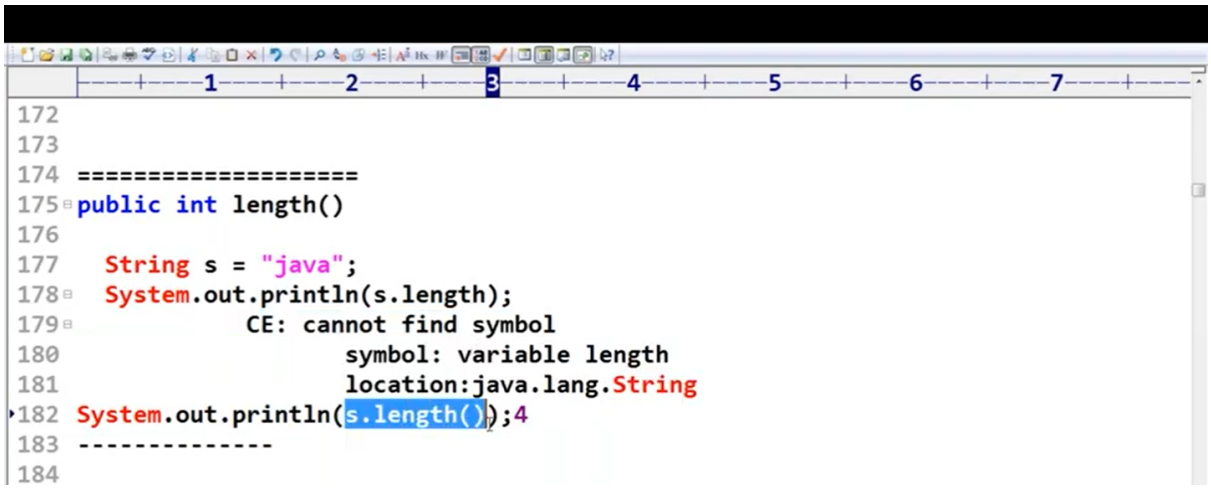

```
69
70 equals()==>Content comp
71
72 s1="Durga";
73 s2="durga";
74 sop(s1.equals(s2))false
75 sop(s1.equalsIgnoreCase(s2))true
```

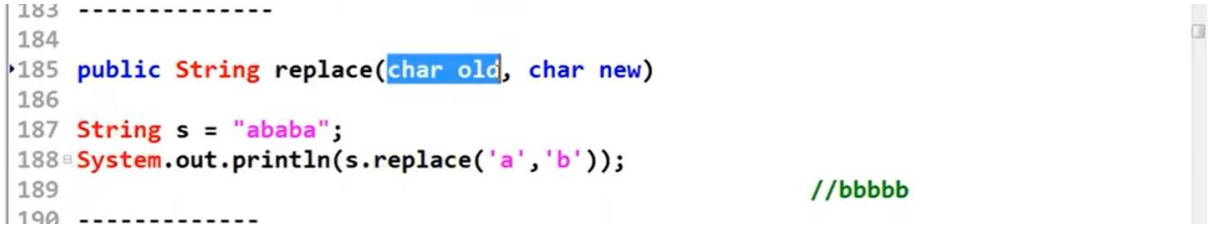



```
127
128 public boolean equals(Object o)
129
130      To perform content comparison where case is important.
131      This is overriding version of Object class equals() method
132
133 public boolean equalsIgnoreCase(String s)
134     To perform content comparison where case is not important.
135
136 String s = "java";
137 System.out.println(s.equals("JAVA"));//false
138 System.out.println(s.equalsIgnoreCase("JAVA"));//true
139
140
141
142
143 ==============
144
145
```



