Java.lang.Package

The most commonly required classes and interface which are required for casting any java program whether it is simple or complex are encapsulated into a separate package which is nothing but lang package.

It is not required to import lang package explicitly because by default it is available to every java program.

- **The following are some of the commonly used classes in lang package…….**

  1) object

  2) String

  3) String Builder

  4) String Buffer

  5) wrapper classes(auto boxing and autounboxing)

  1) Object:-

  - The most common methods which are required for any java object are encapsulated into a separate class which is nothing but object class.
  - Some people made this class as parent for all java classes so that its methods are by default available to every java class automatically.
  - Every class in java is the child class of object either directly or indirectly, if our class want to extends any other class then only our class is direct child class of object.

  Ex.

  Class A

  {…………………..

  …………………..

  }

- If our class extends any other class then our class is not direct child class of object. If extends object class indirectly.

Ex.

```
class A extends B
{
………………….
………………….
}
```

- Object class define the following 11 methods

1)public string tostring()

2)public native int hashcode()

3) public Boolean equals(object o)

4)protected native object code() throws CodeNotSupportException

5)public final class getcalss()

6)protected void finalize() throws Throwable

7)public final void wait() throws Interrupted Exception

8)public final native void wait(long ms) throws IE

9)public final void wait(long ms,int ns) Throws IE

10)public final native void notify();

11)public final native void notifyAll()

1)**tostring() method:-**

- we can use this method to find string separations of an object
- Whenever we are trying to print any object reference internally 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.m(String args[])

{

Student s1=new Student("Ravi",111);

Student s2=new Student("Rocky",101);

s.o.pln(s1); => s.o.pln(s1.toString()); Student @3e25a5

s.o.pln(s2);        Student@19821f

}

}
```

- In the above case class tostring() method got created which is implemented as fallows.

```
Public String toString()

{

return getclass().getName + "@" +Integer.tohexString(hashcode());

}
```

- classname@hexadecimal String representation of hash code
- To provide our own String representation we have to override toString() in our class which is highly recommended.
- Whenever we are trying to print student object reference to return his name and roll number we have to override toString() as follows

```
public String toString()

{

return name;

return name+"….."+rollno;

return "This is student with name="+name+",with rollno:"+rollno;

}
```

- In string StringBuffer and in an wrapper classes toString method is overridden to return proper string form. Hence,it is highly recommended to override toString() method in our class also…..

```
class Test

{

public string toString()
{
Return “test”;
}
Public static void main(String args[])
{
Test t=new Test();
String s=new String(“durga”);
Integer i=new Integer(10);
s.o.pln(t) ;   //test
s.o.pln(s);   //durga
s.o.pln(i);     //10
}
}
```

(2)hashcode():-

- for every object Jvm will assign one unique id which is nothing but hashcode
- Jvm use hashcode while saving objects into hashtable or hashSet or hashmap.
- Based on our requirement we can generate hashcode by overriding hashcode method in our class.
- If we are not overriding hashcode() method method then object class hashcode() method will be executed which generates hashcode based on address of the object but whenever we are overriding
- Hashcode() method,Then hashcode is no longer related to address of the object.
- overriding hashcode() method is said to be proper iff for every object we have to generate a unique number.

Ex.(1)case1:-

```
class student
{
……

…….
…….
public int hashcode()
{
return 100;
}
}
```

case(2):-

```
class Student
{
……….

……….
………..
        public int hashcode()
{
        return rollno;
}
}
```

Case 1:- It is improper way of overriding hashcode() because we are generating some hashcode for every object.

Case 2:-It is proper way to overriding hashcode() because we are generating a different hashcode for every object.

toString vs hashcode:-

```
ex. Class Test
{
int  I;
```

```
class Test
{
        int I;
Test(int i)
{
This.i=I;
}
p.s.v.m(……..)
{
Test t1=new Test(10);
Test t2=new Test(20);
s.o.pln(t1);  Test @1a3b2b
s.o.pln(t2); Test @2a4b2a
}
}
```

```
Test (int i)
{
        this.i=I;
}
        public int hashcode()
{
return i;
        }
        p.s.v.m(………)
        {
        Test t1=new Test(10);
        Test t2=new Test(20);
s.o.pln(t1);   Test @ a
s.o.pln(t2);  Test @64
}
}
```

Object->toString()

Object->hashcode()

object->toString()

Test->hashcode()

0-15

0

1

2

1

