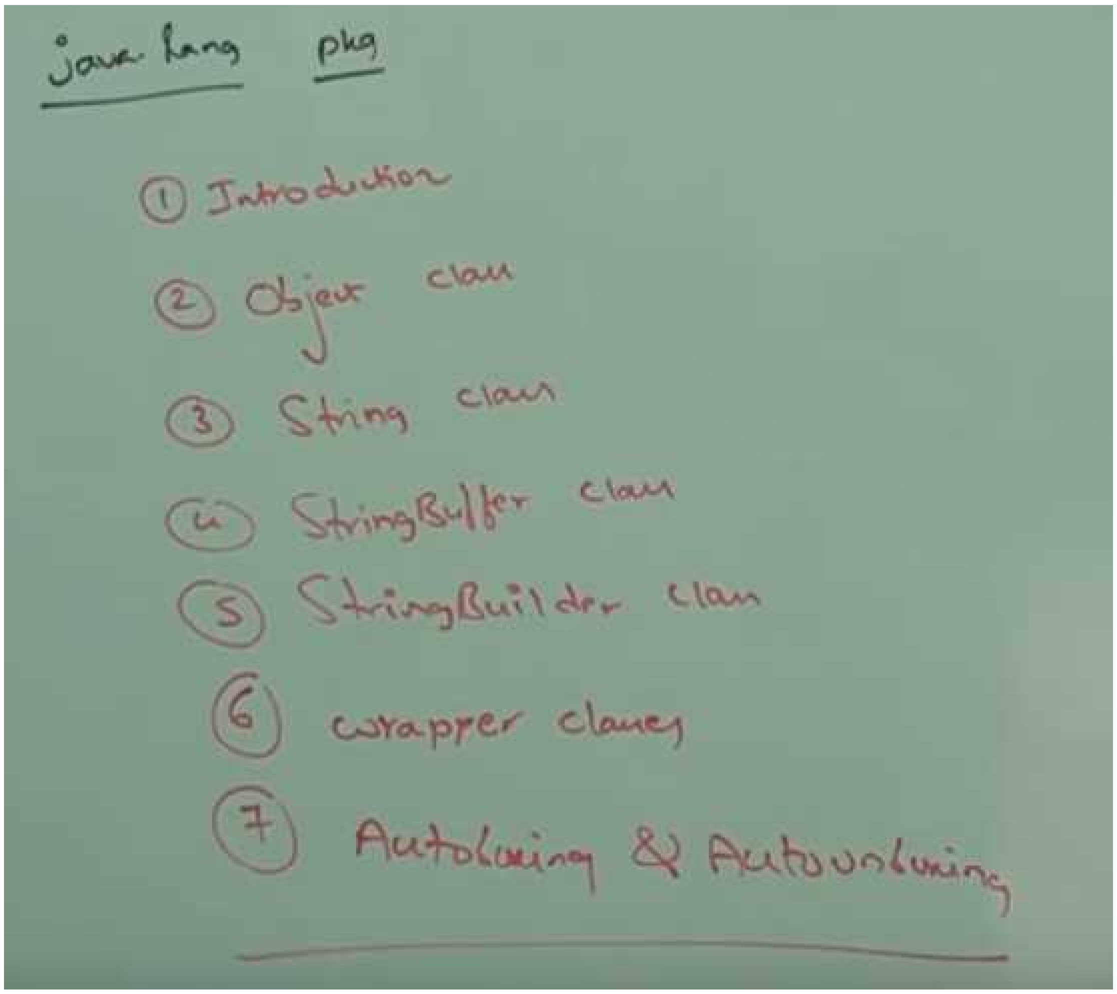
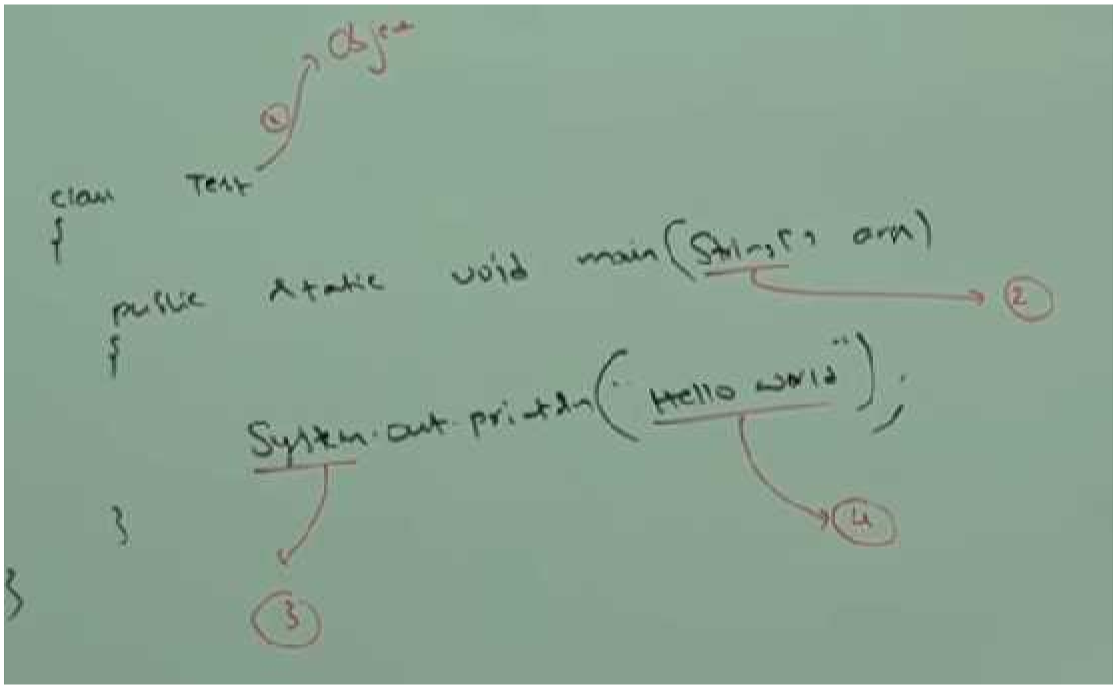


java.lang pkg

1. Introduction
2. Object class
3. String class
4. StringBuffer class
5. StringBuilder class
6. wrapper classes
7. Autoboxing & Autounboxing

For writting any java program whether it is simple or complex the most commonly classes and interfaces are grouped into a separate pkg which is java.lang pkg.



```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("Hello world");
    }
}
```

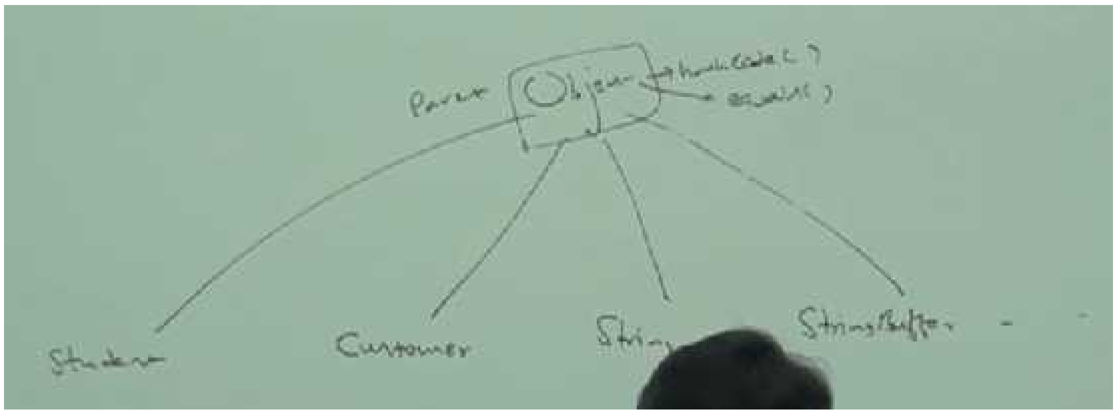
We are not required to import java.lang pkg implicitly bcz all classes and interfaces present in lang pkg bydefault availabe to every java program.

java.lang.object:

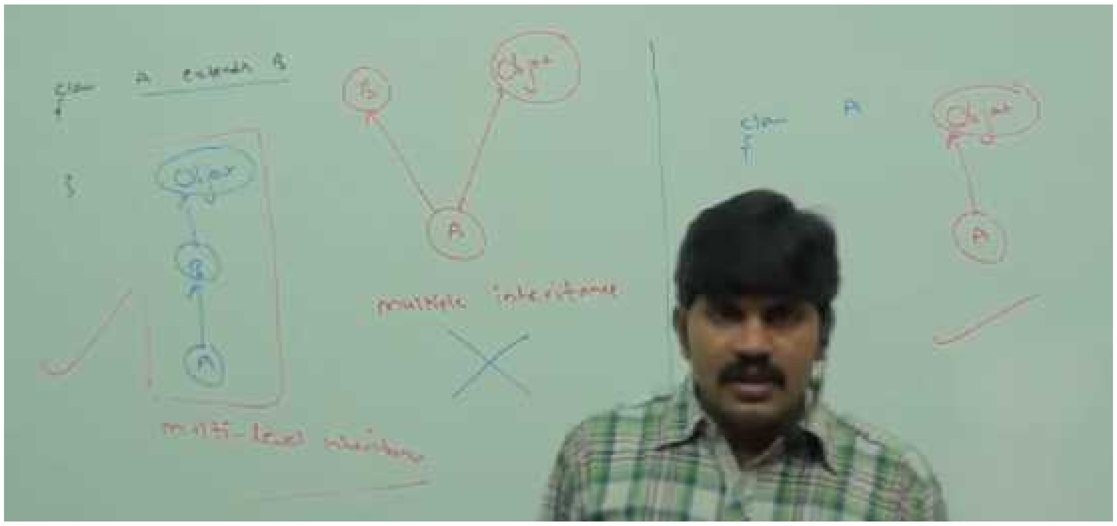
The most commonly required methods for every java class(whether it is predefined class or customized class) are defined in a separate class which is nothing but object class.

Every class in java is the child class of object either directly or indirectly so that object class method by default available to every java class.

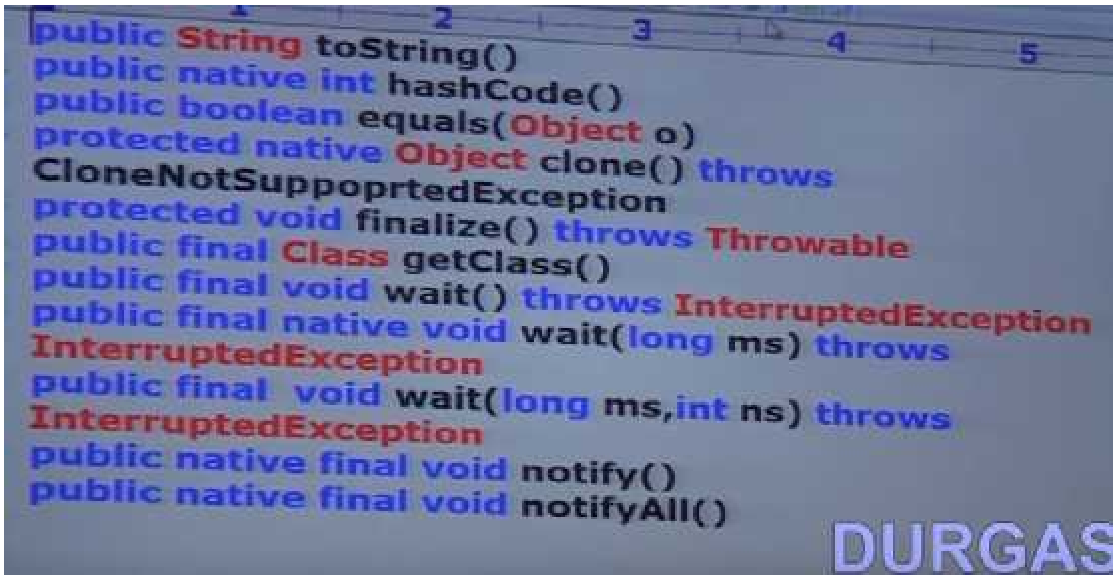
Hence Object class is considered as root of all java classes.



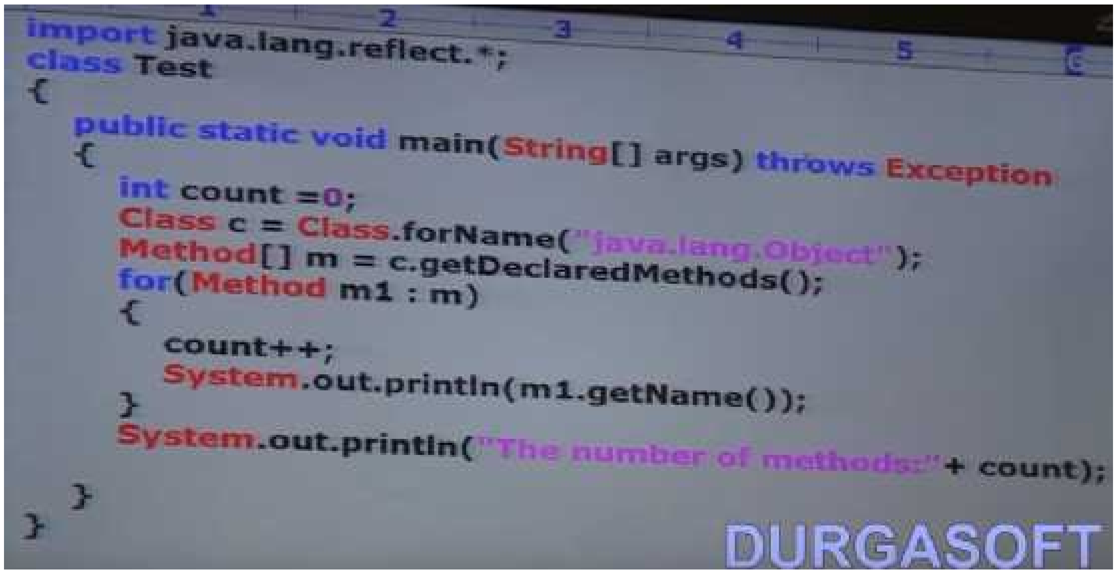
If our class does not extend any other class then only our class is the direct child class of object.



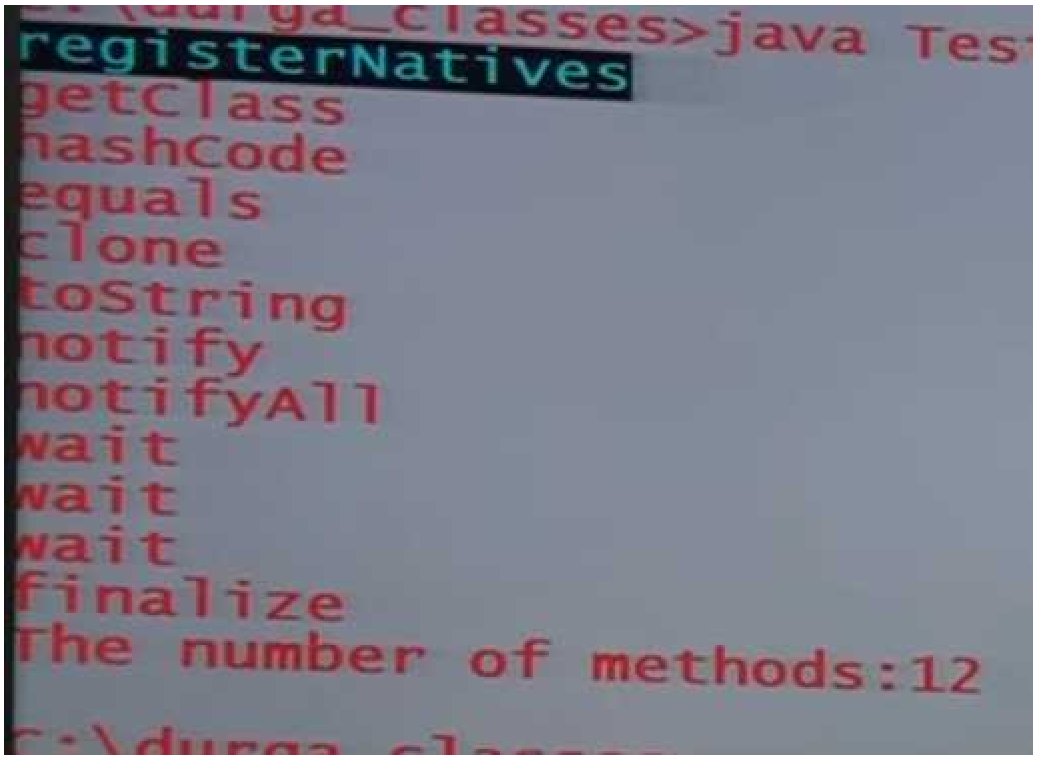
Object class defines the following 11 methods,



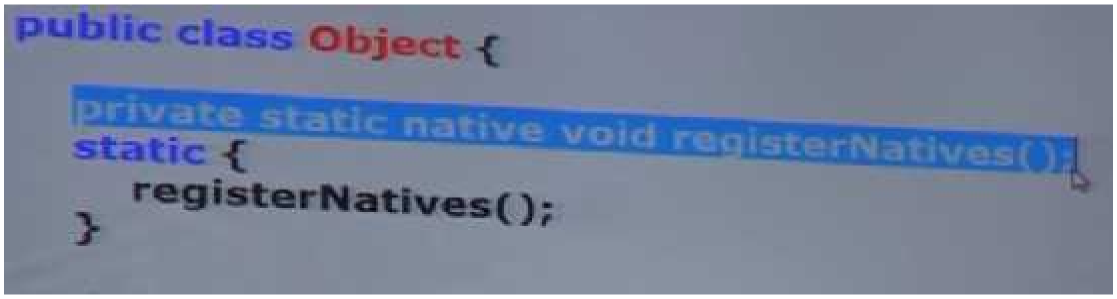
```
public String toString()
public native int hashCode()
public boolean equals(Object o)
protected native Object clone() throws
CloneNotSuppoprtedException
protected void finalize() throws Throwable
public final Class getClass()
public final void wait() throws InterruptedException
public final native void wait(long ms) throws
InterruptedException
public final  void wait(long ms,int ns) throws
InterruptedException
public native final void notify()
public native final void notifyAll()
```



```
import java.lang.reflect.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        int count =0;
        Class c = Class.forName("java.lang.Object");
        Method[] m = c.getDeclaredMethods();
        for(Method m1 : m)
        {
            count++;
            System.out.println(m1.getName());
        }
        System.out.println("The number of methods:"+ count);
    }
}
```



```
c:\durga_classes>java Tes
registerNatives
getClass
hashCode
equals
clone
toString
notify
notifyAll
wait
wait
wait
finalize
The number of methods:12

c:\durga_classes>
```



```
public class Object {
    private static native void registerNatives();
    static {
        registerNatives();
    }
```

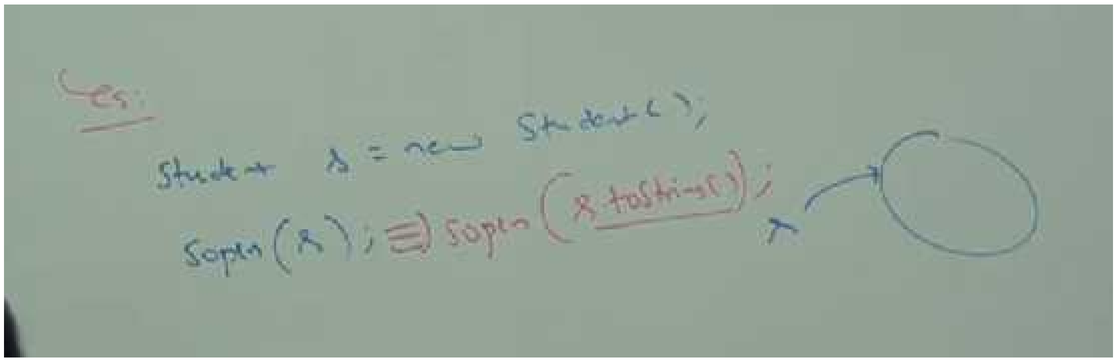
Strictly speaking object class contains 12 methods. The extra method is registerNatives().

toString()

We can use toString(), method to get string representation of an object.