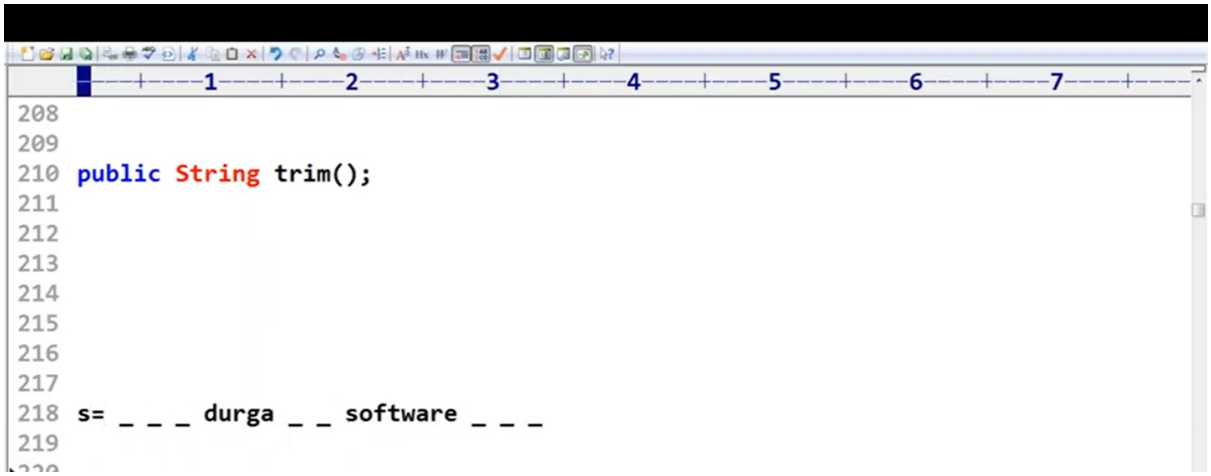
s.length – is a variable in array wheras in String we need to call length method



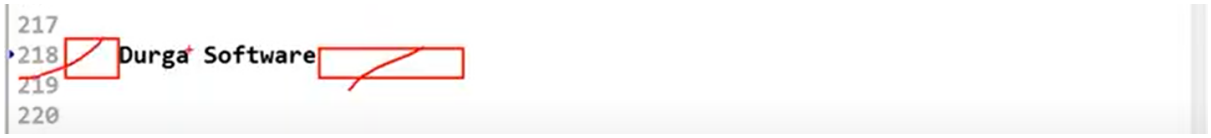


All white spaces will be removed. All beginning and trailing.

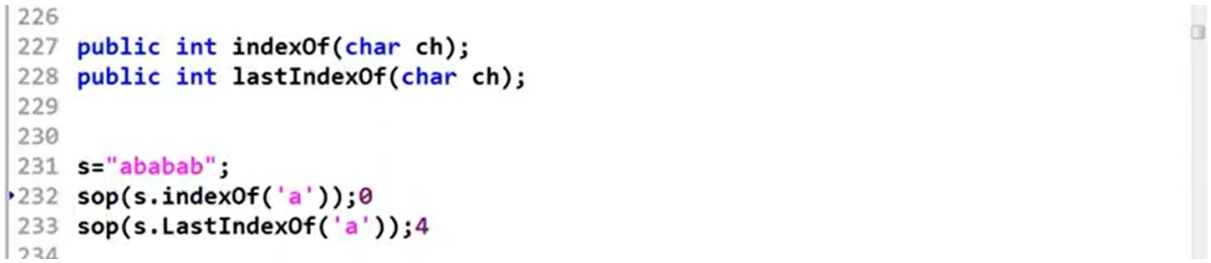
Here _ is blanks just for ref



```
208
209
210 public String trim();
211
212
213
214
215
216
217
218 s= _ _ _ durga _ _ software _ _ _
219
```



```
217
218 Durga Software
219
220
```