```
1
9
A(10)
B(11)
C(12)
D(13)
E(14)
F(15)
```

Ex3:-

```
class Test
{
int  I;
Test(int i)
{
This.i=i;
}
Public int hashcode()
{
return i;
}
public string tostring()
{
return i+" ";
}
p.s.v.m(…………….)
{
Test t1=new Test(10);
Test t2=new Test(100);
s.o.pln(t1);   //10
s.o.pln(t2);   //100
}
}
Test->toString()
```

NOTE:-

- If we are giving opportunity to object class toString() method than it will call internally hashcode() method
- If we are giving opportunity to over class toString() method than it may not call hashcode() method

(3) equals() method:-

-> we can use equals() method to check equality of two objects

Public Boolean equals(object o)

Ex.

```
class Student
{
String name;
int rollno;
Student(String name,int rollno)
{
this.name=name;
this.rollno=rollno;
}p.s.v.m(……………)
{
Student s1=new Student("durga",101);
Student s2=new Student("Ravi",102);
Student s3=new Student("durga",101);
Student s4=s1;
s.o.pln(s1.equals(s2));    //false
s.o.pln(s1.equals(s3));  //false
s.o.pln(s1.equals(s4));   //true
}}
```

- In the above case object class equals() method will be executed which is always must for reference comparison(Address comparison),

that is , if two reference pointing to the same object then only equals() method returns true. This behavior is exactly same as==operator

- If we want to perform content comparison instead of reference compression we have to override equals() method in our classes.
- Whenever we are overriding equals() method we have to consider the following things,

  (1) What is the meaning of equality

  (2) In the case of different type of objects(Hetrogeneous) equals method should return false but not ClassCastException

  (3)If we are passing Null arrangement our equals method should returns false but not a NullPointerException.

  - The following is the valid way of overriding equals() method in

Ex.

```
 Class Student
{
public Boolean equals(object o)
{
        Try
        {
                String name1=this.name;
                int rollno1=this.rollno;
                Student s2=(Student)c;
                Student name2=s2.name;
```

```
int rollno2=s2.rollno;

isf(name1.equals(name2) && rollno1==rollno2)

{

return true;

}

else

{

return false;

}

Catch(ClassCasteException  e)

{

return false;

}

Catch(NullPointerException e)

{

return false;

}

Student s1=new Student("durga",101);

Student s2=new Student("pavan",102);

Student s3=new Student("durga",101);

Student s4=s1;

s.o.pln(s1.equals(s2));          //false

s.o.pln(s1.equals(s3));           //true
```

```
                s.o.pln(s1.equals(s4));            //true
                s.o.pln(s1.equals("durga"));  //false
                s.o.pln(s1.equals(101));    /false
```

short way of waiting equals method()

```
public Boolean equals(object o)
{
    Try
    {
    Student s2=(Student) o;
if(name.equals(s2.name) && rollno==s2.rollno)
return true;
else
return false;
Catch(ClassCasteException e)
{
return false;
}
Catch(ClassCasteException e)
{
return false
}
  }
}
```

Relationship between==operators and .equals() method:-

- ➔ If s1==s2 is true, then s1.equals(s2) is always True
- ➔ If s1==s2 is false, then we can't expert about s1.equals(s2) Exactly. It may returns true or false
- ➔ If s1.equals(s2) return true, we can't console anything about s1==s2, it may returns either true or false
- ➔ If s1.equals(s2) is false, then s1==s2 is always false

Difference between == operators and .equals() method:-

== operator
.equals()

(1) It is an operator applicable for both premitives (1) It is a method applicable only for object reference but
And object reference not for premitives

(2) In the case of object reference == operator is (2) Bydefault .equals() method present in object class is
Is always meant for reference comparision i.e also meant for reference comparision only.

If two reference pointing to the same object

Then only ==operator return t

(3)We can't override ==operator for content (3) We can override .equals() method for

content

Comparision comparision

(4)In the case of heterogeneous type object == (4)In the case of hetrogenous objects

.equals() method

Operator causes compile time error saying simply return false and we can't get any

Incomparable types compile time or runtime error

(5) For any object reference or r==null is (5)for any object reference r,r.equals(null)

Always false is always false

Note:-

Q) What is the difference between Double equal operator(==) and .equals()?

--> '==' operator is always meant for reference comparison, where as .equals() method meant for content

Comparisons.

Ex.

String s1=new String("durga");

String s2=new String("durga");

s.o.pln(s1==s2); //false

s.o.pln(s1.equals(s2)); //true

- In String .equals() is override for content comparison.
- In String Buffer class .equals() is not overridden for contents comparison hence object class .equals() got Executed which is meant for reference comparison.
- In wrapper class .equals() is overridden for content comparison.

Contract between .equals() and hashcode();-

1.If two objects are equal by .equals() compulsory their hashcode must be same.

2.If two objects are not equal by .equals() then there are no restructions on hashcode(), can be

Same Or different.

3.If hashcodes of 2 objects are equal. Then we can't hashcode above .equals(). It may be

returns true or False.

4. If hashcode and a objects are not equals then we can always .equals() returns false

Conclusion:-

- To satisfy the above contract between .equals() and hashcode(), whenever we are

overriding .equals()

compulsory we should override hashcode().

- If we are not overriding we can't get any compile time and run-time error.

➔ But it is not a good program practice.

Q) Consider the following .equals()