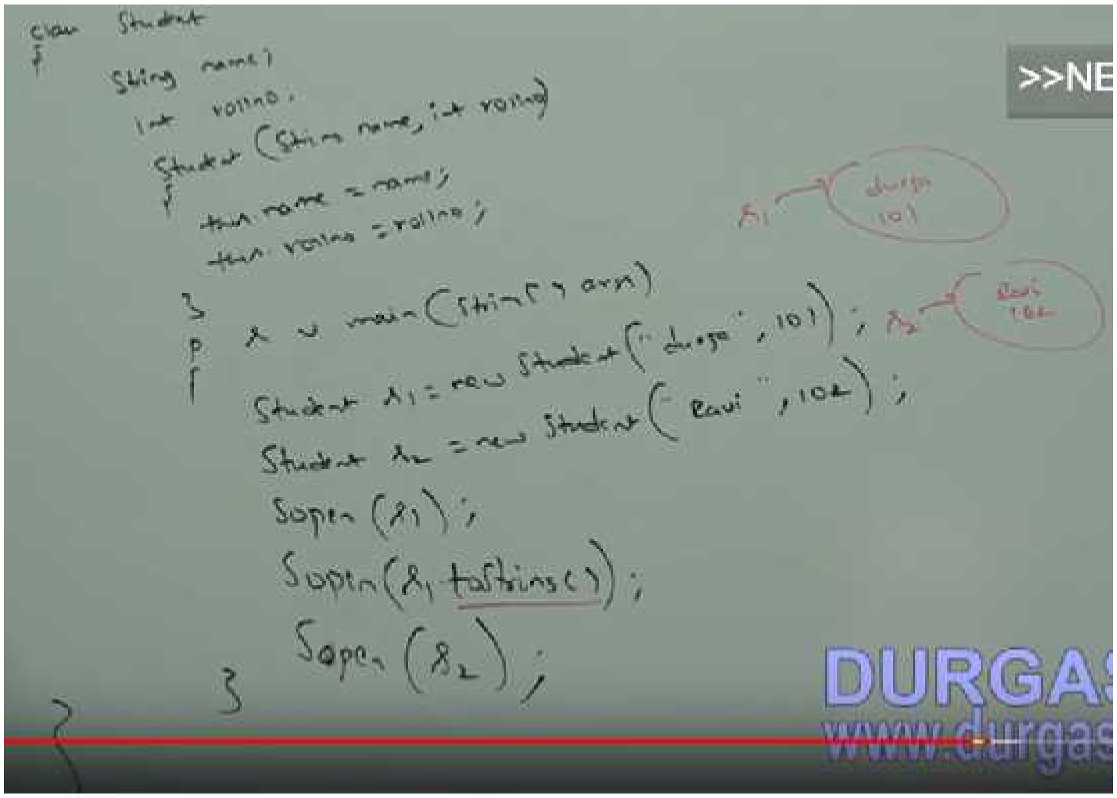
```
String s = (obj).toString();
```

Whenever we are trying to print object referrence internally toString(), method will be called.



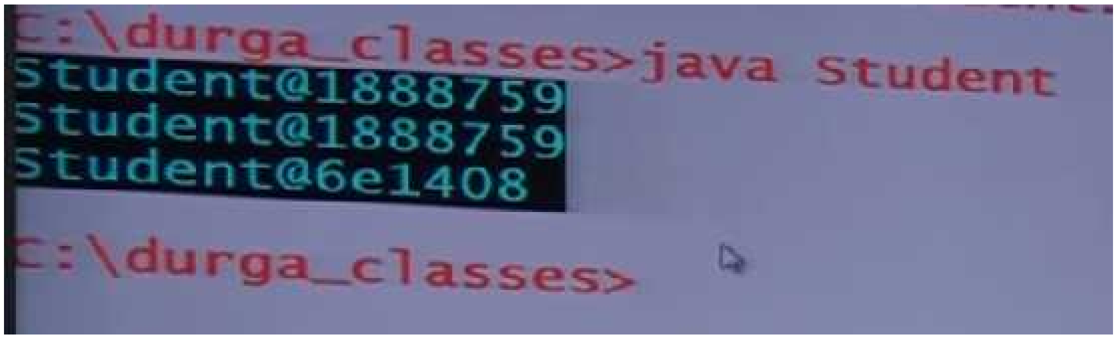
```
ex:
    Student s = new Student();
    sopln(s); ⇒ sopln(s.toString());
```

If our class does not contain toSTring(), method then object class toString(),, method will be executed.



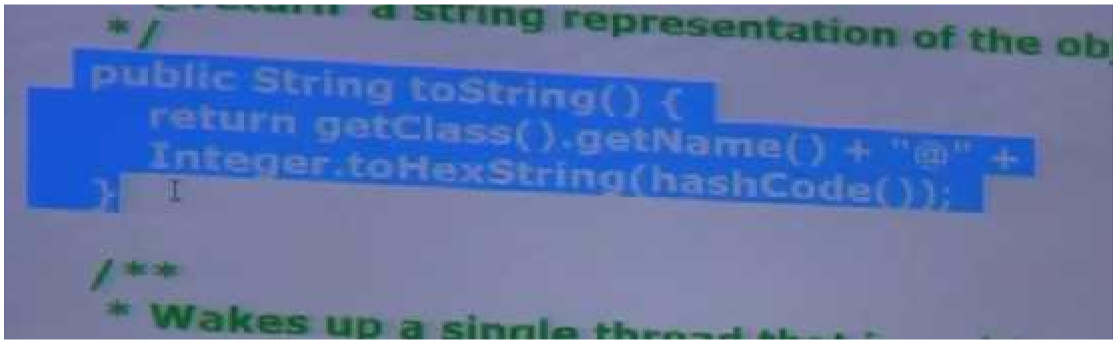
```
class Student
{
    String name;
    int rollno;
    Student(String name, int rollno)
    {
        this.name = name;
        this.rollno = rollno;
    }
    p s v main(String[] args)
    {
        Student s1 = new Student("durga", 101);
        Student s2 = new Student("ravi", 102);
        Sopln(s1);
        Sopln(s1.toString());
        Sopln(s2);
    }
}
```

s1 → durga 101
s2 → ravi 102

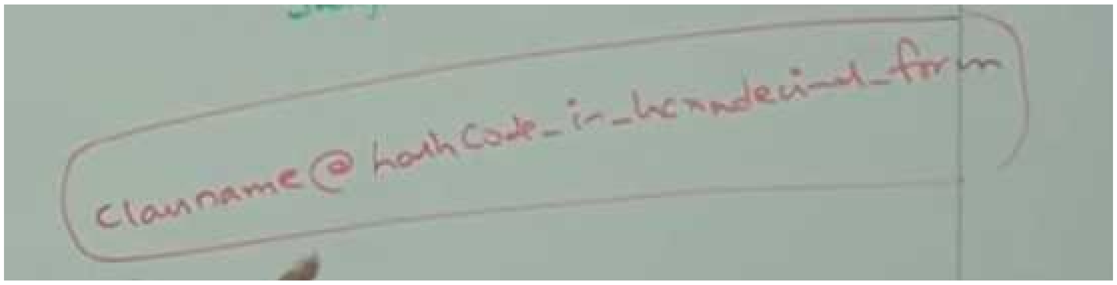


```
C:\durga_classes>java Student
Student@1888759
Student@1888759
Student@6e1408

C:\durga_classes>
```

In the above example Object class toString(), got executed which is implemented as follows,



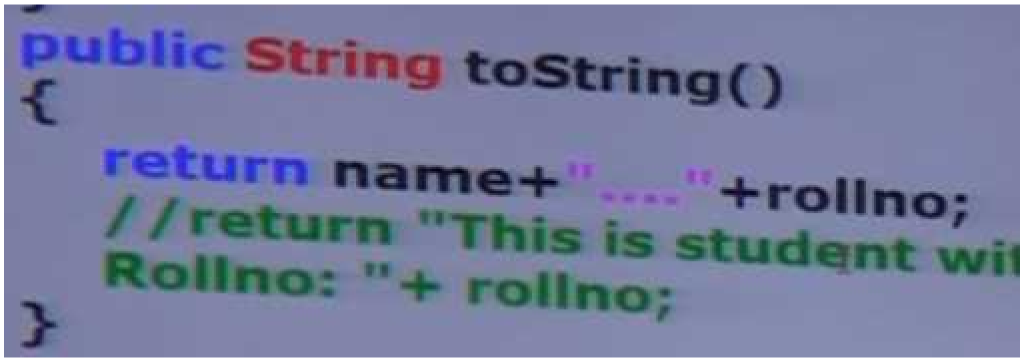
```
 */
public String toString() {
    return getClass().getName() + "@" +
    Integer.toHexString(hashCode());
}

/**
 * Wakes up a single thread
```



Classname@hashCode_in_hexadecimal_form

Based on our requirement we can override toString(), method to provide our own string representation.

Whever we are trying to print student object reference to print his name andd rollno, we have to override toString() method.



```
public String toString()
{
    return name+"....."+rollno;
    //return "This is student wi
    Rollno: "+ rollno;
}
```

hashCode()

For every object a unique number is generated by JVM, which is nothing but hashcode.

HashCode won't represent address of object.

JVM will use hashcode while saving objects into hashing related data structures like HashTable, HashMap, HashSet etc..

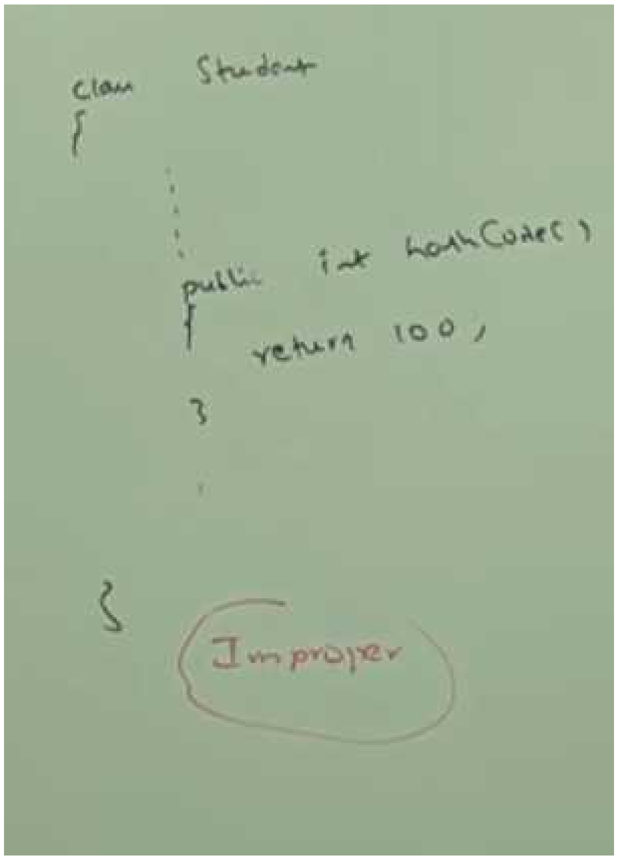
The main advantadge of saving objects based on hashcode is search operation will become easy.(The powerful search algorithem upto today is Hashing.)

If we are giving the chance to Object class hashCode method it will generate hashCode based on address of the object. It doesn't mean

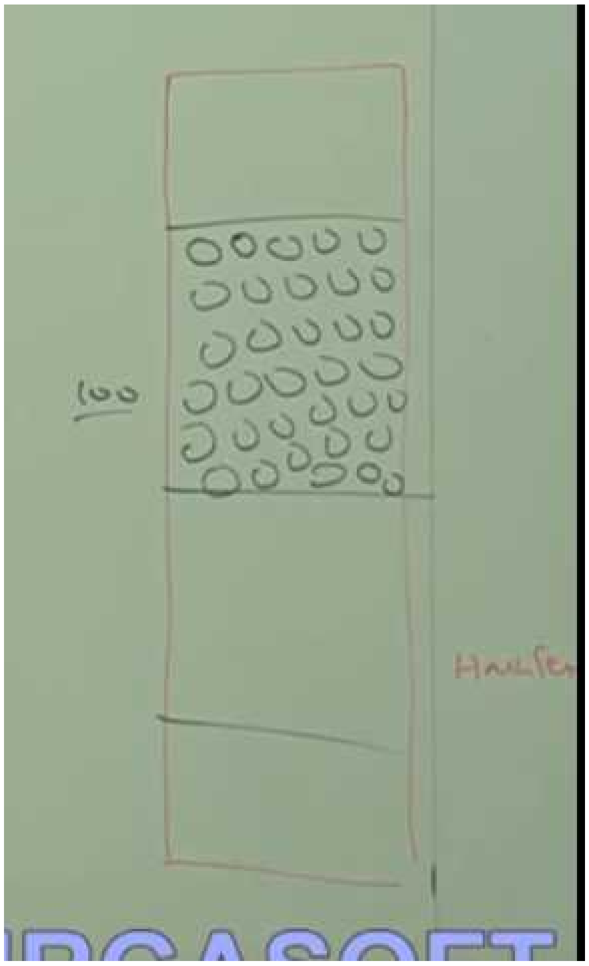
hashCode represents address.

Based on our requirement we can override hashcode method in our class to genetates our own hashCode.

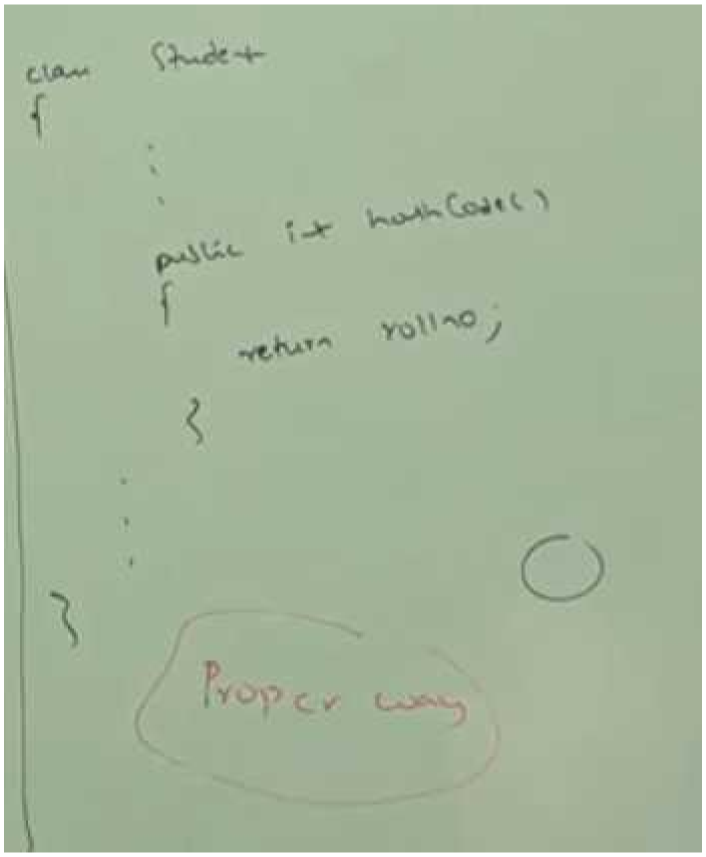
Overriding hashCode method is said to be proper if and only if for every object we have to generate a unique number as hashcode.



```
class Student
{
      :
      :
      public int hashCode()
      {
         return 100;
      }
      :
}
```



This is improper way of overriding hashcode method because for all student objects we are generating same number as hashcode.

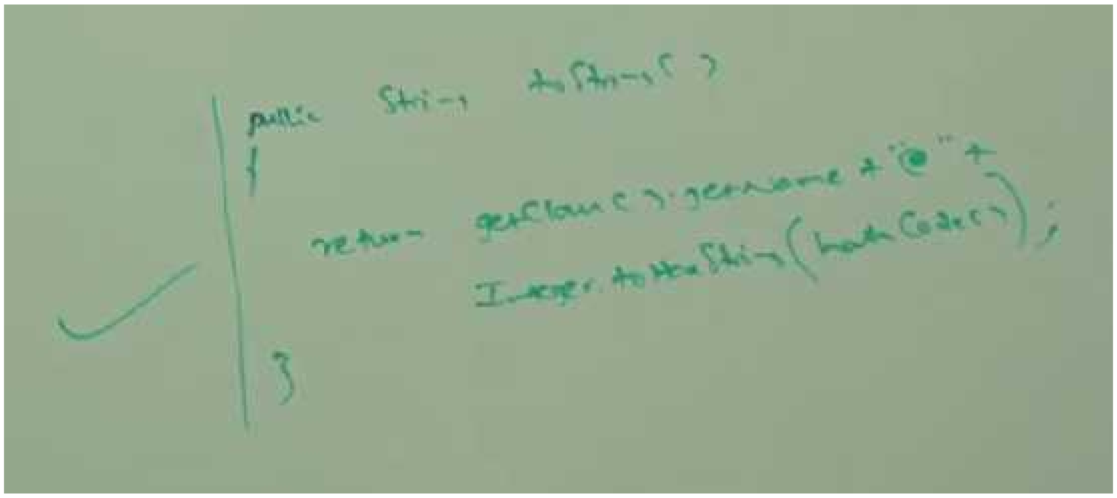


```
class Student
{
    :
    public int hashCode()
    {
        return rollno;
    }
    :
}
```

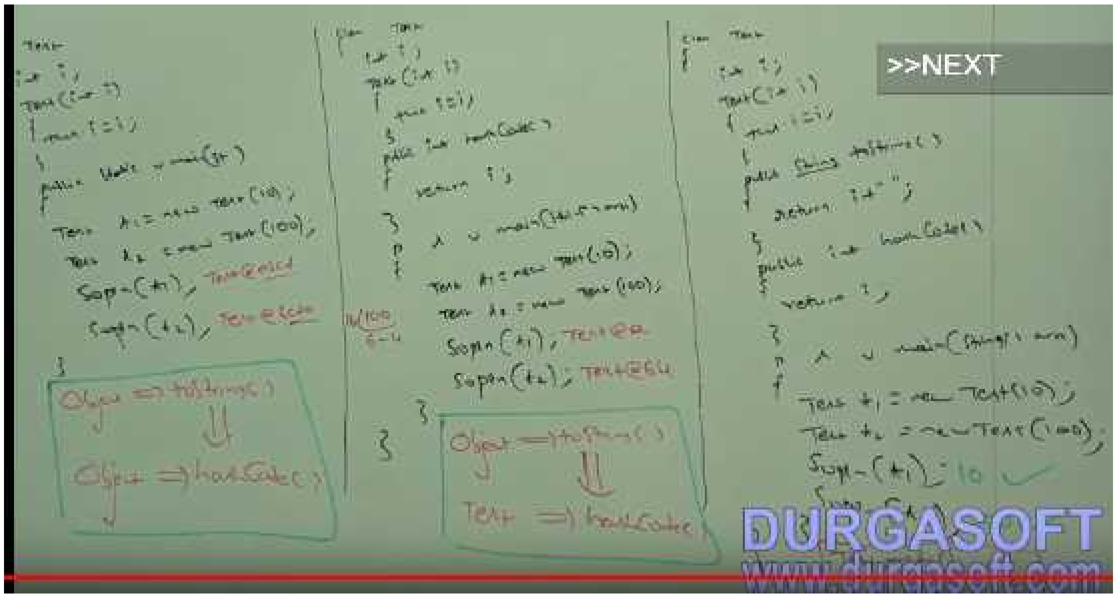
Proper way

This is proper way of overriding hashcode method because We are generating a different hashcode for every object.