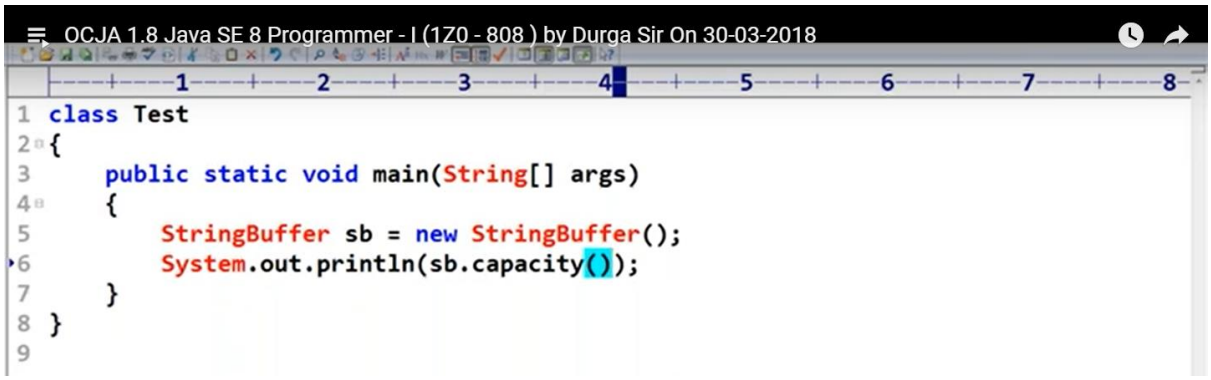


```
226
227 public int indexOf(char ch);
228 public int lastIndexOf(char ch);
229
230
231 s="ababab";
232 sop(s.indexOf('a'));0
233 sop(s.LastIndexOf('a'));4
234
```

If the String content keeps on changing then its gonna create a new object everytime you change it.

Then go with String Buffer or Builder. No new object will be created in SB.

String Buffer by default a capacity of 16 will be allocated . Length and Capacity are diff




```
1 class Test
2 {
3     public static void main(String[] args)
4     {
5         StringBuffer sb = new StringBuffer();
6         System.out.println(sb.capacity());
7     }
8 }
9
```

Which means u can add up to 16 chars

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
16
```

Add 16 character and check

```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          StringBuffer sb = new StringBuffer();
6          System.out.println(sb.capacity());
7          sb.append("abcdefghijklmnop");
8          System.out.println(sb.capacity());
9      }
10 }
```

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
16
```

If you add 17 character.. When it reaches the max capacity, a new StringBuffer object will be created with capacity = (CurrentCapacity+1)*2 -> 34

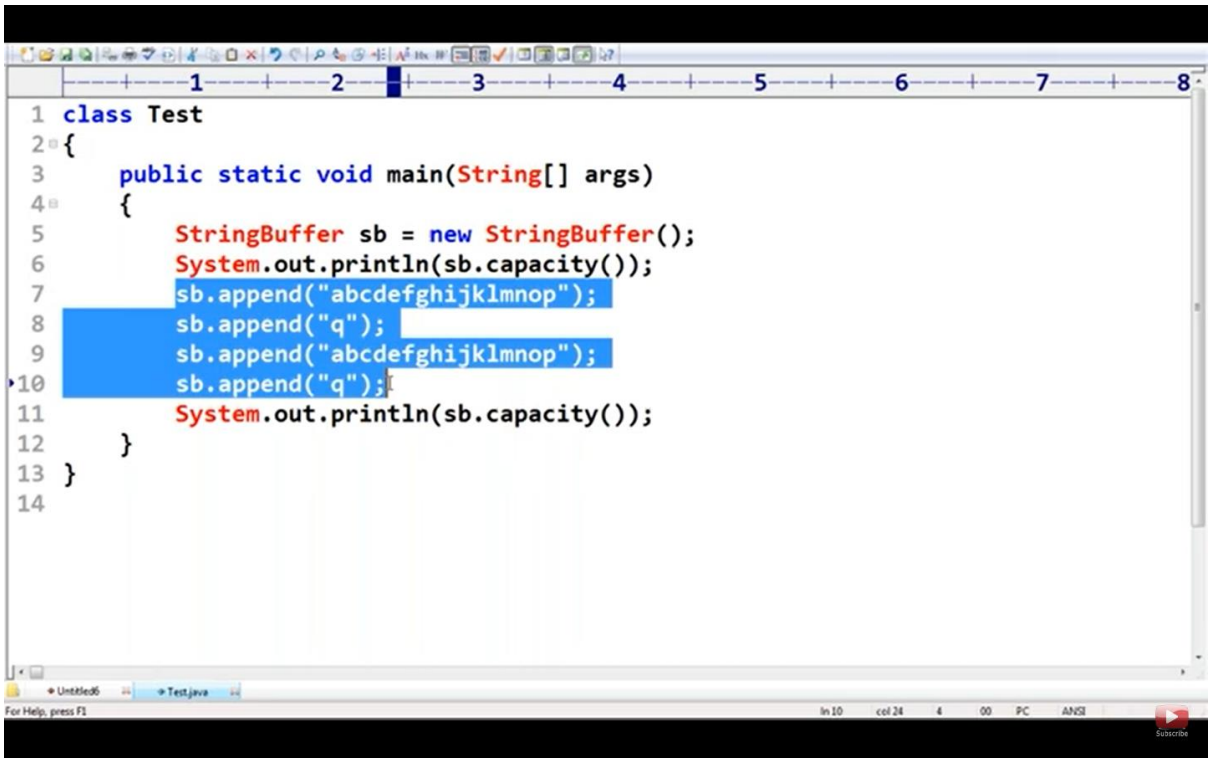
OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 30-03-2018

```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          StringBuffer sb = new StringBuffer();
6          System.out.println(sb.capacity());
7          sb.append("abcdefghijklmnop");
8          sb.append("q");
9          System.out.println(sb.capacity());
10     }
11 }
12
```

OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 30-03-2018

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
16
34
```

Add 34 chars-> no impact will be 34 capacity only



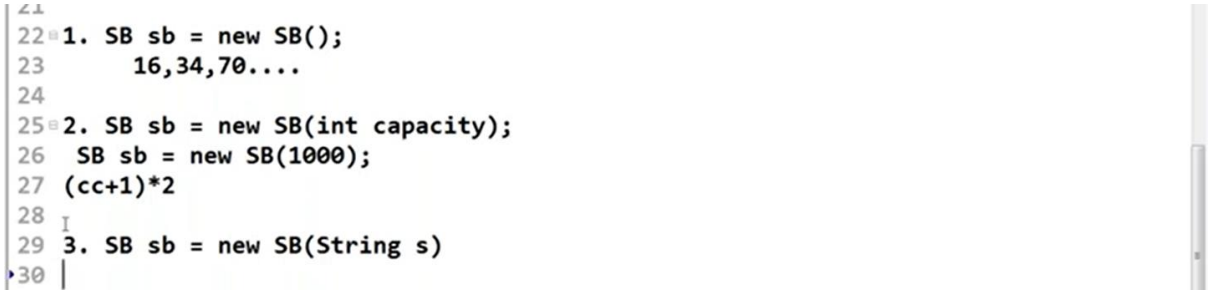
```
class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb = new StringBuffer();
        System.out.println(sb.capacity());
        sb.append("abcdefghijklmnop");
        sb.append("q");
        sb.append("abcdefghijklmnop");
        sb.append("q");
        System.out.println(sb.capacity());
    }
}
```

Add 35 char

34+1-> 35*2-> 70

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
16
70
```

Everytime new object will be created with new capacity and the ref var will be repointed and the old object will be eligible for garbage collection. Rather than doing this , can we specify the capacity



```
1. SB sb = new SB();
     16,34,70....

2. SB sb = new SB(int capacity);
 SB sb = new SB(1000);
(cc+1)*2

3. SB sb = new SB(String s)
```

21

If you pass a String , creates a capactity = 16+Length of the String -> 21

```
1 class Test
2 {
3     public static void main(String[] args)
4     {
5         StringBuffer sb = new StringBuffer("durga");
6         System.out.println(sb.capacity());
7     }
8 }
9
```

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
21
```

```
304
305
306 StringBuffer sb = new StringBuffer(String s);
307 capacity = s.length()+16
```