Object Class toString(), Method

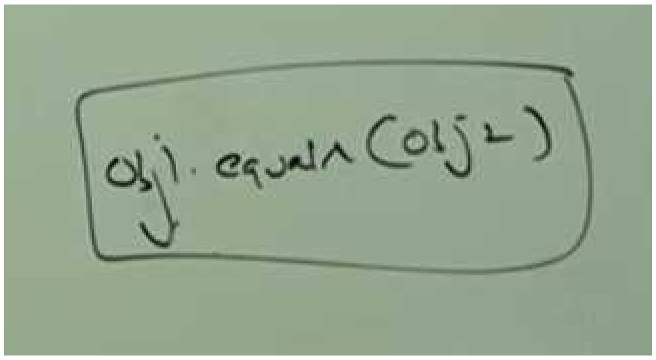


```
public String toString()
{
    return getClass().getName + "@" +
           Integer.toHexString(hashCode());
}
```

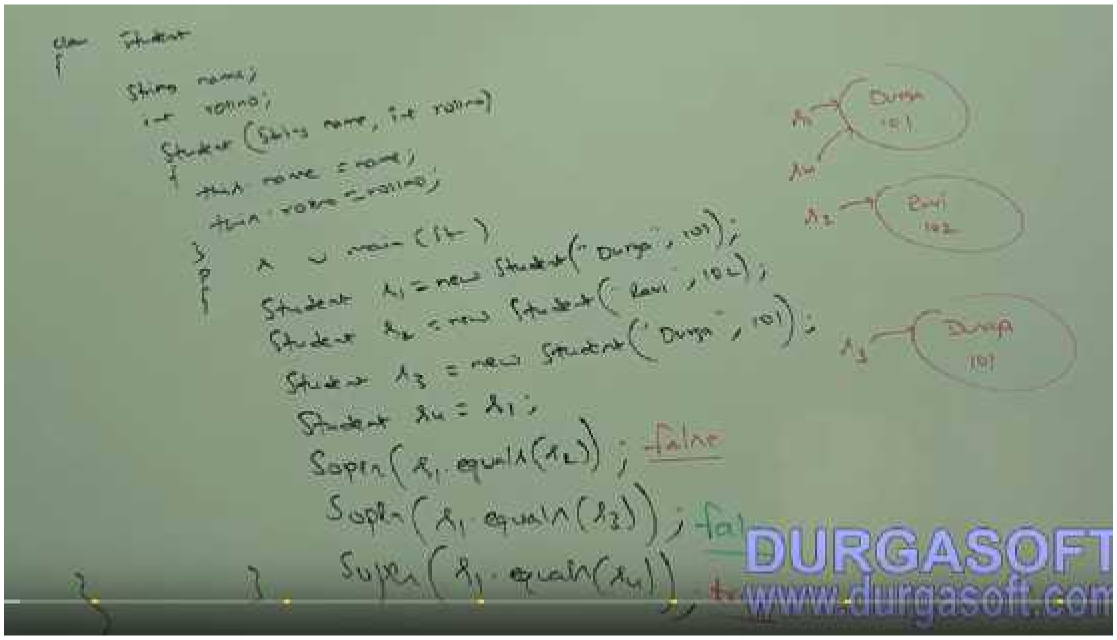


equals():

We can equals method to check equality of two objects.



If our class does not contain equals method then object class equals method will be executed.



```
class Student
{
    String name;
    int rollno;
    Student(String name, int rollno)
    {
        this.name = name;
        this.rollno = rollno;
    }
    p s v main(St )
    {
        Student s1 = new Student("Durga", 101);
        Student s2 = new Student("Ravi", 102);
        Student s3 = new Student("Durga", 101);
        Student s4 = s1;
        Sopln(s1.equals(s2)); false
        Sopln(s1.equals(s3)); fa
        Sopln(s1.equals(s4)); t
    }
}
```

If two references points to the same object then only .equals(), method returns true.(Object class .equals method)

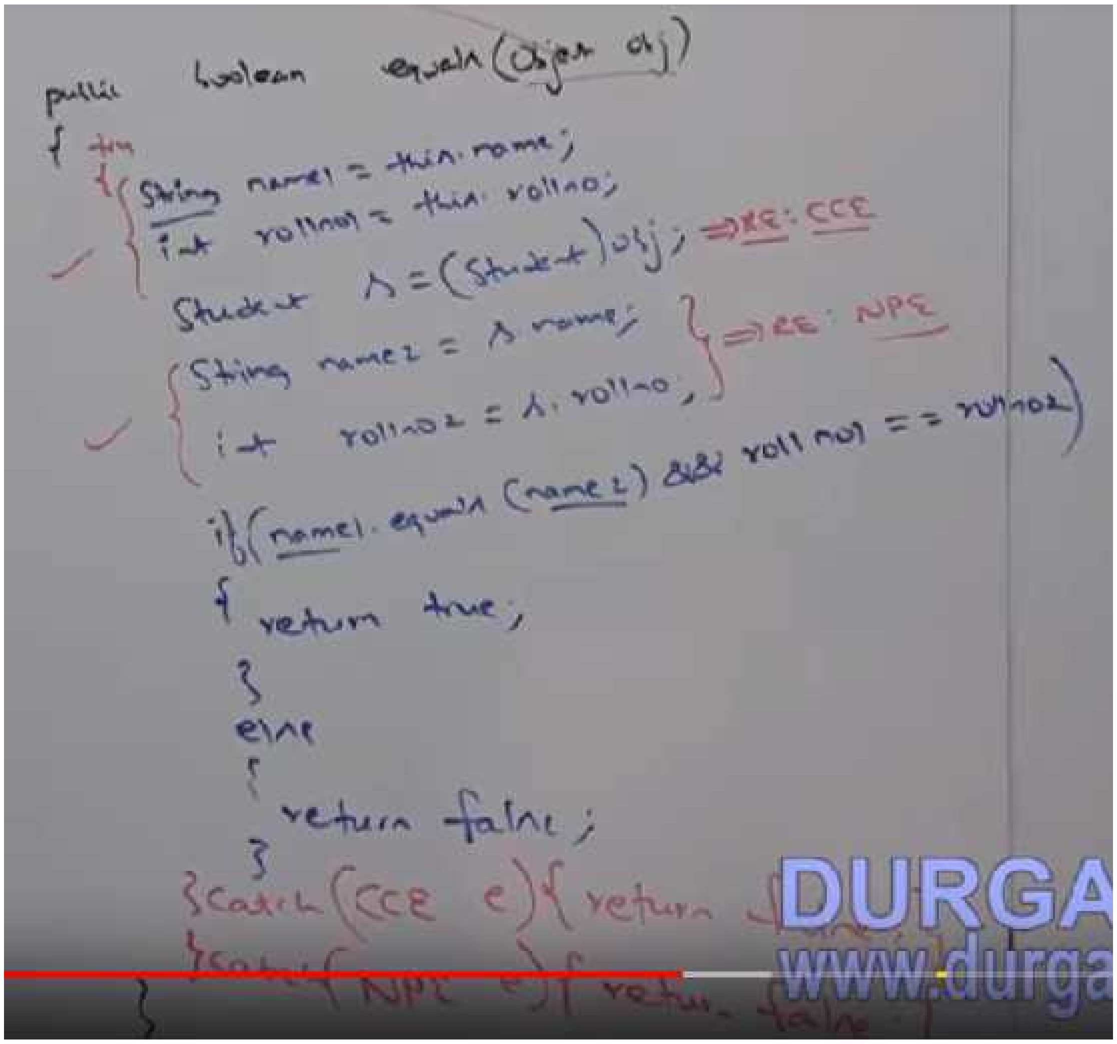
While overriding equals method for content comparision we have to take care about the following,

1. What is the meaning of equality, i.e. whether we have to check only names or only rollnos or both.

2. If we are passing different type of object our equals method should not raise class cast exception i.e. we have to handle class cast exception to return false.

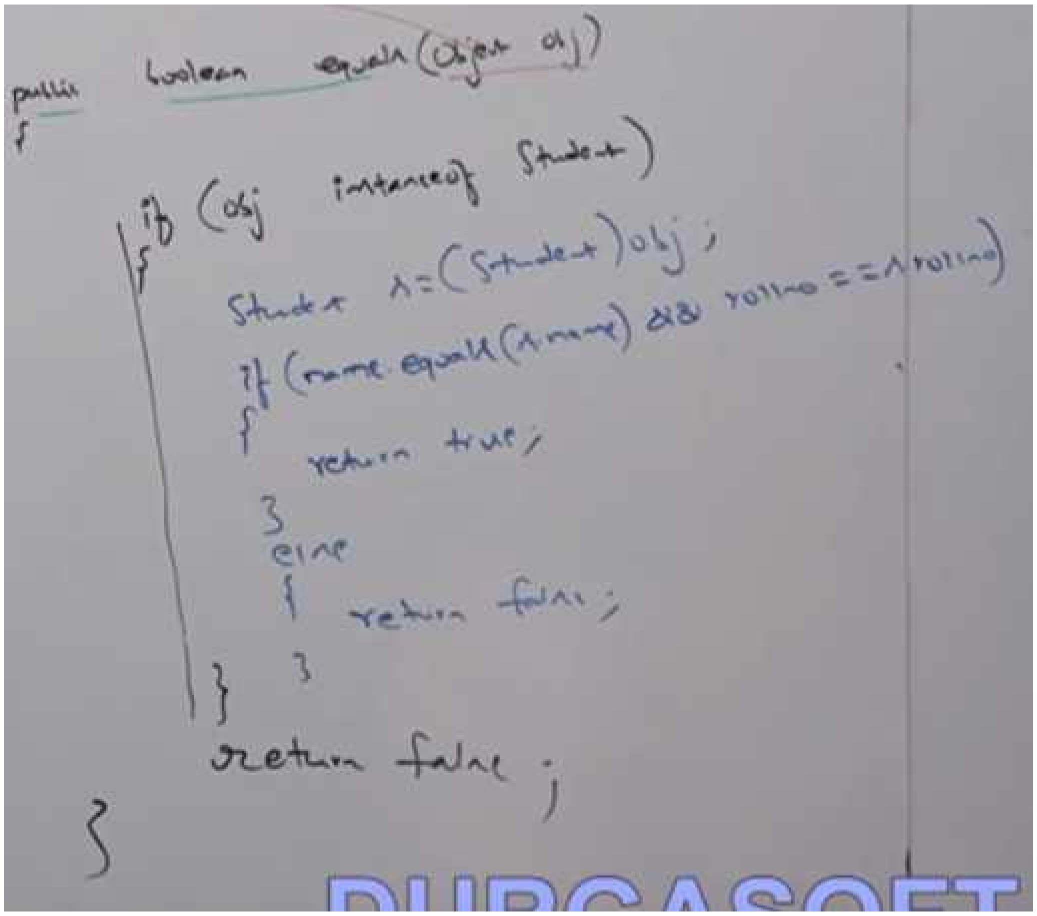
3. If we are passing null argument then our equals method should not raise null pointer exception. i.e. we have to handle null pointer exception to return false.

The following is the proper way of overriding equals method for student class content comparision.



```
public boolean equals(Object obj)
{ try
  { String name1 = this.name;
    int rollno1 = this.rollno;
    Student s = (Student)obj; ⇒RE: CCE
    String name2 = s.name;
    int rollno2 = s.rollno;   } ⇒RE: NPE
    if(name1.equals(name2) && rollno1 == rollno2)
    { return true;
    }
    else
    {
      return false;
    }
  }catch(CCE e){ return false;
  }catch(NPE e){ return false; }
}
```

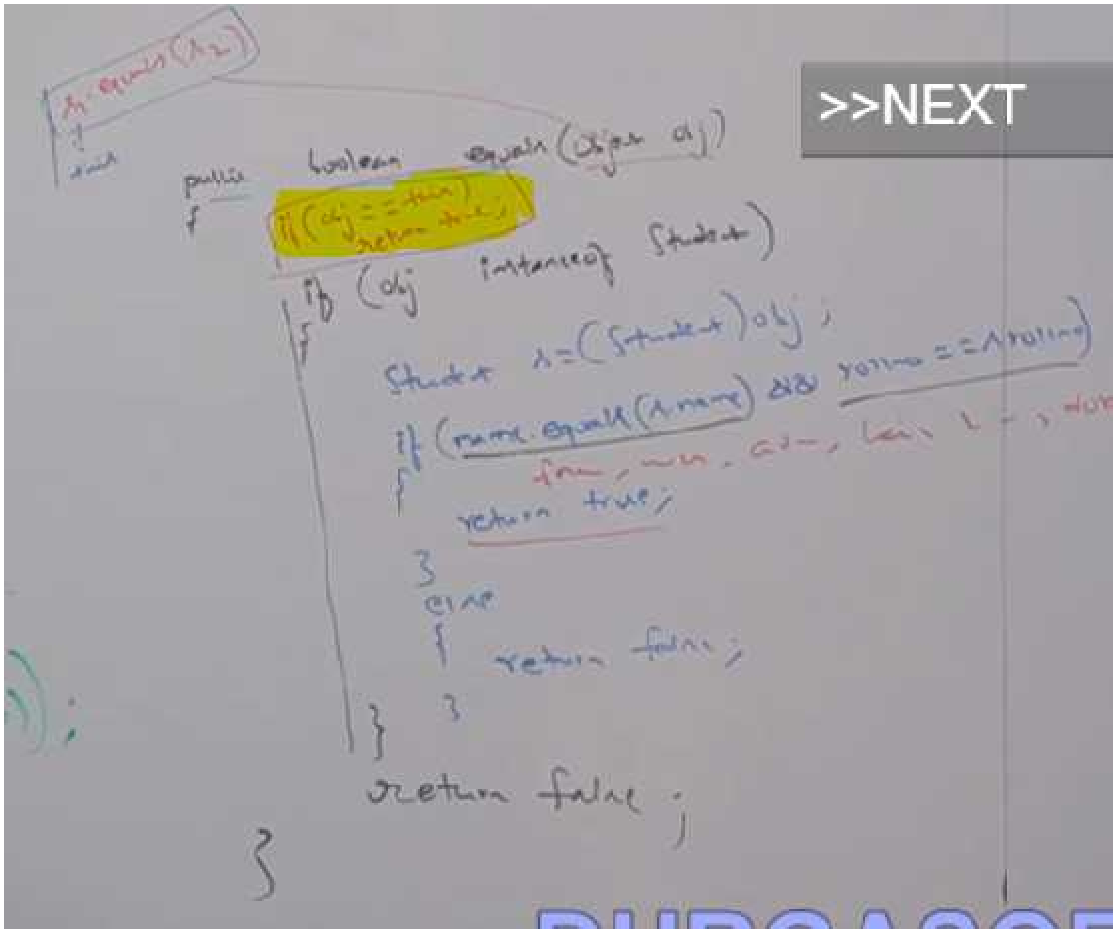
Simplified version:



```
public boolean equals(Object obj)
{
    if (obj instanceof Student)
    {
        Student s=(Student)obj;
        if(name.equals(s.name) && rollno==s.rollno)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}
```

Note:

To make above equals methods more efficient we have to write the following code at the beginning inside equals method.

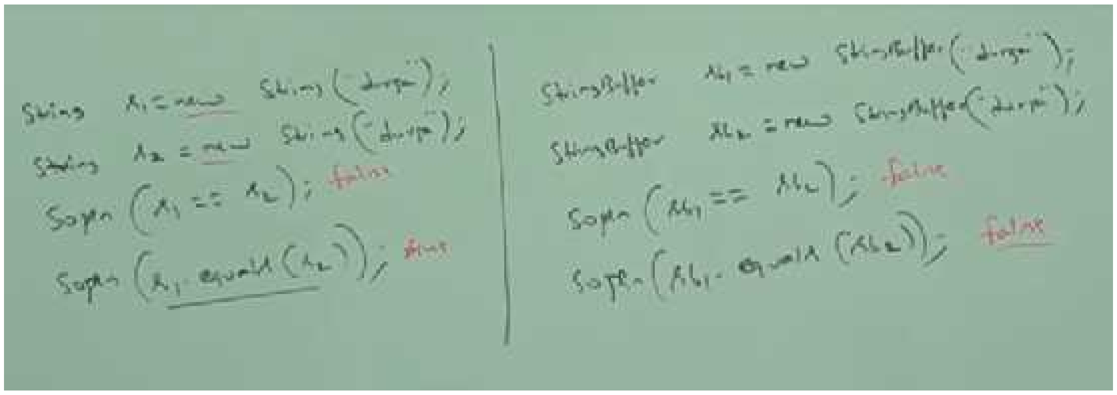


```
s1.equals(s2)

public boolean equals(Object obj)
{
    if(obj==this)
      return true;
    if(obj instanceof Student)
    {
        Student s=(Student)obj;
        if(name.equals(s.name) && rollno==s.rollno)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}
```

According to this if both references pointing to the same object then without performing any comparision .equals(), method returns true directly.

In String class .equals(), method is overriden for content comparision hence, even though objects are different if content is same then .equals(), method returns true.

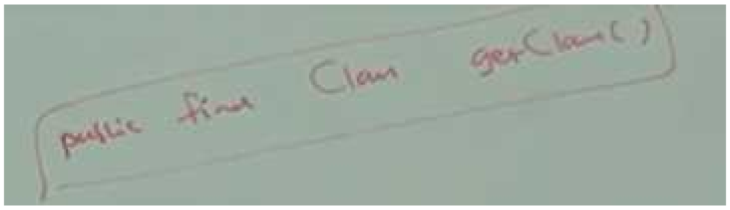
In StringBuffer, .equals(), method is not overriden for content comaprsion, hence if objects are different .equals() method returns false even though content is same.



```
String s1 = new String("durga");
String s2 = new String("durga");
Sopln(s1 == s2); false
Sopln(s1.equals(s2)); true

StringBuffer sb1 = new StringBuffer("durga");
StringBuffer sb2 = new StringBuffer("durga");
Sopln(sb1 == sb2); false
Sopln(sb1.equals(sb2)); false
```

getClass(), Method:

We can use getClass(), method to get runtime class definitaion of an object.