```
316
317
318
319 public int length();
320 public int capacity();
321 public char charAt(int index);
322
323 StringBuffer sb = new StringBuffer("durga");
324 System.out.println(sb.charAt(3));g
325 System.out.println(sb.charAt(30));
326     RE:StringIndexOutOfBoundsException
327
328
329
330 public void setCharAt(int index,char ch);
331    To replace the character locating at specified index with provided character
332
333
334
```

OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 30-03-2018

```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          StringBuffer sb = new StringBuffer("durga");
6          sb.setCharAt(2,'Z');
7          System.out.println(sb);
8      }
9  }
10
```

OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 30-03-2018

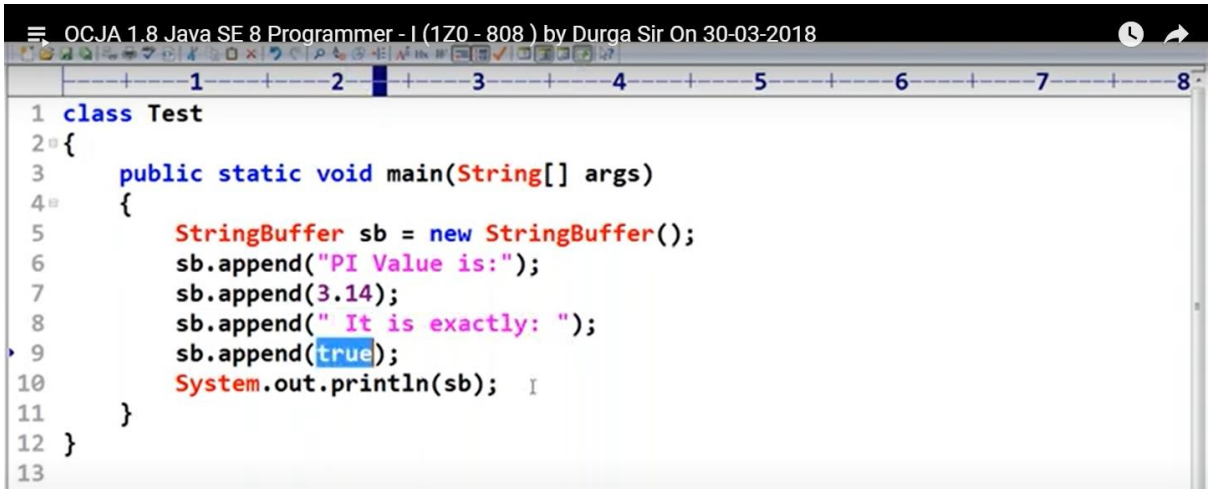
```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          StringBuffer sb = new StringBuffer("durga");
6          sb.append("software");
7          System.out.println(sb);
8      }
9  }
10
```

Append can take any arg

```
1  class Test
2  {
3      public static void main(String[] args)
4      {
5          StringBuffer sb = new StringBuffer("durga");
6          sb.append("software");
7          sb.append(10);
8          sb.append(10.5);
9          sb.append(true);
10         sb.append(20);
11         System.out.println(sb);
12     }
13 }
14
```

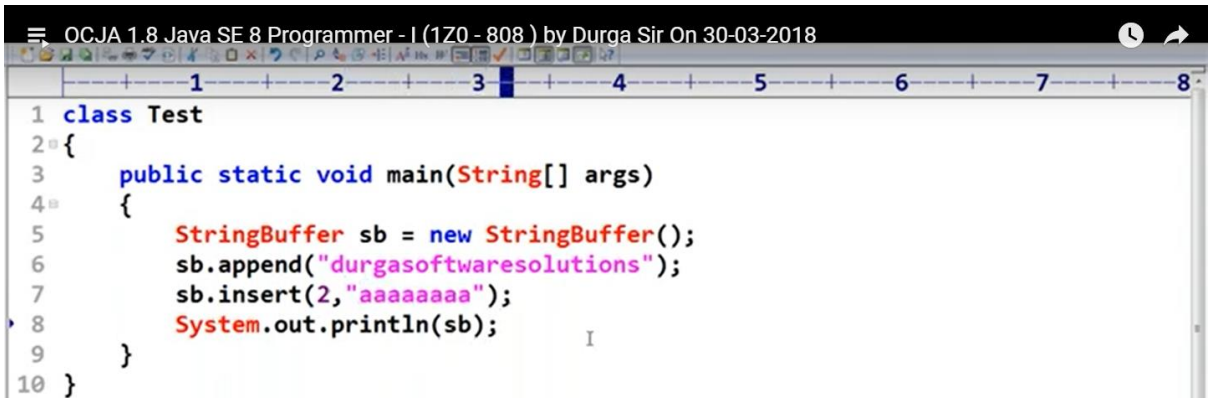
OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 30-03-2018

```
D:\durgaclasses>javac Test.java

D:\durgaclasses>java Test
durgasoftware1010.5true20
```




```
D:\durgaclasses>java Test
PI Value is:3.14 It is exactly: true

D:\durgaclasses>
```

Append adds at the end , insert add at the specified index and performs a shift

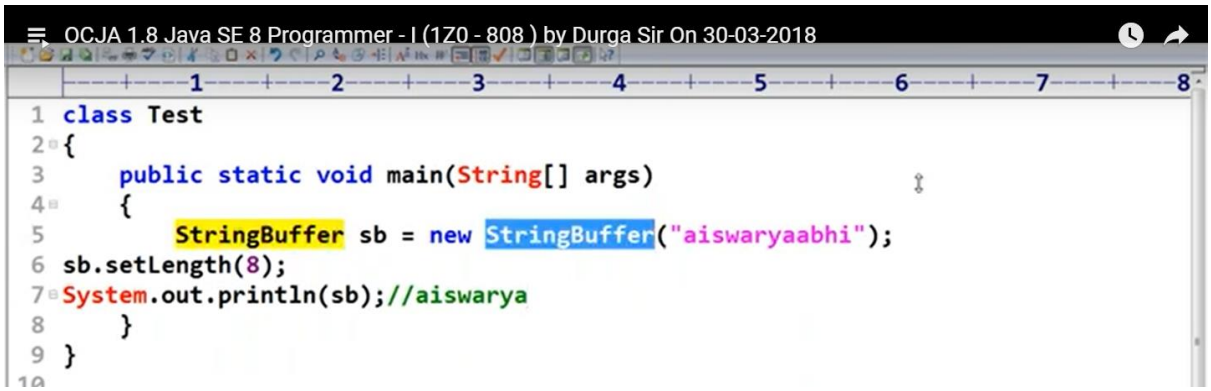