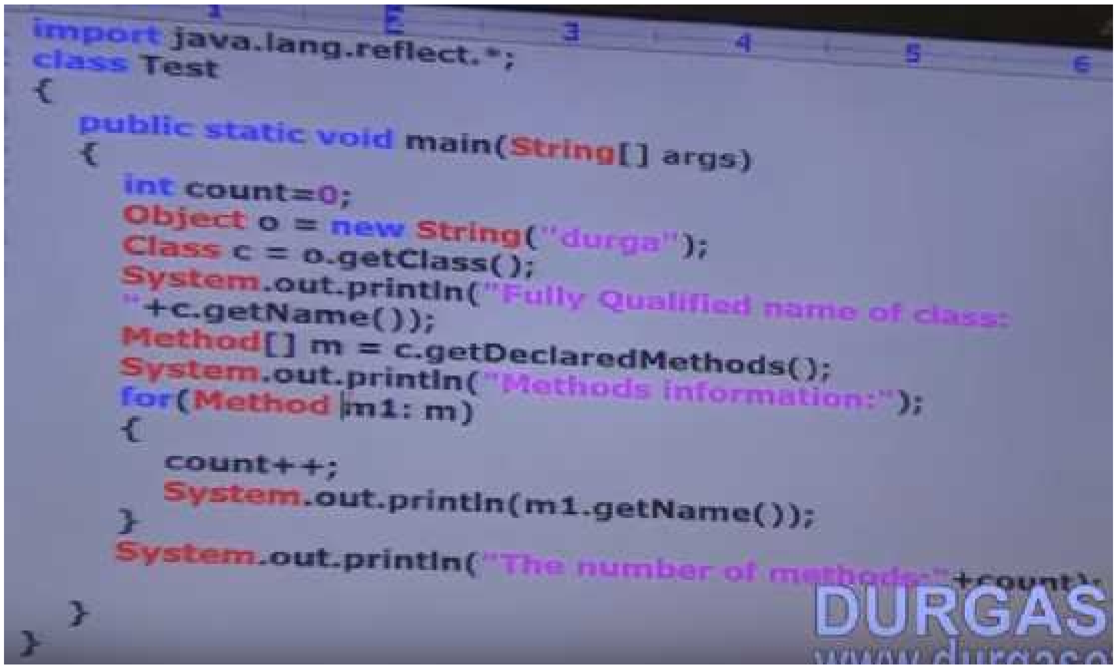


```
public final Class getClass()
```

By using this class Class object we can access class level properties like fully qualified name of the class

methods information

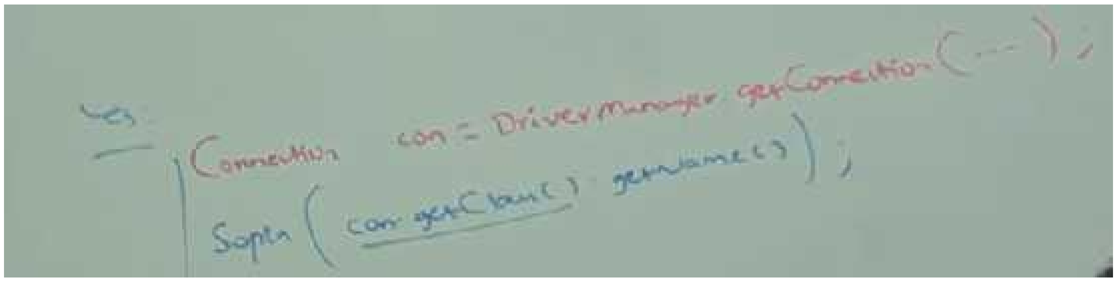
constructors information etc..



```
import java.lang.reflect.*;
class Test
{
  public static void main(String[] args)
  {
    int count=0;
    Object o = new String("durga");
    Class c = o.getClass();
    System.out.println("Fully Qualified name of class:
"+c.getName());
    Method[] m = c.getDeclaredMethods();
    System.out.println("Methods information:");
    for(Method m1: m)
    {
      count++;
      System.out.println(m1.getName());
    }
    System.out.println("The number of methods:"+count);
  }
}
```

E.g. 2

To display database vendor specific connection interface implemeted class name



```
eg:
Connection con = DriverManager.getConnection(....);
Sopln(con.getClass().getName());
```

Note:

1. After loading every .class file JVM will create an object of the type java.lang.Class, in the heap area.

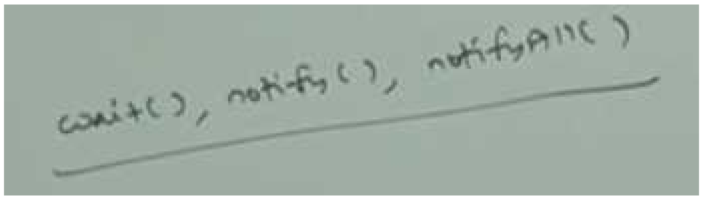
2. Programmer can use this class object to get class level information.

3. We can use getClass(), very frequently in reflections.

finalize(), Method:

Just befor destroying an object garbage collector calls finalize(), method to perform cleanup activities.

Once finalize method completes automatically GC destroys that object.



wait(), notify(), notifyAll()

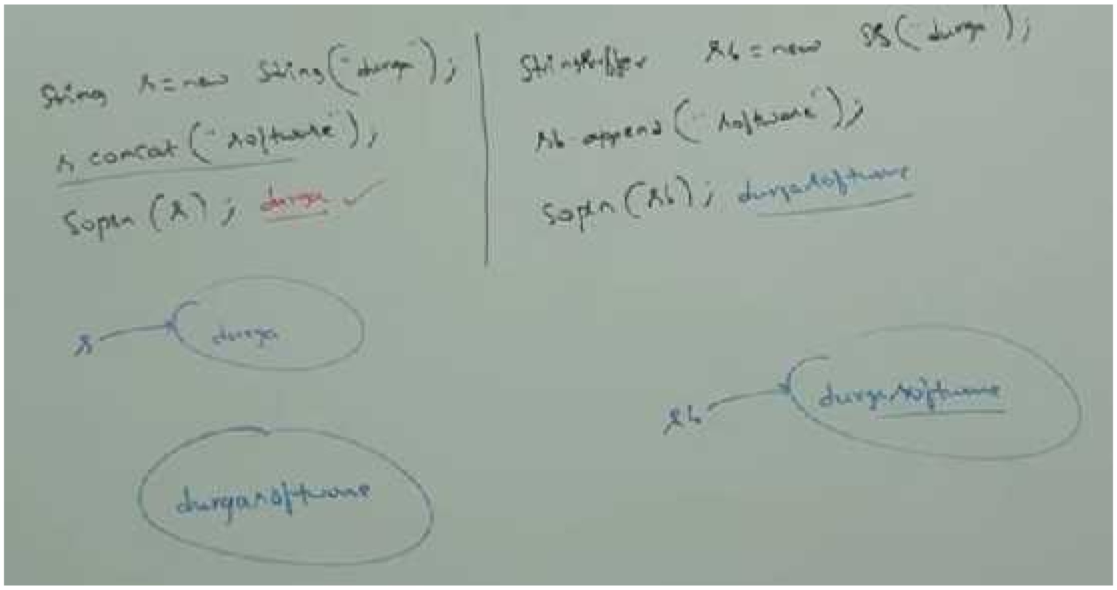
We can use these methods interthread communication. The thread which is expecting updation, it os responsible to call wait(), method. Then immediately the thread will entered into waiting state.

The thread which is responsible to perform updation, after performing updation, the thread can call notify method. The waiting thread will get that notification and continue it's execution with those updates.

java.lang.String:

Once we create a string object we can't perform any changes on the existing object. If we are trying to perform any change with those changes a new object will be created. this non changable behaviour is nothing but immutabilty of string.

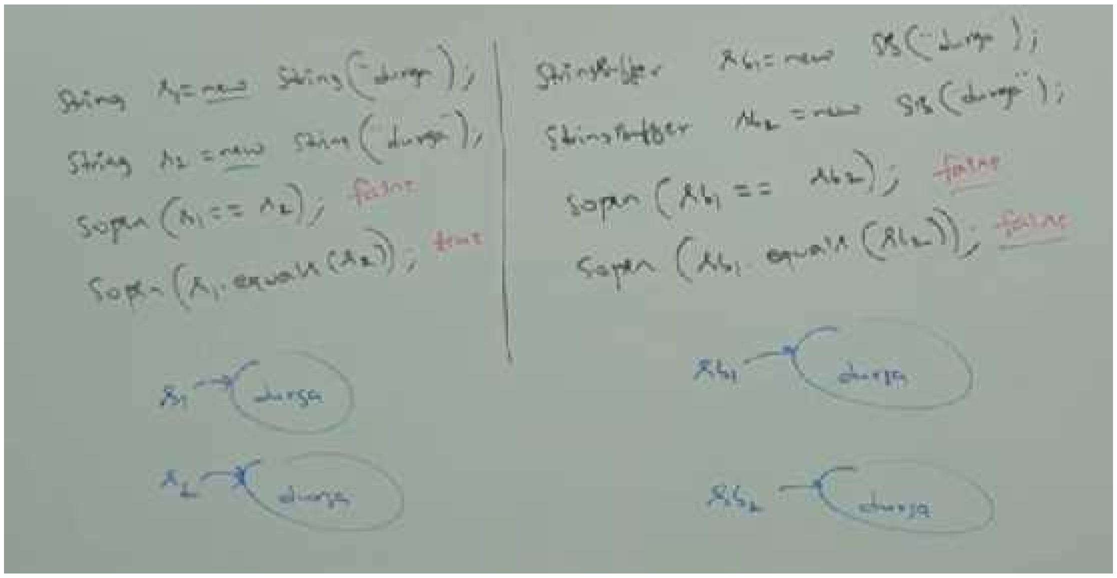
Once we creates string buffer object we can perform any change in the existing object . This changable behaviour is nothing but mutability of stringBffer object



```
String s = new String("durga");
s.concat("software");
Sopln(s); durga ✓
```

```
StringBuffer sb = new SB("durga");
sb.append("software");
Sopln(sb); durgasoftware
```

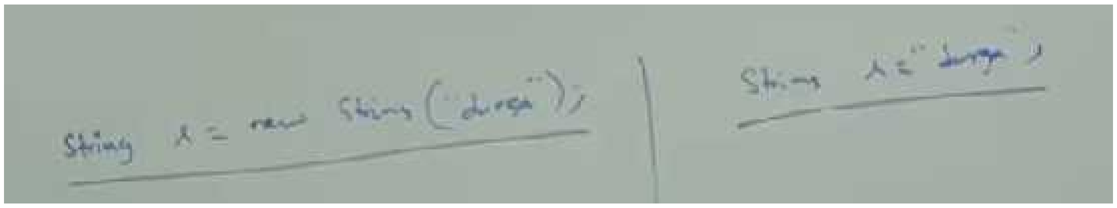
Case 2:



```
String s1 = new String("durga");
String s2 = new String("durga");
Sopln(s1 == s2); false
Sopln(s1.equals(s2)); true
```

```
StringBuffer sb1 = new SB("durga");
StringBuffer sb2 = new SB("durga");
Sopln(sb1 == sb2); false
Sopln(sb1.equals(sb2)); false
```

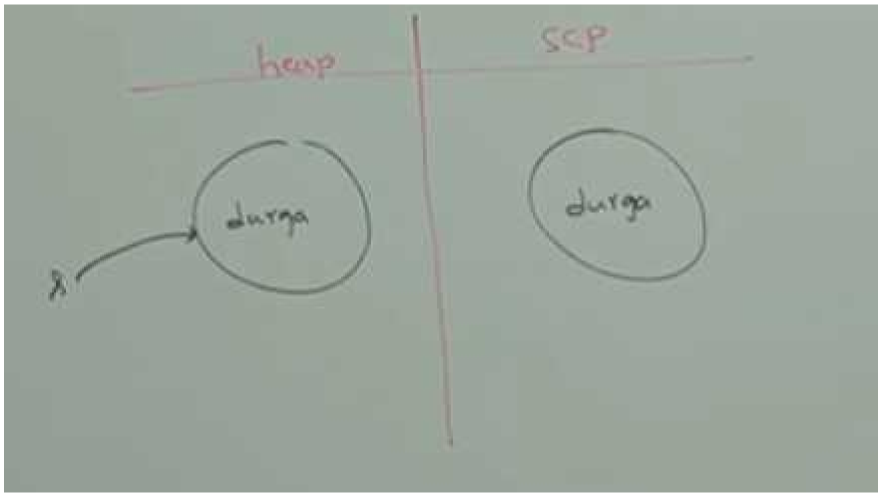
Case 3:



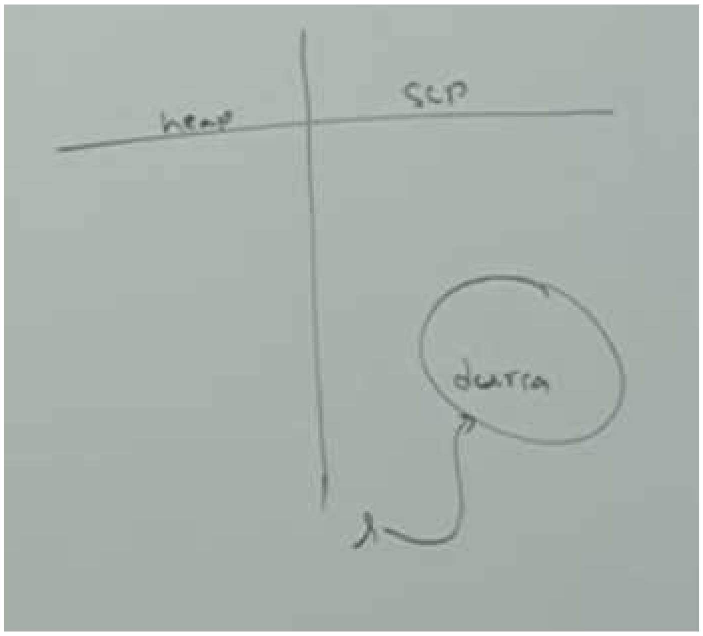
```
String s = new String("durga");
```

```
String s = "durga";
```

In first case two object will be created, one in heap area and other in scp and 's' is always pointing to heap object.



In 2nd case only one object will be created in scp and 's' is alwas pointing to that object.



Note:

1. Object creation in scp is always optional. 1 st it will check is there any object already present in scp with required content, if object already present then existing object will be reused.

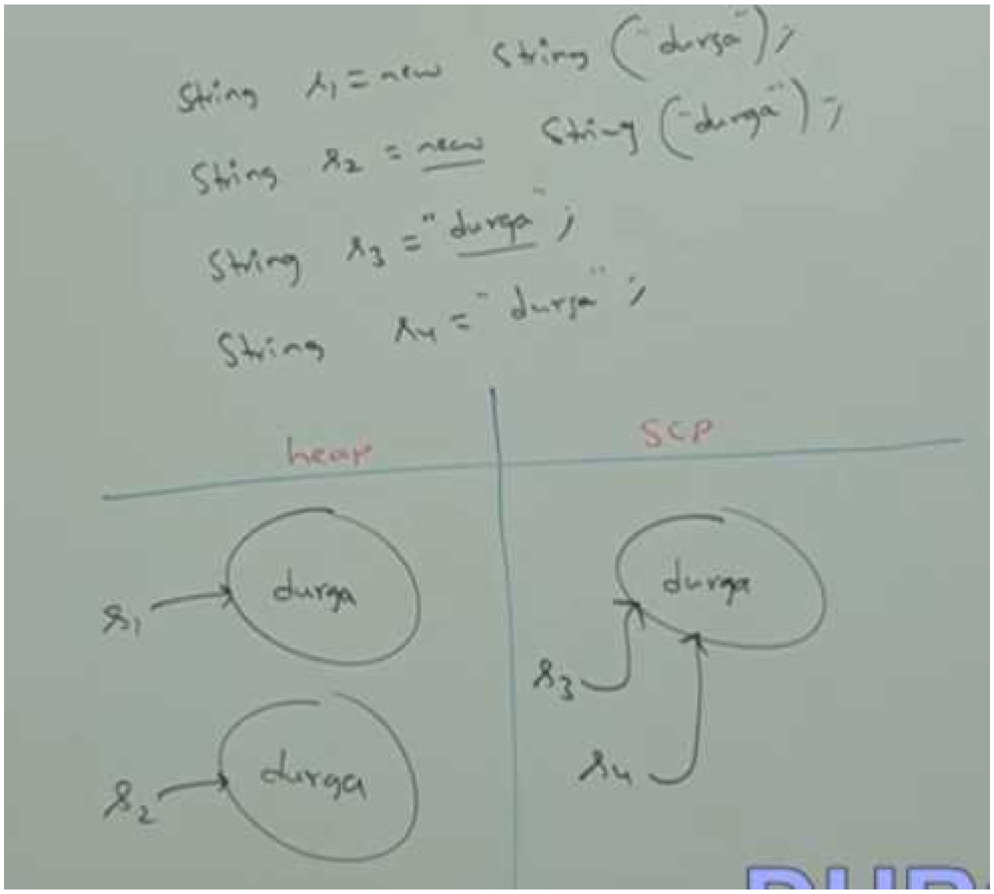
if object not already available then only a new object will be created.

But this rule is appicable only for scp not for the heap.

2. GC is not allowed to access scp area.Hence even though object does not contain refernce variable it is not eligible for GC, if it is present in SCP area.

3. All scp objects will be destroyed automatically at the time of JVM shutdown.

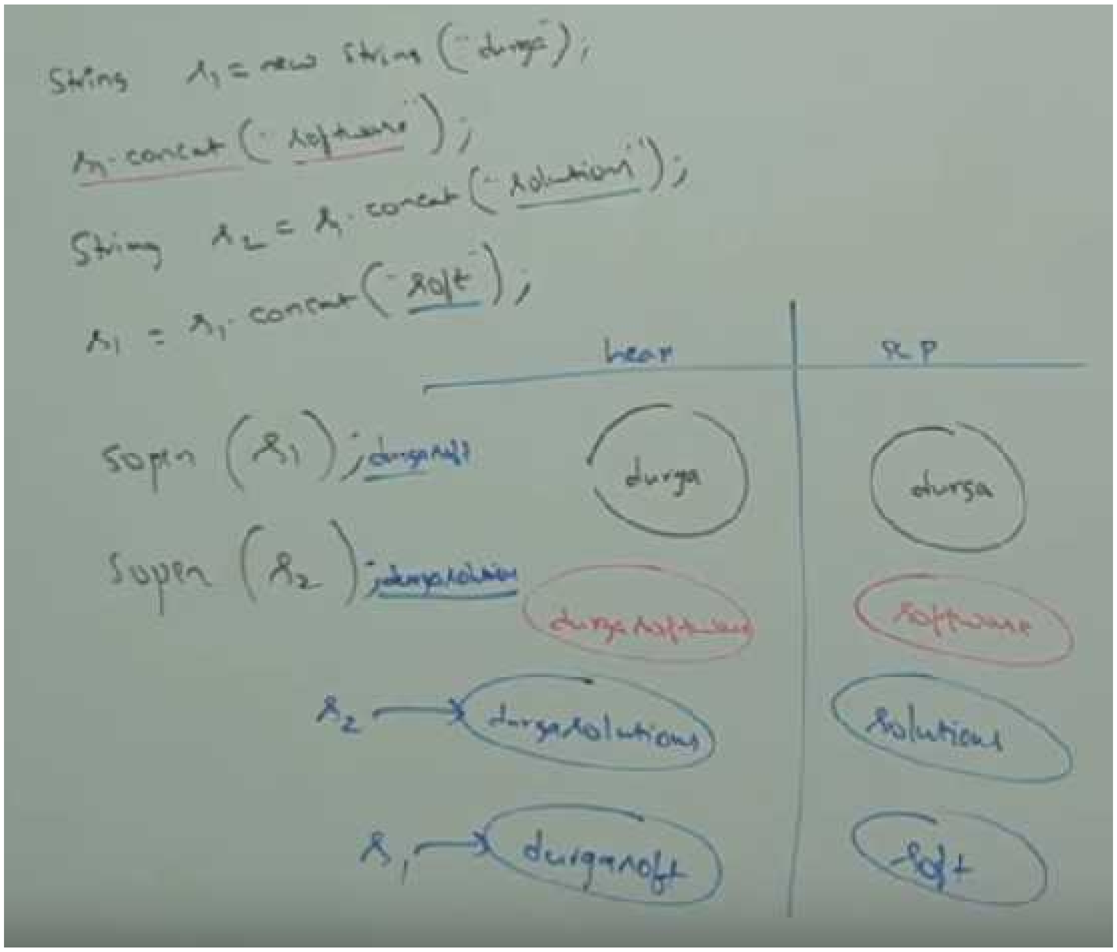
e.g. 2:



```
String s1 = new String("durga");
String s2 = new String("durga");
String s3 = "durga";
String s4 = "durga";
```

Note:

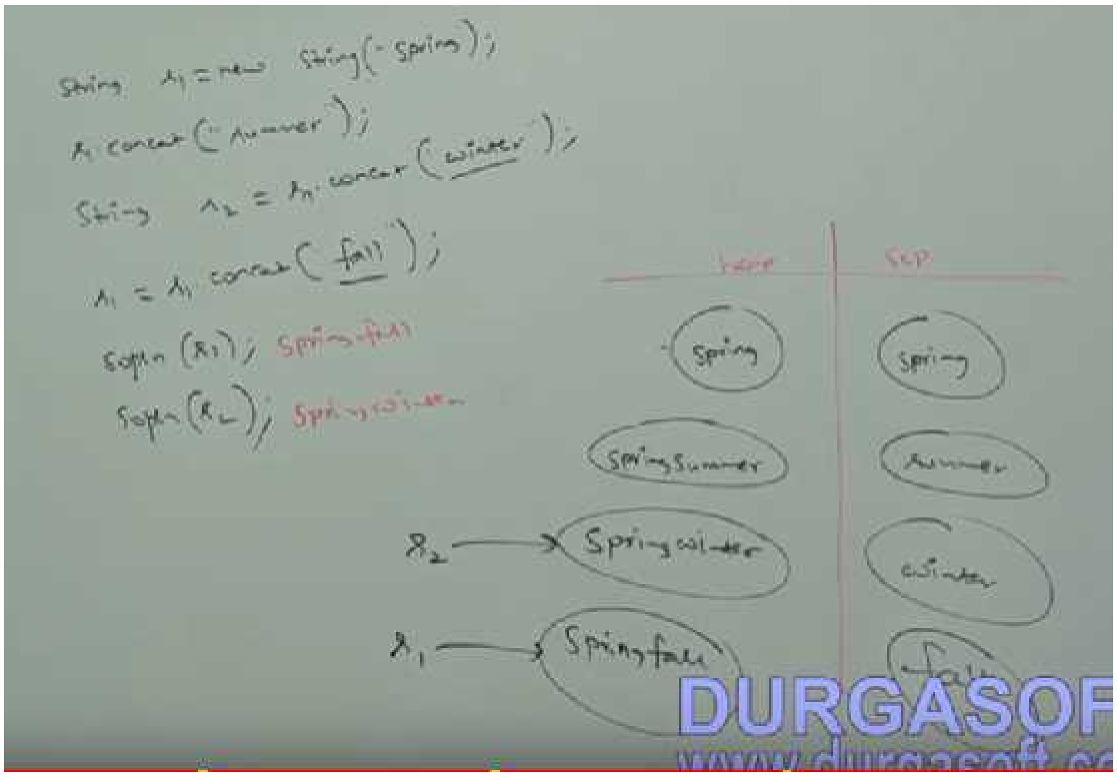
Whenever we are using new operator compulsory a new object will be created in the heap area.Hence there may be a chance of existing two objects with same content in the heap area, but not in scp. i.e. duplicate objects are possible in the heap area but not in scp.



Note:

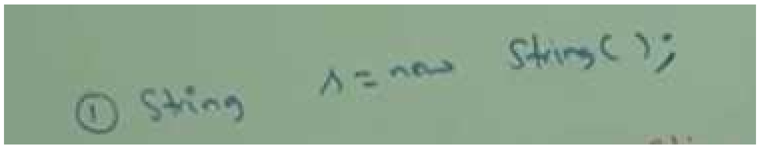
1. For every string constant one object will be placed in scp area.

2. Because of some runtime operation if an object is required to create that object will be placed only in the heap area but in scp.



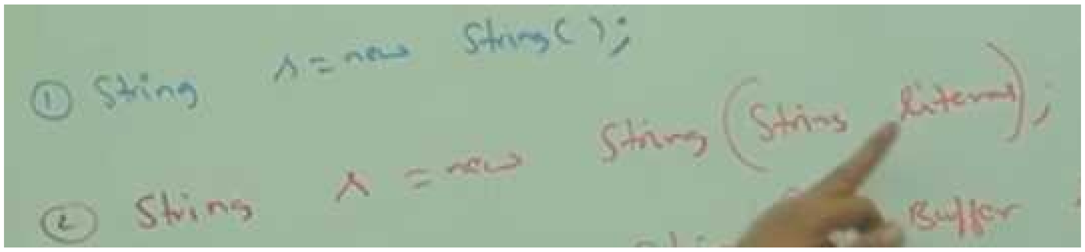
```
String s1 = new String("spring");
s1.concat("summer");
String s2 = s1.concat("winter");
s1 = s1.concat("fall");
sopln(s1); springfall
sopln(s2); springwinter
```

Constructors of String Class:



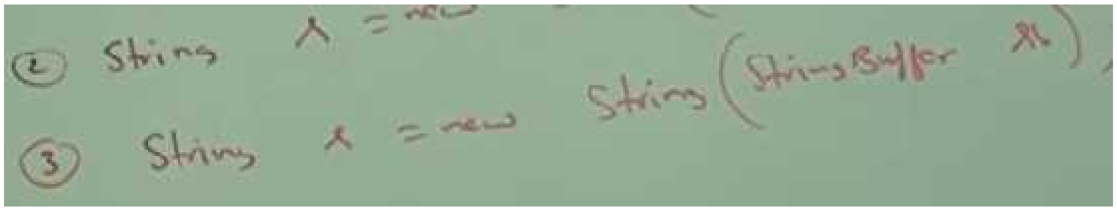
```
① String s = new String();
```

Creates an empty string object.



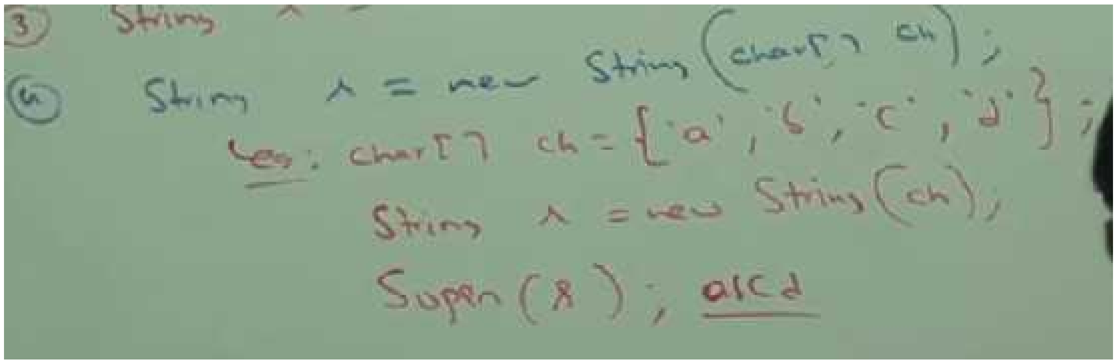
```
① String s = new String();
② String s = new String(String literal);
```

Creats a string object on the heap for the given string literial.



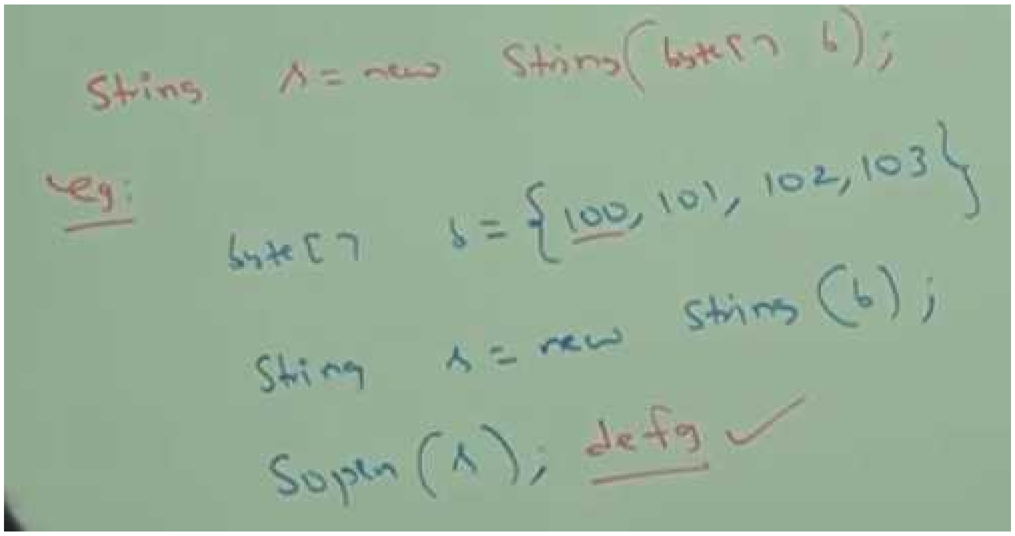
```
② String s = new
                       String(StringBuffer sb)
③ String s = new
```

Creates an equivalent string object for the given stringBuffer.



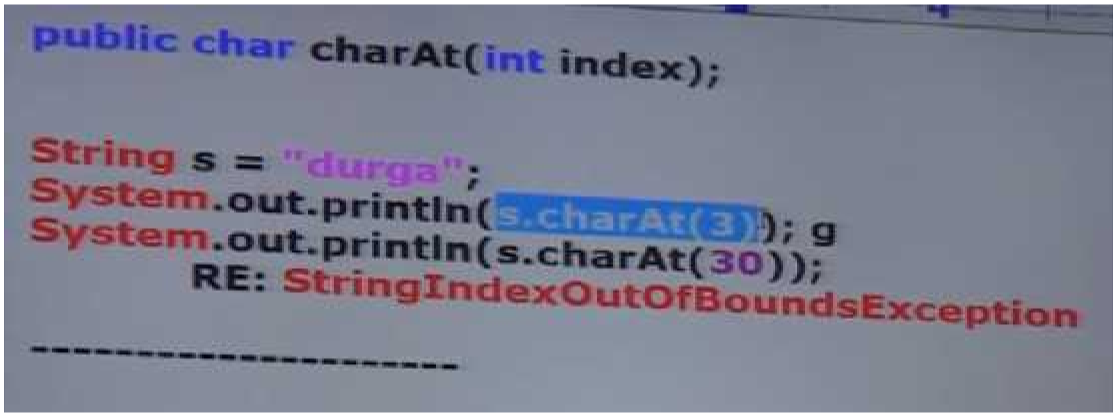
```
③ String s =
④ String s = new String(char[] ch);
  eg: char[] ch = {'a', 'b', 'c', 'd'};
      String s = new String(ch);
      Sopln(s); abcd
```

Creates an equivalent string object for the given char array.

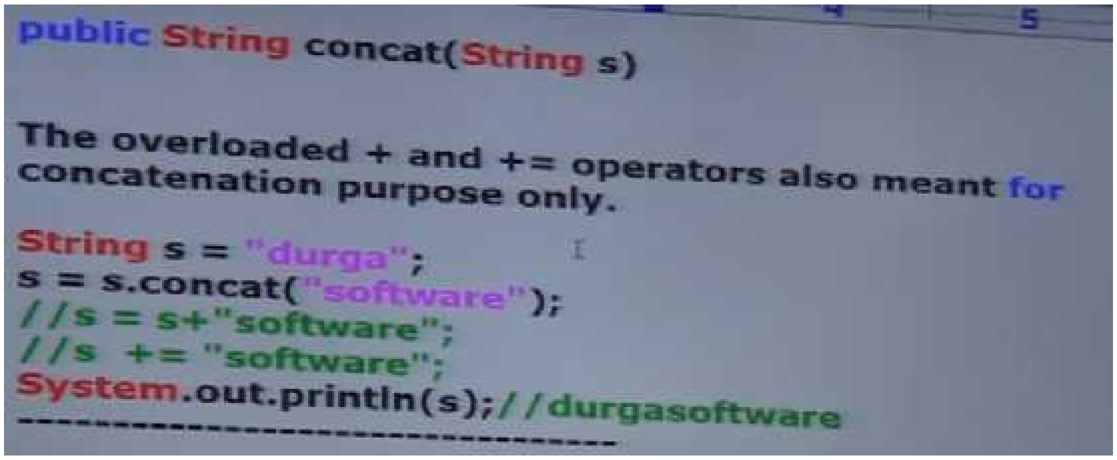


```
String s = new String(byte[] b);
eg:
    byte[] b = {100, 101, 102, 103}
    String s = new String(b);
    Sopln(s); defg ✓
```

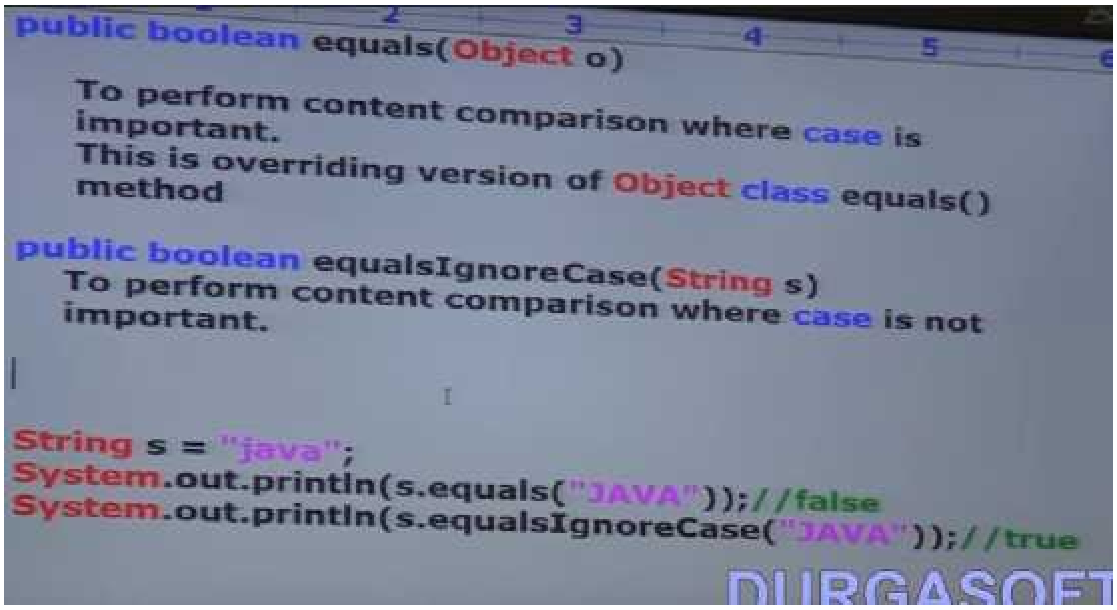
Importment methods of string class:



```
public char charAt(int index);

String s = "durga";
System.out.println(s.charAt(3)); g
System.out.println(s.charAt(30));
        RE: StringIndexOutOfBoundsException

--------------------
```

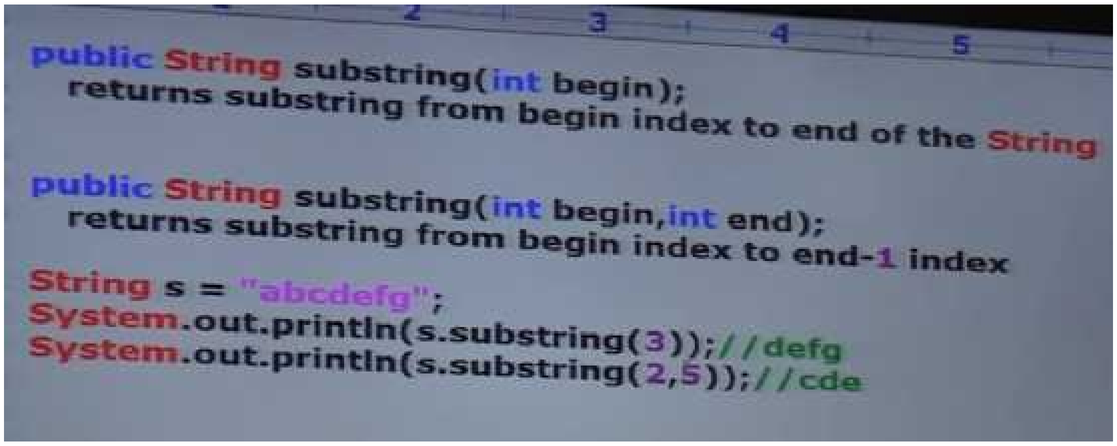


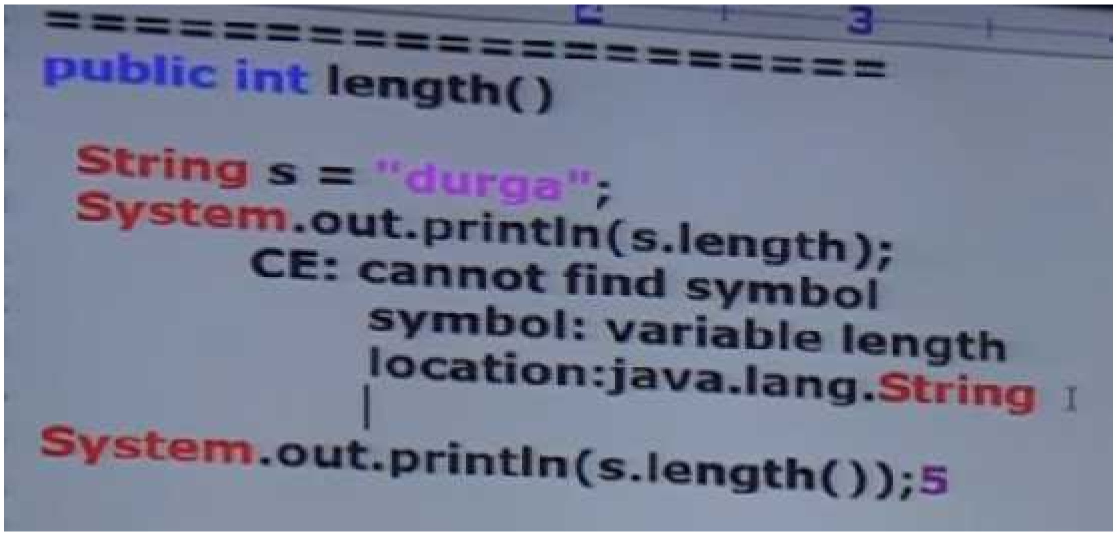
```
public String concat(String s)

The overloaded + and += operators also meant for
concatenation purpose only.

String s = "durga";
s = s.concat("software");
//s = s+"software";
//s  += "software";
System.out.println(s);//durgasoftware
-----------------------------
```

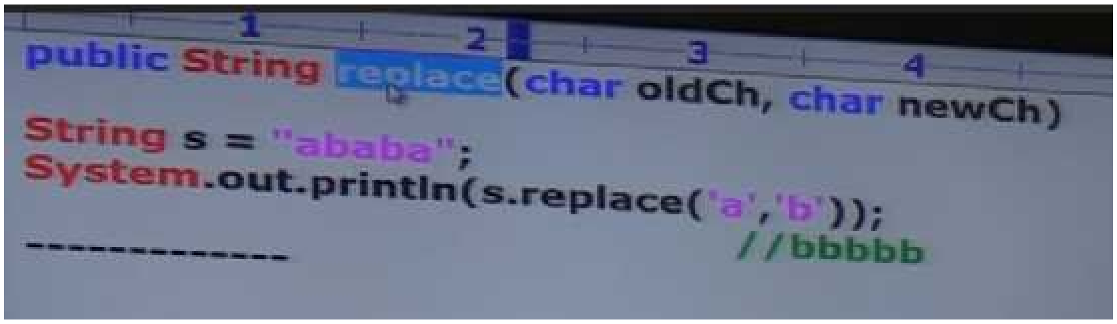


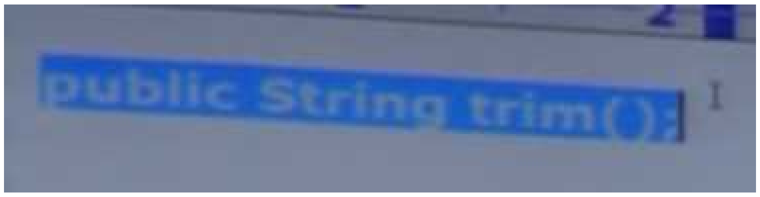
```
public boolean equals(Object o)
    To perform content comparison where case is
    important.
    This is overriding version of Object class equals()
    method

public boolean equalsIgnoreCase(String s)
   To perform content comparison where case is not
   important.


String s = "java";
System.out.println(s.equals("JAVA"));//false
System.out.println(s.equalsIgnoreCase("JAVA"));//true
```