```
D:\durgaclasses>java Test
duaaaaaaargasoftwaresolutions
```

```
public StringBuffer delete(int begin,int end)
    To delete characters locating from begin index to end-1 index
public StringBuffer deleteCharAt(int index)
```

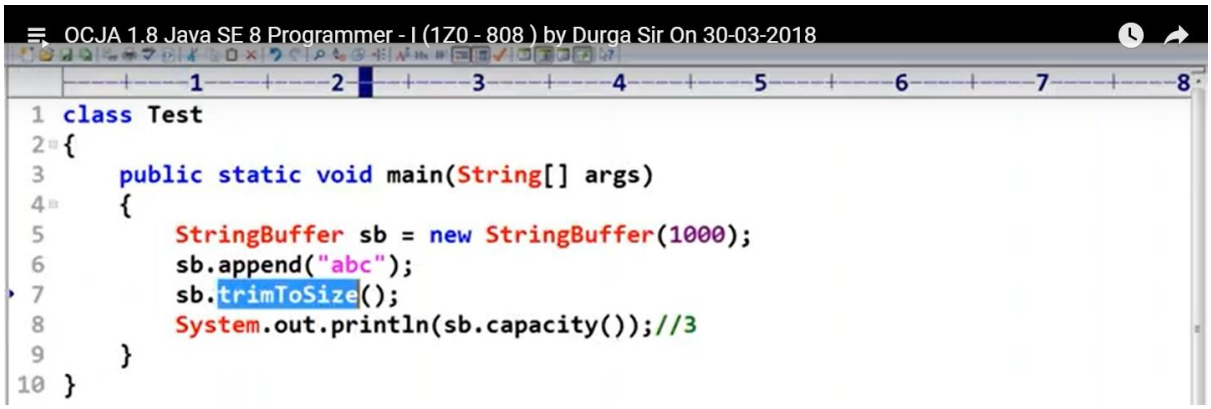
```
public StringBuffer reverse();
```

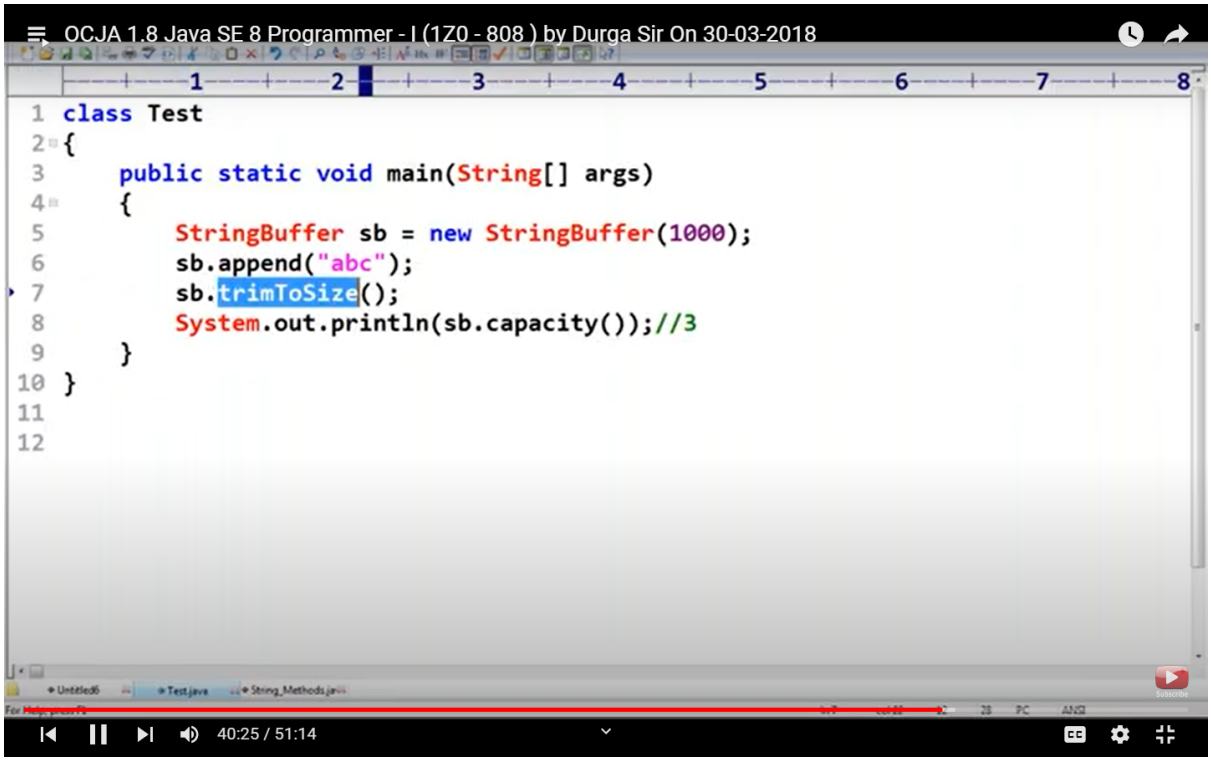
```
durgasoftwaresolutions

8 characters
sb.setLength(8)
```



To delete extra allocated free memory

```
SB sb = new SB(1000);
sb.append("abc");
sb.trimToSize();
```




OCJA 1.8 Java SE 8 Programmer - I (1Z0 - 808) by Durga Sir On 30-03-2018
class Test
{
public static void main(String[] args)
{
StringBuffer sb = new StringBuffer(1000);
sb.append("abc");
sb.trimToSize();
System.out.println(sb.capacity());//3
}
}
40:25 / 51:14