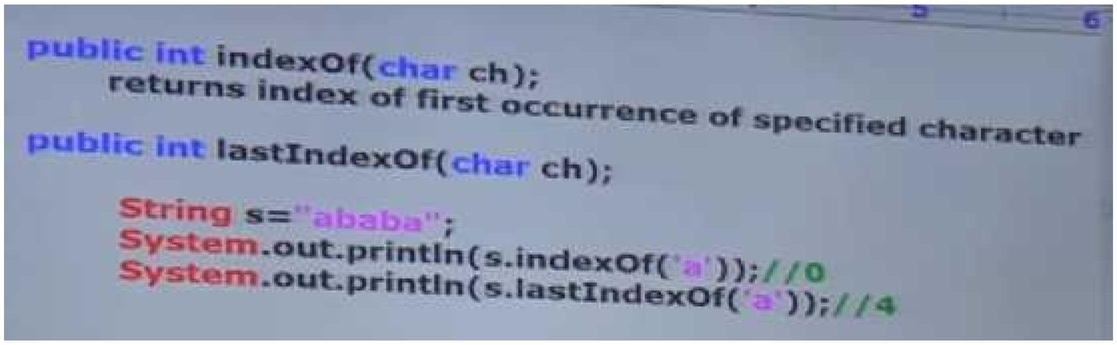
DURGASOFT


```
public String substring(int begin);
  returns substring from begin index to end of the String

public String substring(int begin,int end);
  returns substring from begin index to end-1 index

String s = "abcdefg";
System.out.println(s.substring(3));//defg
System.out.println(s.substring(2,5));//cde
```

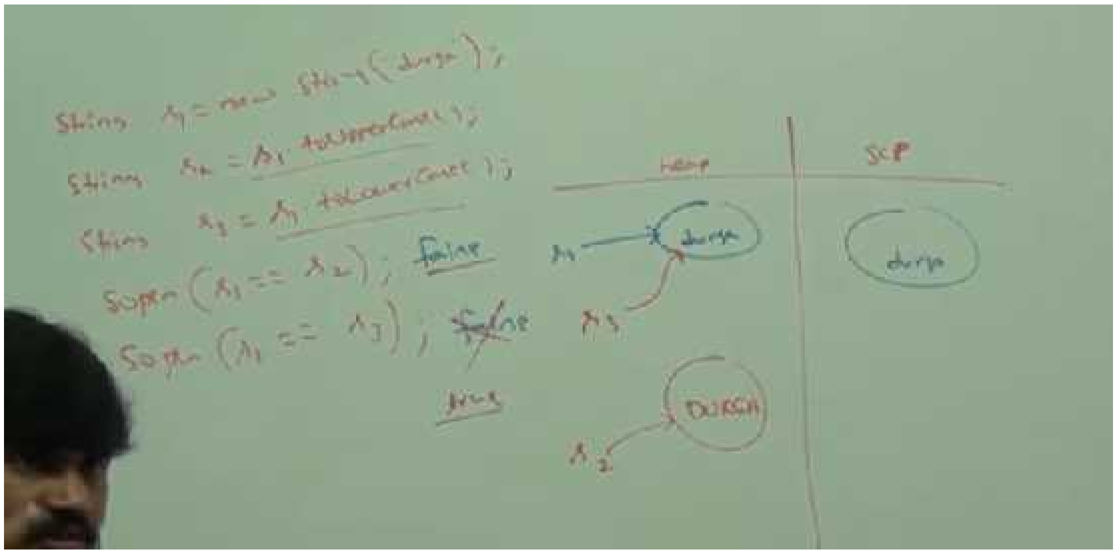

```
=======================
public int length()

  String s = "durga";
  System.out.println(s.length);
          CE: cannot find symbol
                symbol: variable length
                location:java.lang.String

 System.out.println(s.length());5
```


```
public String replace(char oldCh, char newCh)
String s = "ababa";
System.out.println(s.replace('a','b'));
-------------                          //bbbbb
```


```
public String trim();
```



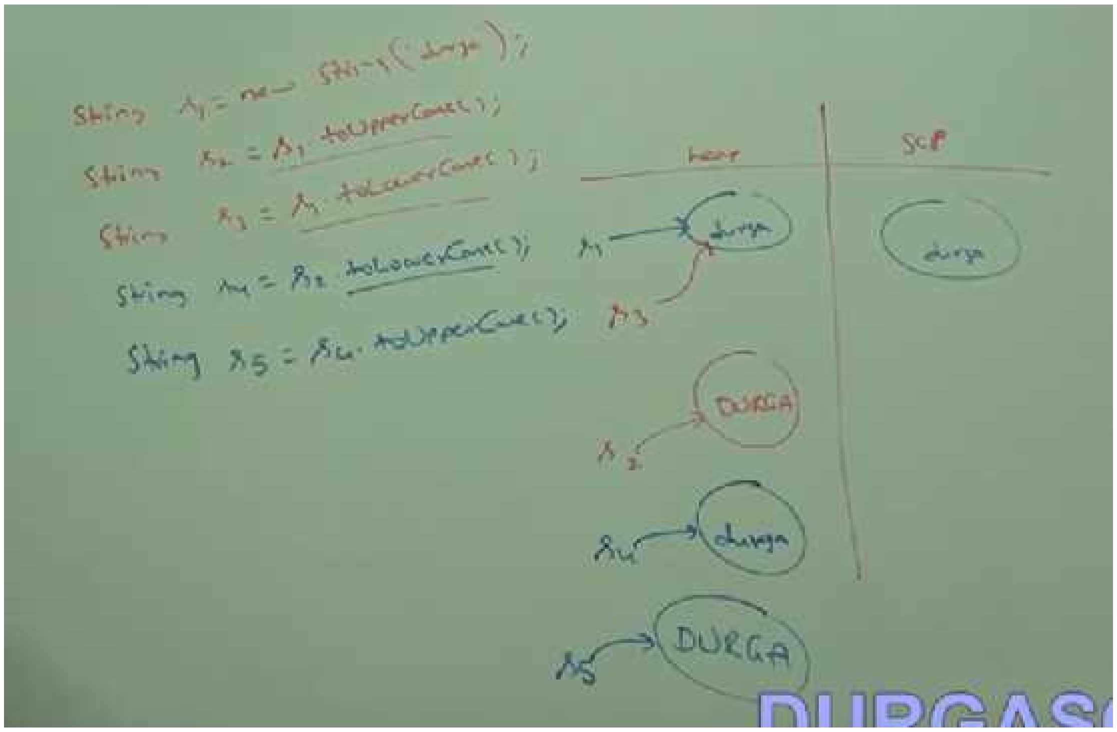
```
public int indexOf(char ch);
        returns index of first occurrence of specified character
public int lastIndexOf(char ch);

        String s="ababa";
        System.out.println(s.indexOf('a'));//0
        System.out.println(s.lastIndexOf('a'));//4
```



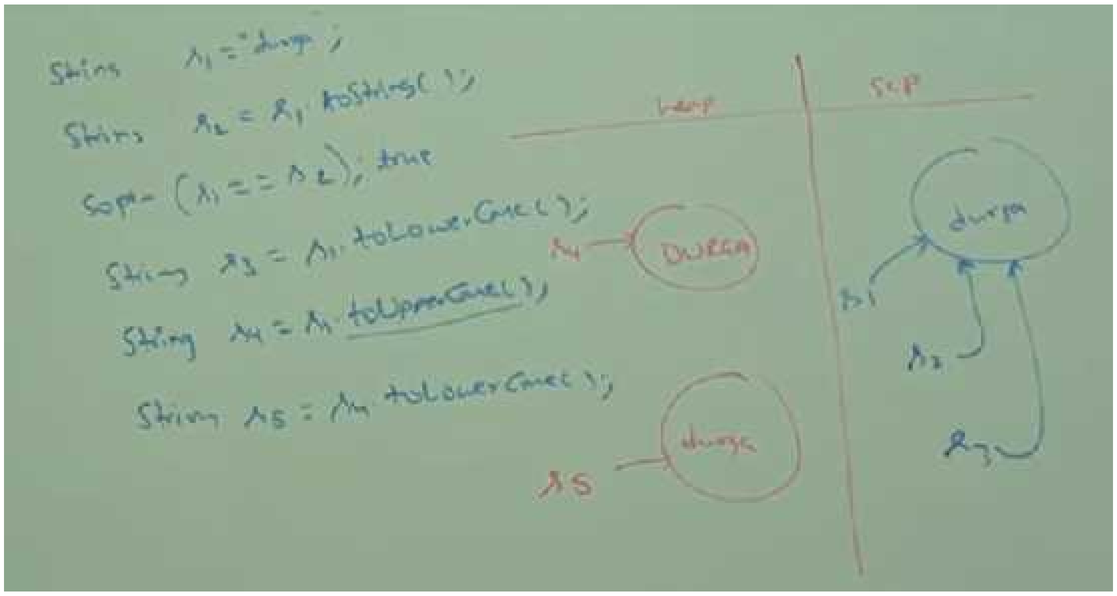
```
String s1 = new String("durga");
String s2 = s1.toUpperCase();
String s3 = s1.toLowerCase();
Sopln(s1 == s2);  false
Sopln(s1 == s3);  ~~false~~
                  true
```

Because of runtime operation if there is a change in content then with those changes a new object will be created on the heap. If there is no change in the content then existing object will be reused and new object won't be created.

Whether the object present in heap or scp, the rule is same.



```
String s1 = new String("durga");
String s2 = s1.toUpperCase();
String s3 = s1.toLowerCase();
String s4 = s2.toLowerCase();
String s5 = s4.toUpperCase();
```



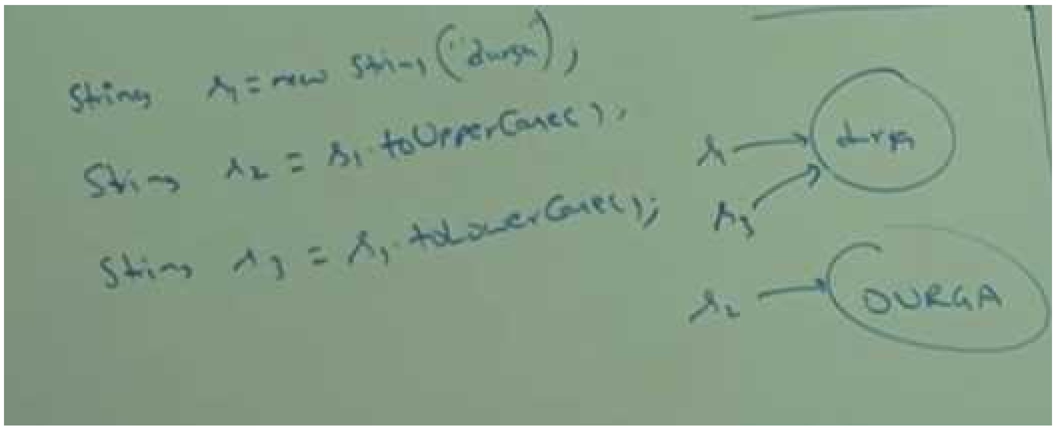
```
String s1 = "durga";
String s2 = s1.toString();
sop(s1 == s2); true
String s3 = s1.toLowerCase();
String s4 = s1.toUpperCase();
String s5 = s4.toLowerCase();
```

Q. How to create our own immutable class.

Once we create an object we can't perform any changes in that object. If we are trying to perform any change and if there is a change in the content then with those changes a new object will be created.

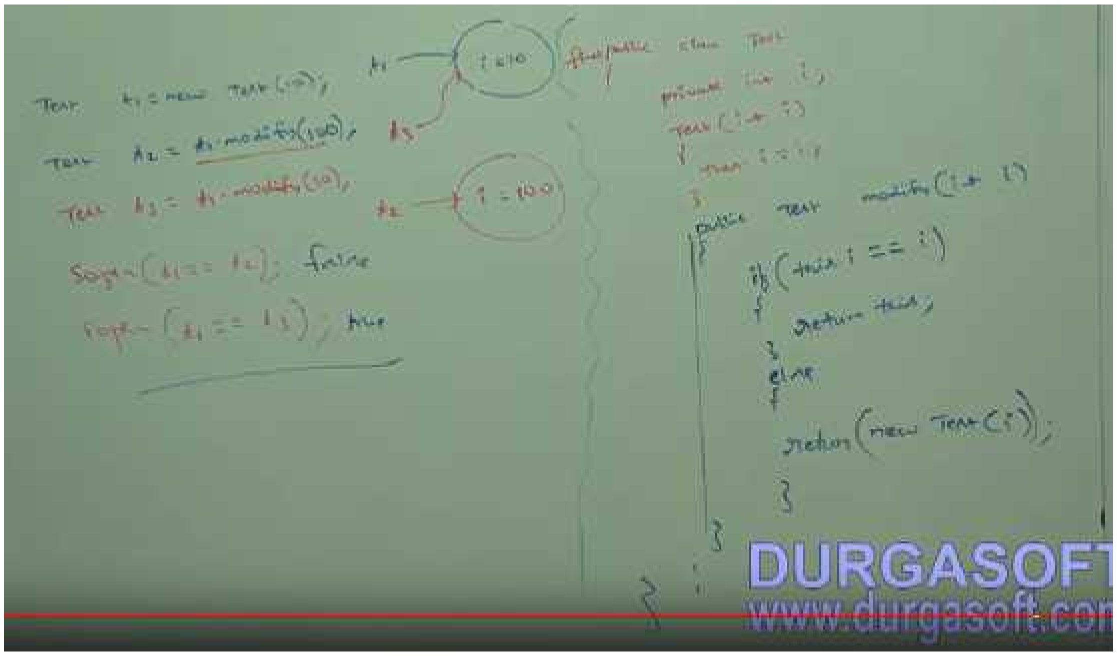
If there is no change in content then existing object will be reused.

This behaviour is nothing but immutability.



```
String s1 = new String("durga");
String s2 = s1.toUpperCase();
String s3 = s1.toLowerCase();
```

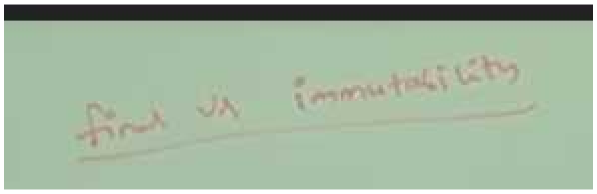
We can create our own immutable class.



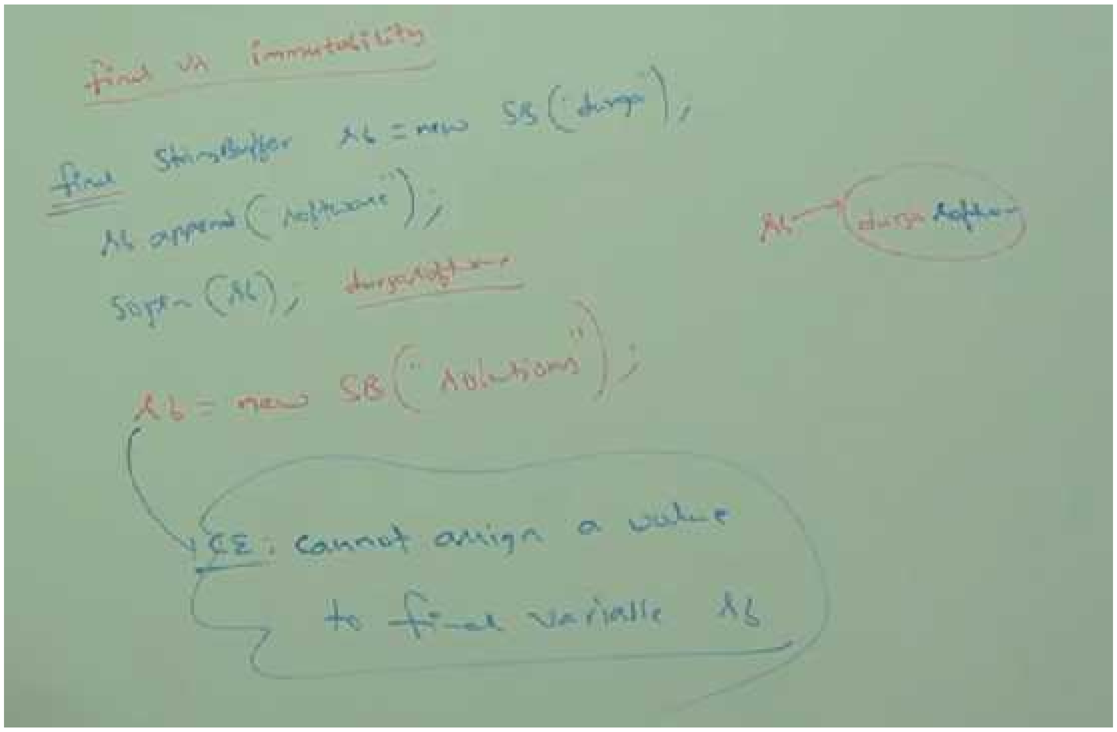
```
Test t1 = new Test(10);
Test t2 = t1.modify(100);
Test t3 = t1.modify(10);
Sop(t1 == t2); false
Sop(t1 == t3); true
```

```
final public class Test
{
    private int i;
    Test(int i)
    {
        this.i = i;
    }
    public Test modify(int i)
    {
        if(this.i == i)
        {
            return this;
        }
        else
        {
            return (new Test(i));
        }
    }
}
```

t1
t3
i = 10
t2
i = 100
DURGASOFT
www.durgasoft.com



final vs immutability



```
final vs immutability

final StringBuffer sb = new SB("durga");
sb.append("software");
Sopln(sb);   durgasoftware
sb = new SB("solutions");

CE: cannot assign a value
    to final variable sb

sb → durgasoftware
```

final applicable for variables but not for objects where as immutabilty applicable for objects but for variables.

By declaring a reference variable as final we won't get any immutability nature. Even though refrenvce variable is final we can perform any type of change on the corrosponding object but we can't perform reassignment for that variable.

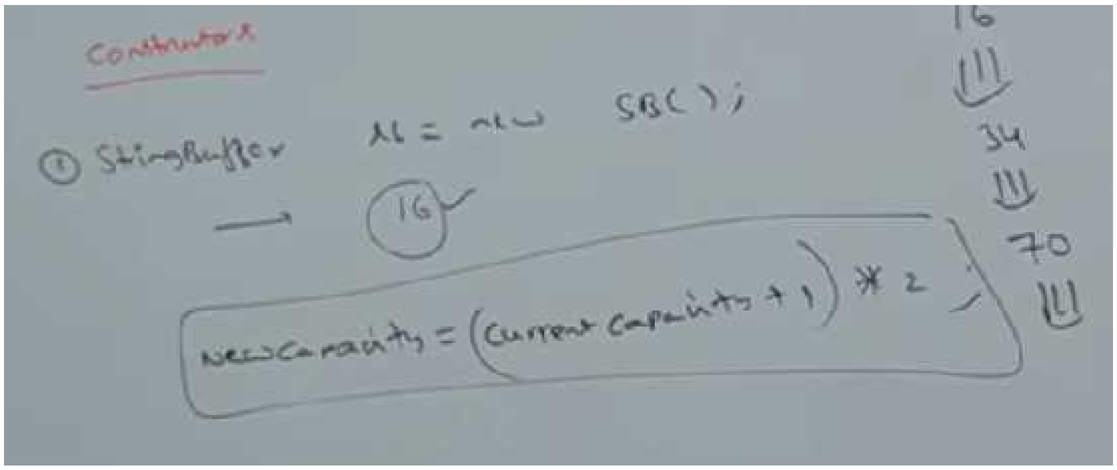
Henace final and immutable both are different concepts.

StringBuffer:

If the content is fixed and won't change frequently, then it is recomendate to go for String.

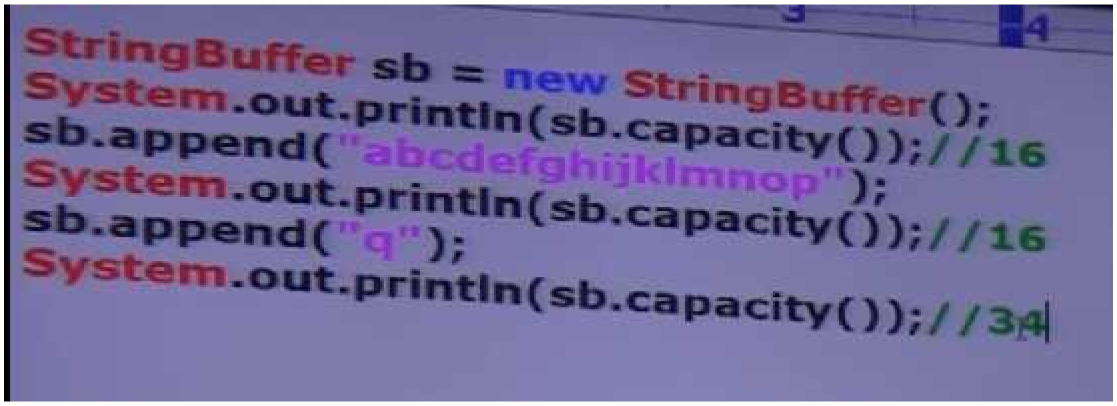
If the content is not fixed and keep on changing then it is not recomendate to use string, because for every change a new object will be created which effects performance of the system.

To handle this requirement we should go for StringBuffer. The main advantadge of StringBuffer over String is all required changes will be performed in the existing objct only.

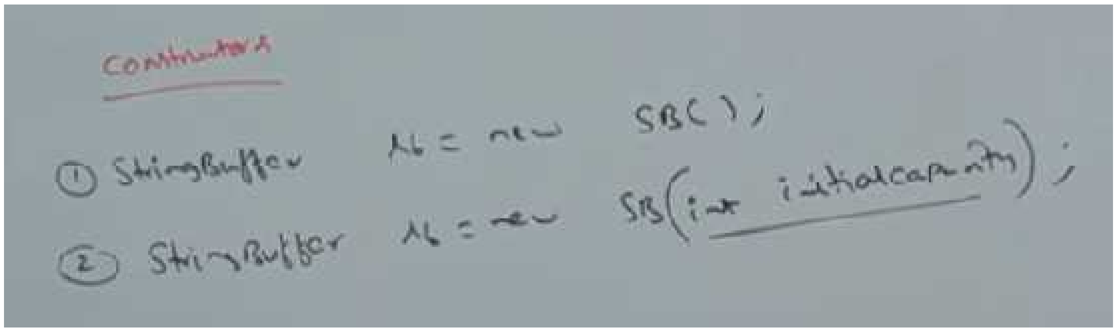

Constructors
① StringBuffer sb = new SB();
16
newCapacity = (current capacity + 1) * 2 ;
16
34
70

Creates an empty StringBuffer Object with default initial capacity 16.

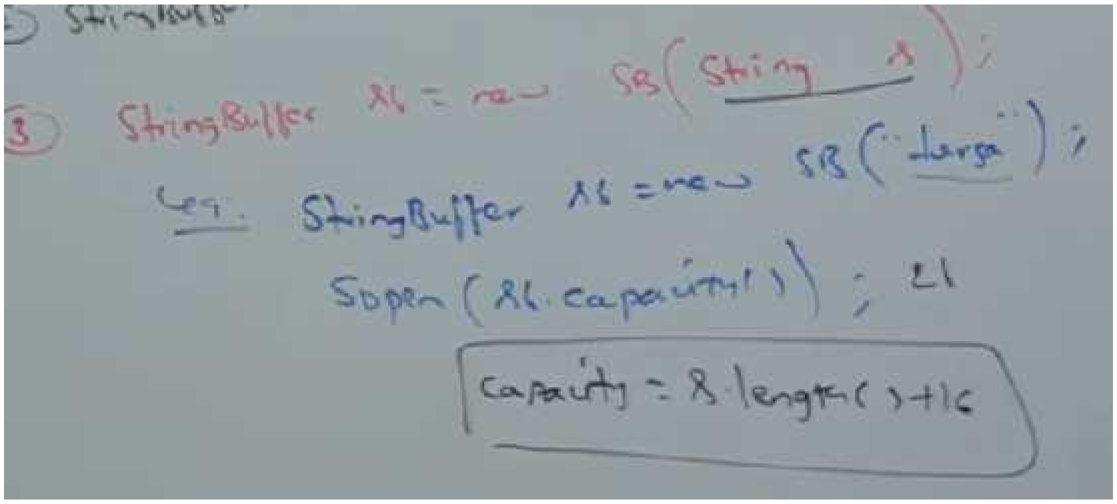
Once StringBuffer reaches it's max capacity a new StringBuffer will be created with new capacity = (currentCapacity + 1) * 2.



```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());//16
sb.append("abcdefghijklmnop");
System.out.println(sb.capacity());//16
sb.append("q");
System.out.println(sb.capacity());//34
```


Constructors
① StringBuffer sb = new SB();
② StringBuffer sb = new SB(int initialcapacity);

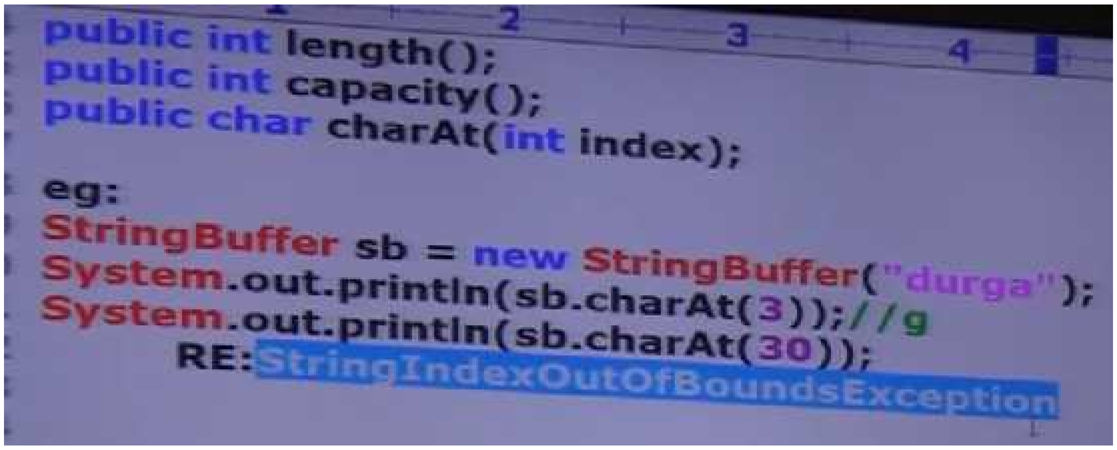
Creates an empty StringBuffer Object with specified intial capacity.

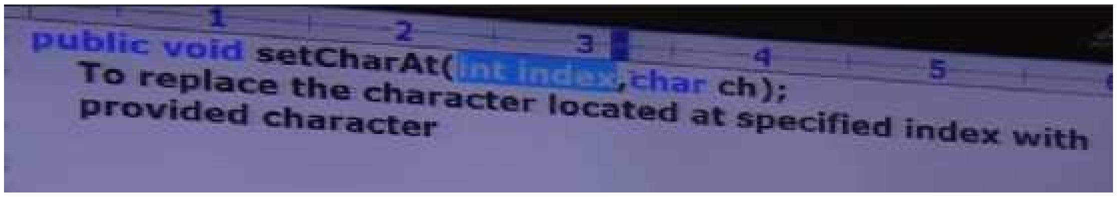

```
3) StringBuffer sb = new SB(String s);
   eg: StringBuffer sb = new SB("durga");
       Sopln(sb.capacity()); 21
       capacity = s.length()+16
```

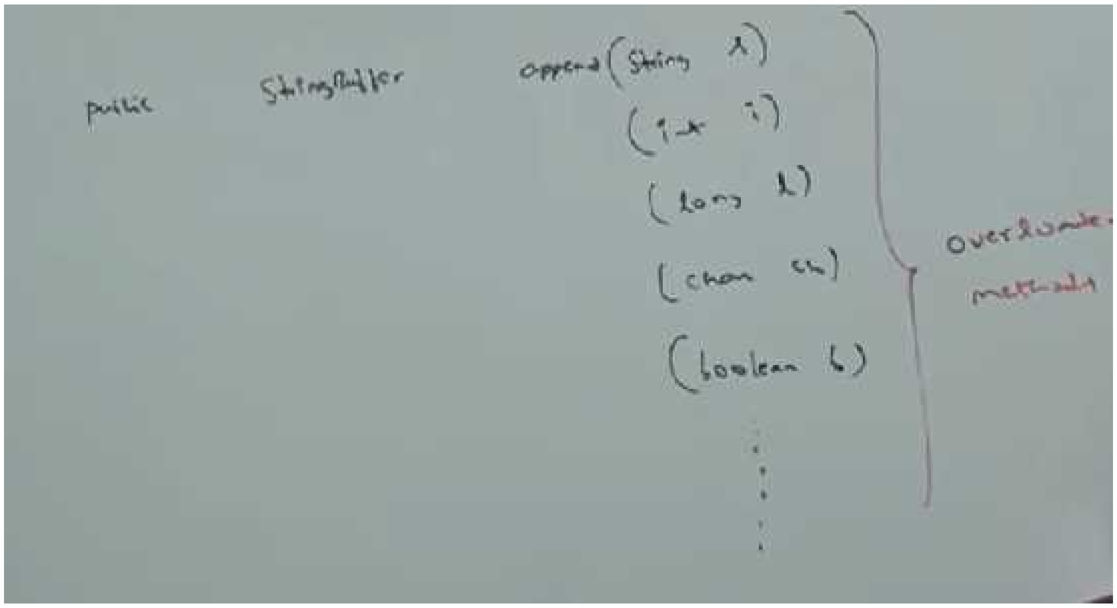
Creates an equivalent stringBuffer for the given string with

capacity= s.length()+16.

Important Methods of StringBuffer,

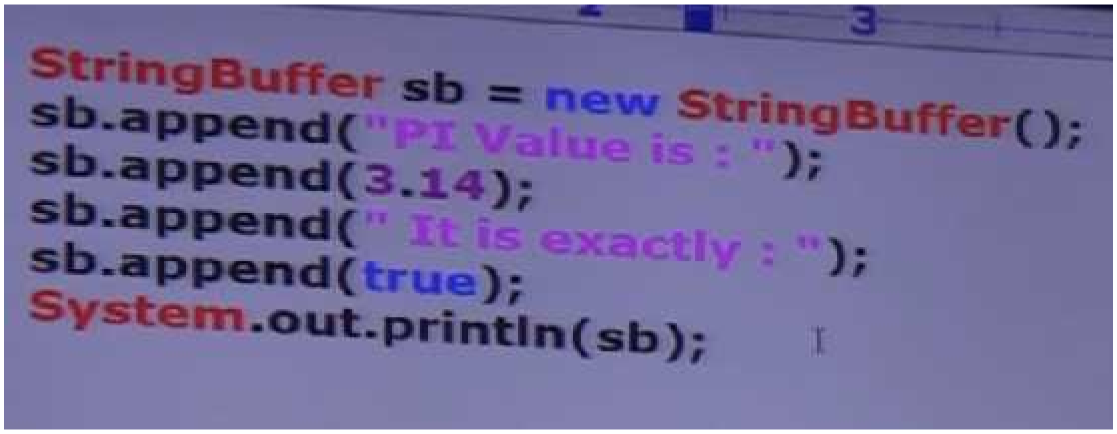

```
public int length();
public int capacity();
public char charAt(int index);

eg:
StringBuffer sb = new StringBuffer("durga");
System.out.println(sb.charAt(3));//g
System.out.println(sb.charAt(30));
      RE:StringIndexOutOfBoundsException
```


```
public void setCharAt(int index,char ch);
   To replace the character located at specified index with
   provided character
```

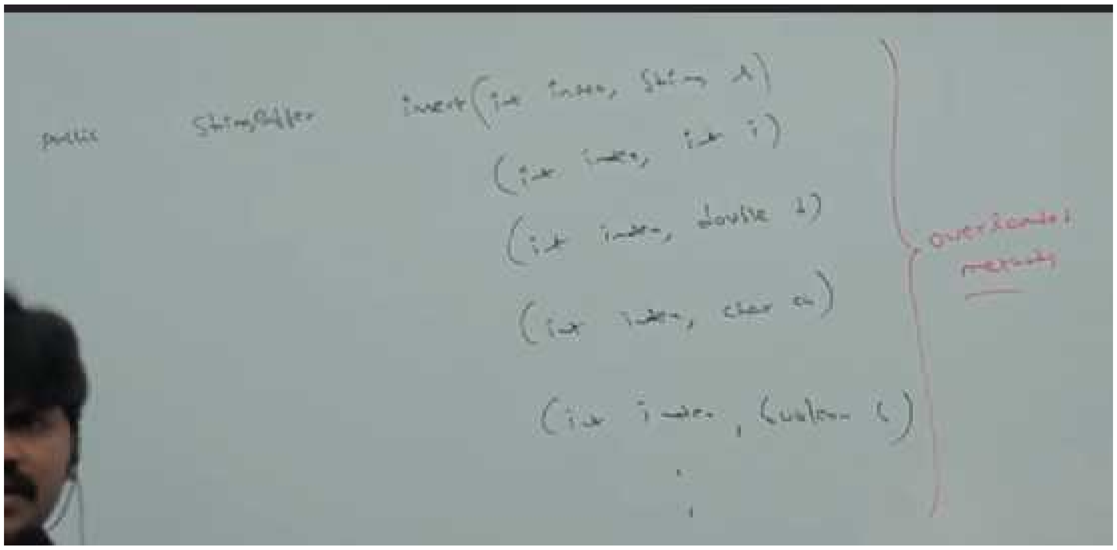


```
public    StringBuffer    append(String s)
                                (int i)
                                (long l)
                                (char ch)
                                (boolean b)
                                   .
                                   .
                                   .
```

overloaded methods

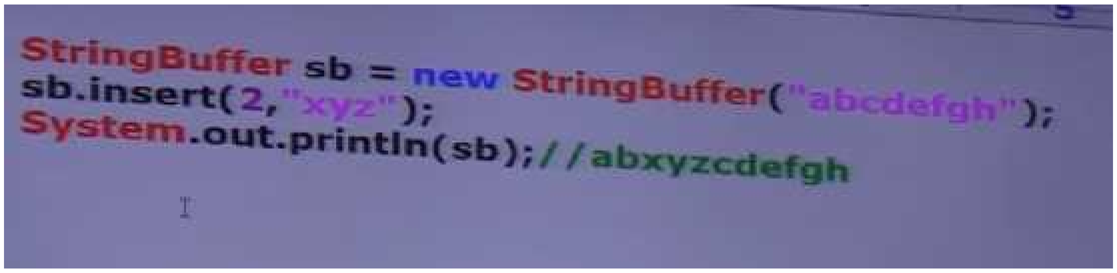


```
StringBuffer sb = new StringBuffer();
sb.append("PI Value is : ");
sb.append(3.14);
sb.append(" It is exactly : ");
sb.append(true);
System.out.println(sb);
```

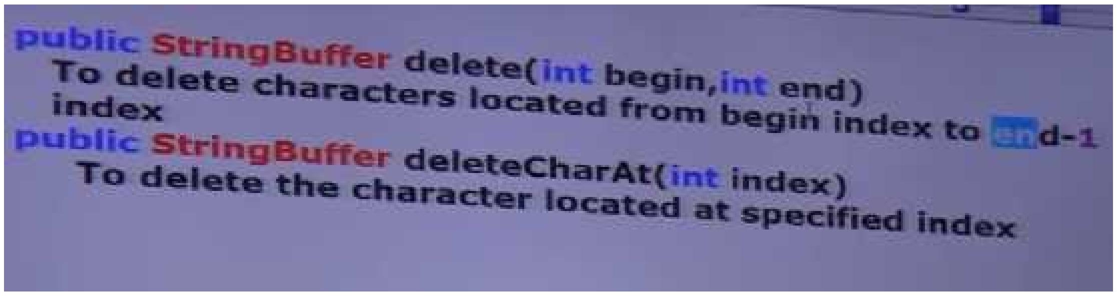


```
public    StringBuffer    insert(int index, String s)
                                (int index, int i)
                                (int index, double d)
                                (int index, char ch)
                                (int index, boolean b)
                                   .
                                   .
```

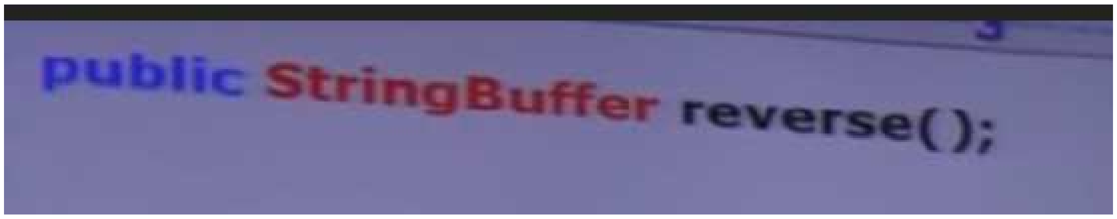
overloaded methods



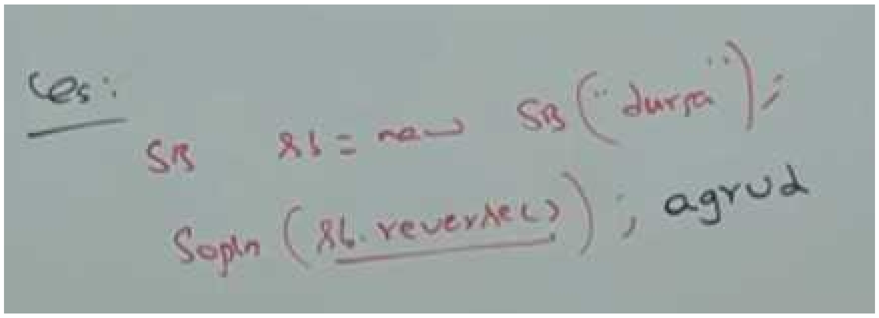
```
StringBuffer sb = new StringBuffer("abcdefgh");
sb.insert(2,"xyz");
System.out.println(sb);//abxyzcdefgh
```



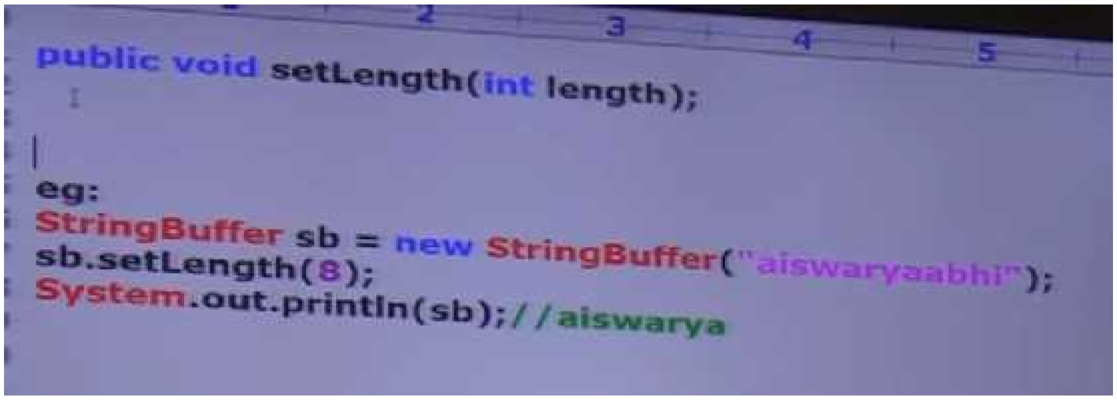
```
public StringBuffer delete(int begin,int end)
  To delete characters located from begin index to end-1
  index
public StringBuffer deleteCharAt(int index)
   To delete the character located at specified index
```



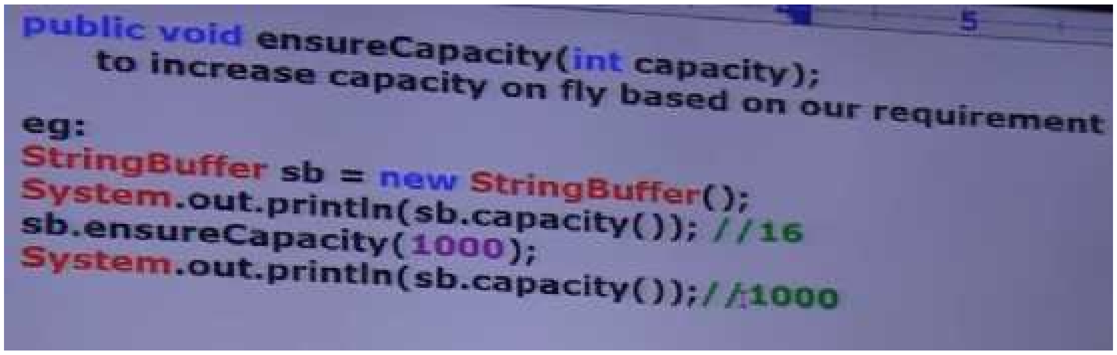
```
public StringBuffer reverse();
```



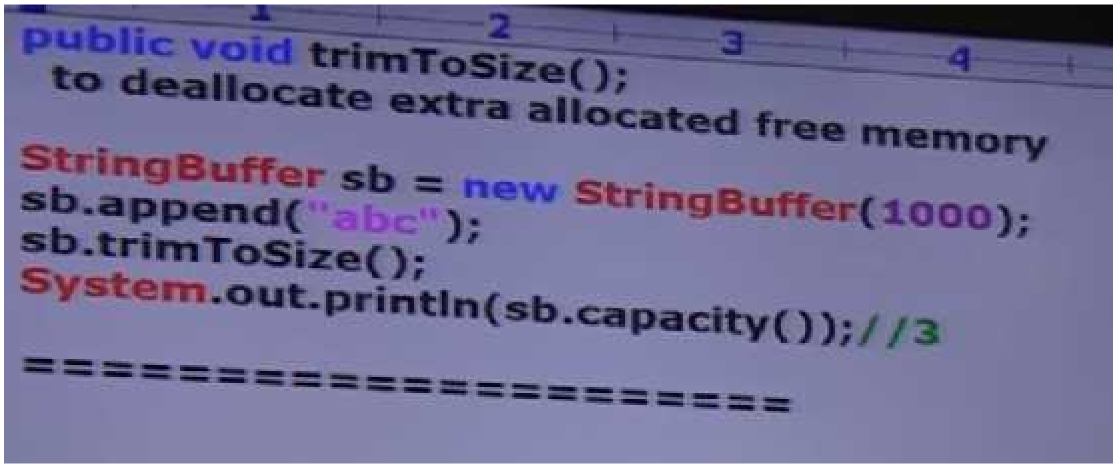
```
ex:
    SB  sb = new  SB("durga");
      Sopln (sb.reverse()); agrud
```



```
public void setLength(int length);

eg:
StringBuffer sb = new StringBuffer("aiswaryaabhi");
sb.setLength(8);
System.out.println(sb);//aiswarya
```



```
public void ensureCapacity(int capacity);
    to increase capacity on fly based on our requirement
eg:
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity()); //16
sb.ensureCapacity(1000);
System.out.println(sb.capacity());//1000
```



```
public void trimToSize();
  to deallocate extra allocated free memory

StringBuffer sb = new StringBuffer(1000);
sb.append("abc");
sb.trimToSize();
System.out.println(sb.capacity());//3

======================
```

StringBuilder:

Every method present in stringBuffer is syncronized and hence only one thread is allowed to operate on StringBuffer Object at a time, which may creates performance problems. To handle this requirement Sun people introduced StringBuilder concept in 1.5 version.

StringBuilder is exactly same as StringBuffer except the following differences,

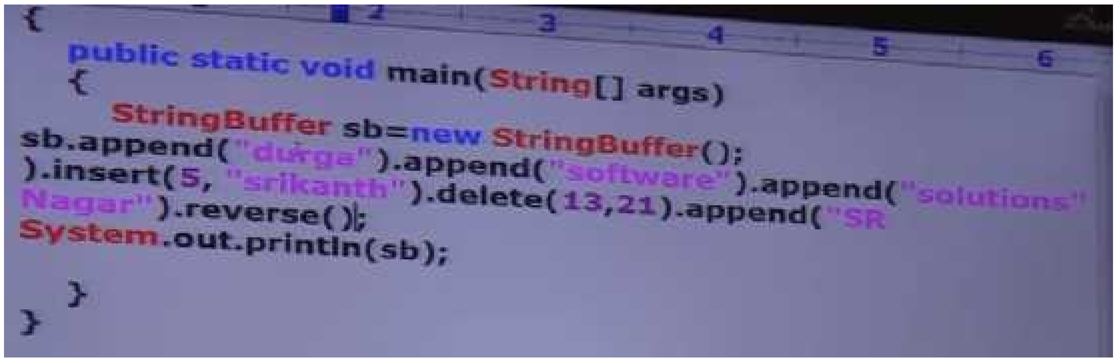
String vs StringBuffer vs StringBuilder:

1. if the content is fixed and won't change frequently then we should go for String.

2. If the content is not fixed and keep on changing but thread safty required then we should go for stringBuffer.

3. If the content is not fixed keep on changing but thread safety is not required then we should go for stringBuilder.

Method Chaining:



```
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer();
sb.append("durga").append("software").append("solutions"
).insert(5, "srikanth").delete(13,21).append("SR
Nagar").reverse();
System.out.println(sb);

    }
}
```

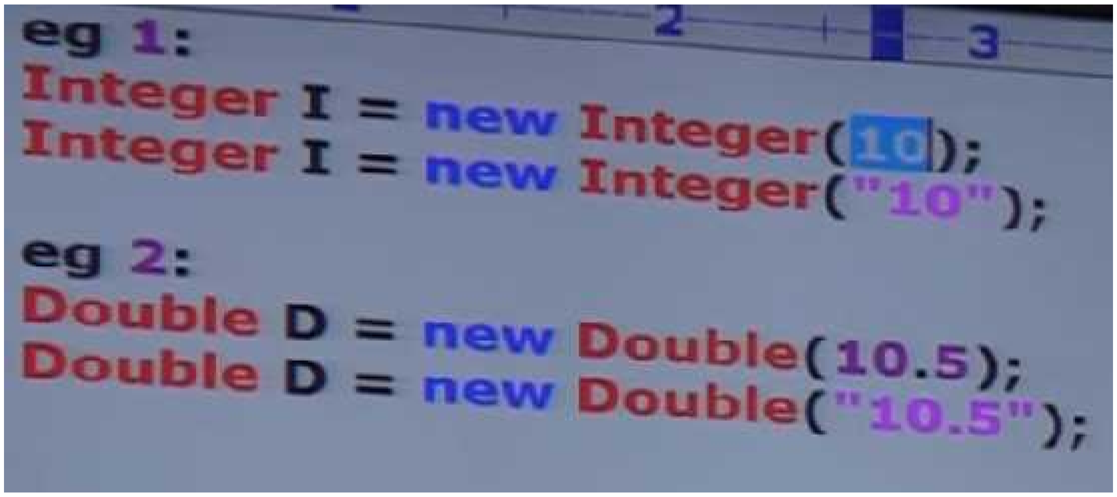
Wrapper Classes:

The main objectives of wrapper classes are to wrap primitive into object form so that we can handle premitives also, just like objects.

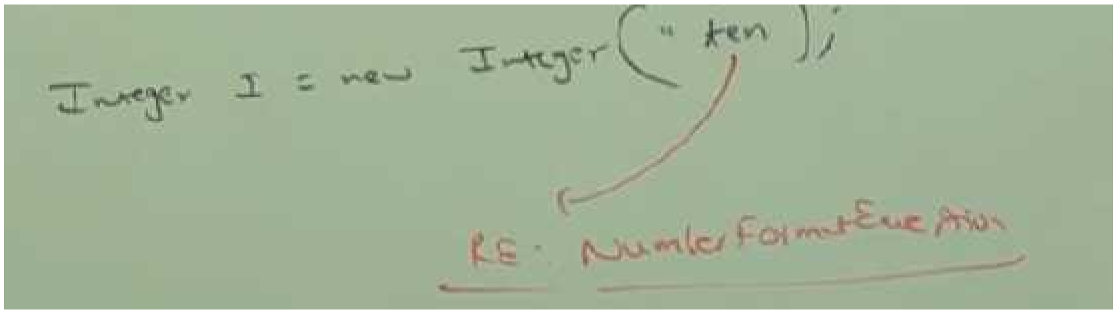
To define several utility methods which are required for the premitives.

Constructors:

Almost all wraper classes contains two constructors, one can take corrosponding premitive as argumrnt and the other can take string as argument.

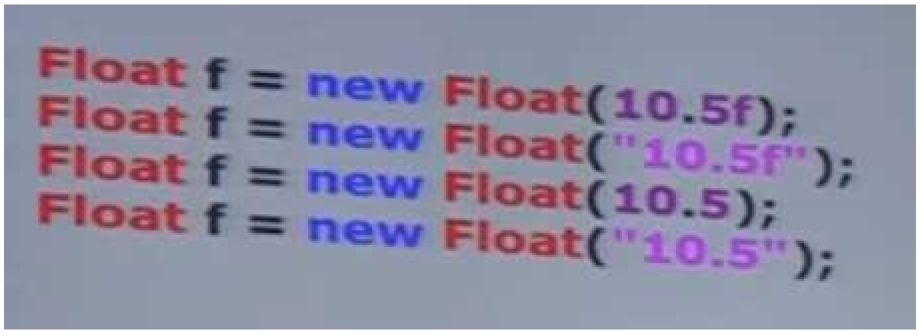


```
eg 1:
Integer I = new Integer(10);
Integer I = new Integer("10");

eg 2:
Double D = new Double(10.5);
Double D = new Double("10.5");
```

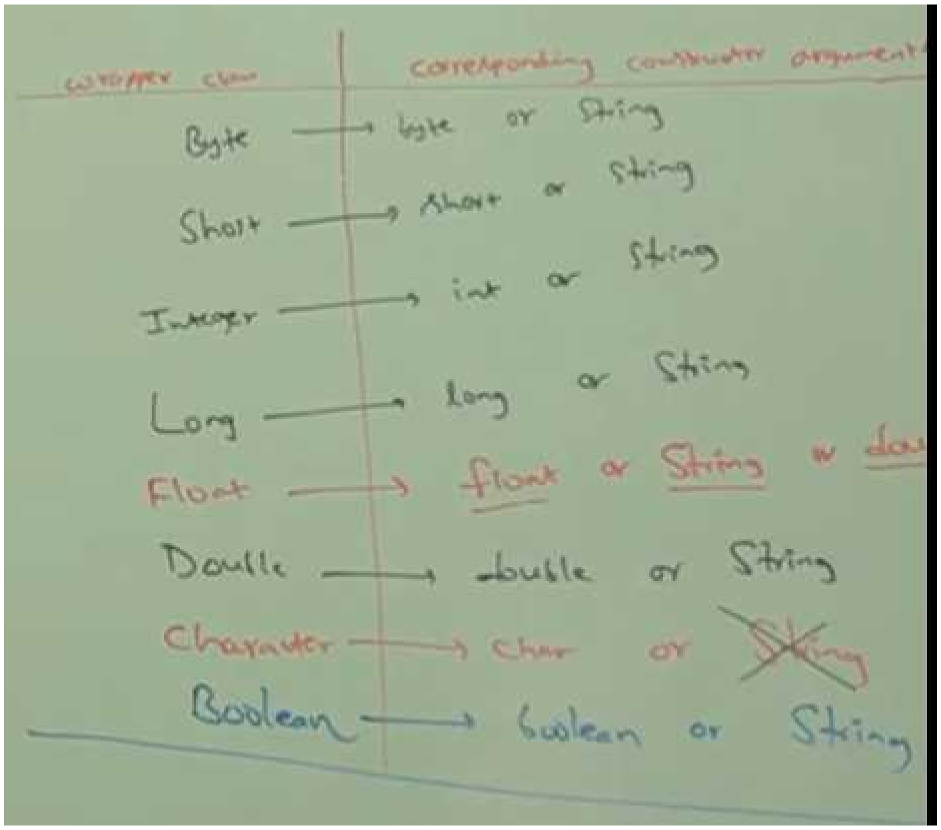


```
Integer I = new Integer("ten");
```

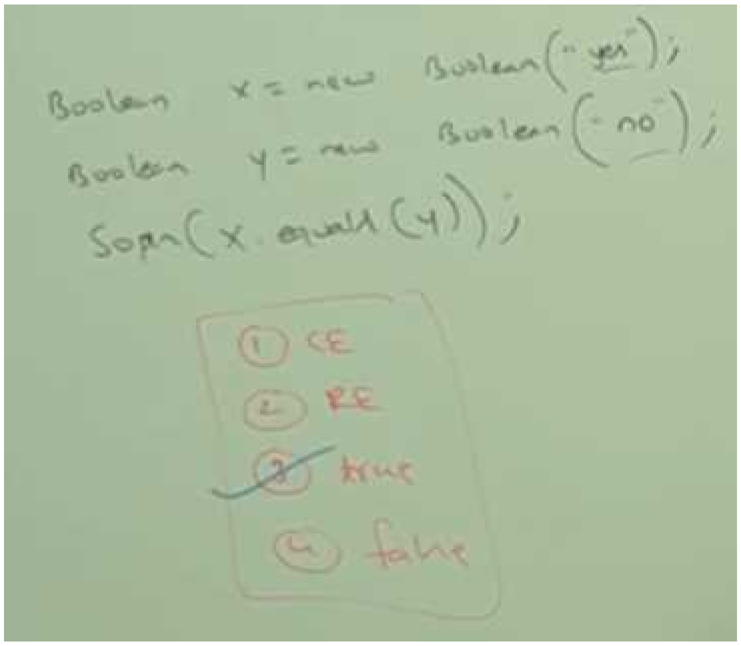
RE: NumberFormatException



```
Float f = new Float(10.5f);
Float f = new Float("10.5f");
Float f = new Float(10.5);
Float f = new Float("10.5");
```



wrapper class	corresponding constructor argument
Byte	byte or String
Short	short or String
Integer	int or String
Long	long or String
Float	float or String or do
Double	double or String
Character	char or ~~String~~
Boolean	boolean or String

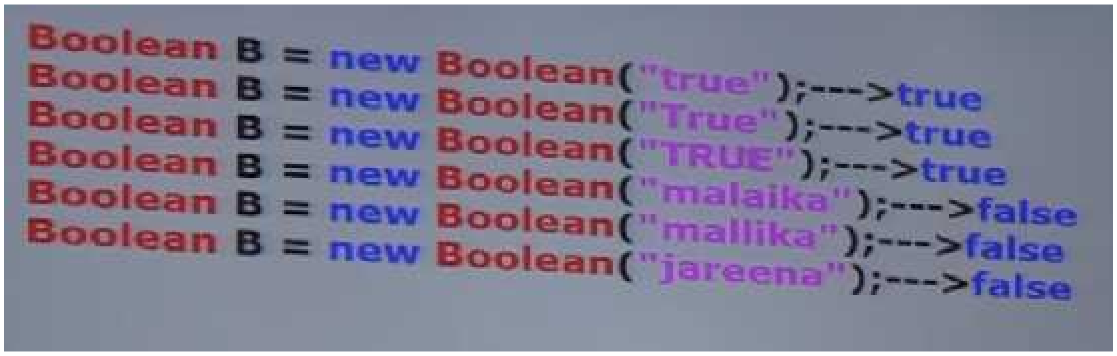


```
Boolean x = new Boolean("yes");
Boolean y = new Boolean("no");
Sopln(x.equals(y));
```

1. CE
2. RE
3. [✓ true]
4. false

If we are passing String type as an argument then case and content both are not inportant,.

If the content is case insentive string of 'true', then it is treated as true, otherwise it is treated as false.



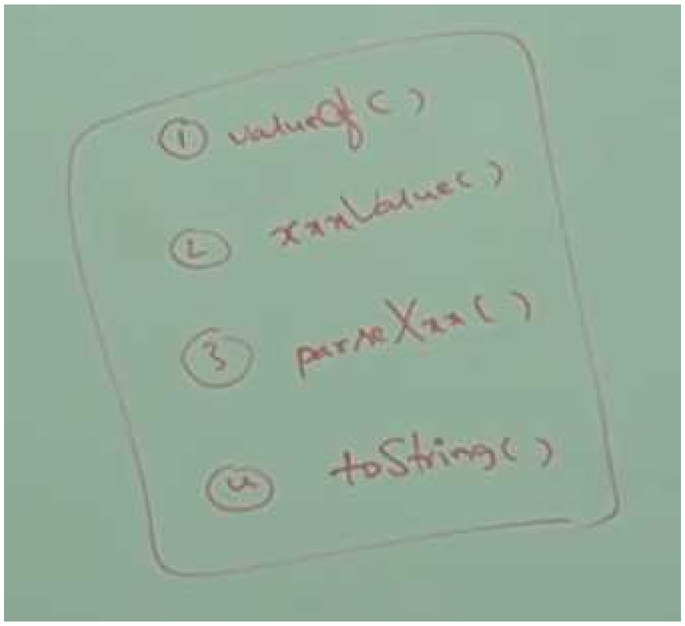
```
Boolean B = new Boolean("true");--->true
Boolean B = new Boolean("True");--->true
Boolean B = new Boolean("TRUE");--->true
Boolean B = new Boolean("malaika");--->false
Boolean B = new Boolean("mallika");--->false
Boolean B = new Boolean("jareena");--->false
```

Note:

In all wrapper classes toString(), method is overriden to return content directly.

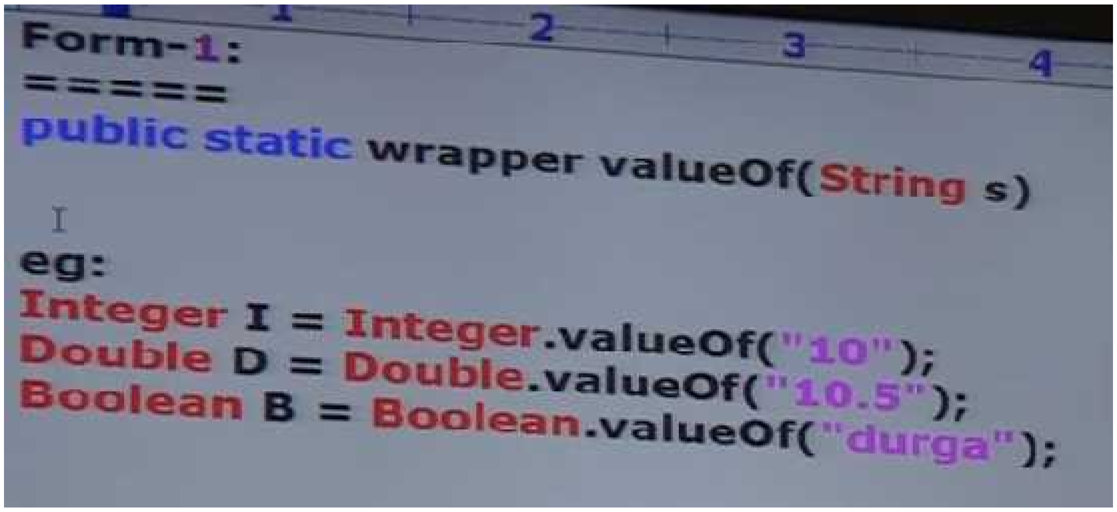
In all warapper classes .equals(), method is overriden for content comparision.

Utility Methods:



1. valueOf( )
2. xxxValue( )
3. parseXxx( )
4. toString( )

We can use valuOf(), methods to create wrapper object for the given primitive or string.



```
Form-1:
=====
public static wrapper valueOf(String s)

eg:
Integer I = Integer.valueOf("10");
Double D = Double.valueOf("10.5");
Boolean B = Boolean.valueOf("durga");
```