```
public Boolean equals(object obj)
{
if(!(obj instance of the person))
{
return false;
}
Person p=(person) obj;
if(name.equals(p.name)&&(age==p.age))
{
```

```
return true;
else
return false;
}
```

1)Which of the following hashcode() are said to be properly implemented.

```
(1) public int hashcode()
    {
    return 100;                    //Wrong
    }

(2) public int hashcode()
    {
    return age +(int)height;     //wrong
    }
(3) public int hashcode()
     {
     return name.hashcode() +age;  //write
     }
(4) public int hashcode()
    {
    return (int)height;        /wrong
    }
(5) public int hashcode()
    {
    return age +name.lenght();
    }
```

Note:-

To Maintain a contract between .equals() and hashcode() what ever the parameters we are using while over siding .equals() we have to use the same parameters while overriding hashcode() also

Clone():-

- ➔ The process of creating exactly duplicate is called cloning
- ➔ The main objective of cloning is to maintain backup

We can get cloned object by using clone() of objects class protected native object clone() throws cloneNotSupportException.

```
class Test implements cloneable
{
int i=10;
Int j=20;
p.s.v.m(String args[]) throws CloneNotSupportException
{
Test t1=new Test();
Test t2=(Test)t1.Clone();
T2.i=888;
T2.j=999;
s.o.pln(t1.i+ "    "+t1.j);
}
}
s.o.pln(t1.hashcode()==t2.hashcode());        //false
s.o.pln(t1==t2)                 //false
```

-> we can call clone() only on cloneable objects.

- An object is said to clonable if the corresponding class implements cloneable interface. Cloneable interface. Cloneable interface presently java.lang.package and doesn’t contain any methods. It is a marker interface.

<u>Deep Cloning and Shallow Cloning:-</u>

- ➔ The process of creating just duplicate reference variable but not duplicate object is called shallow cloning.
- ➔ The process of creating exactly duplicate independents object is bydefault considered as deep cloning.

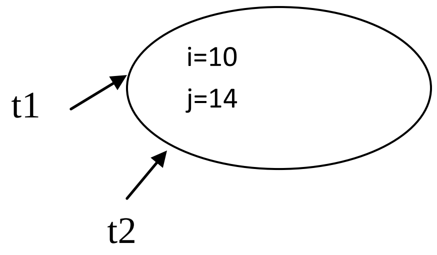
Ex.

Test t1=new Test();

Test t2=t1; //shallow cloning

Test t3=(test)t1..clone(); //deep cloning shallow cloning

- Bydefault cloning means deep cloning



String class

Case(1):-

Immutable Mutable

String s=new String("durga"); SB s=new SB("durga");
s.concate("software"); s.append("software");
s.o.p(s); //durga s.o.pln(s);
/durgasoftware

->One we created a string object we can't ->one we created a StringBuffer
perform any changes in the existing object. object we can Perform any changes
If we are trying to perform any changes in the existing object.This behaviour
with those changes a new object with be created. Is nothing but "mutability of string
This behavior is nothing but" immutability of buffer object"
String object".

getclass():-

This method returns run-time class definition of an object

Ex. Test ob=new Test();

s.o.pln("class name:" +object=class().getname());

Case(2):-

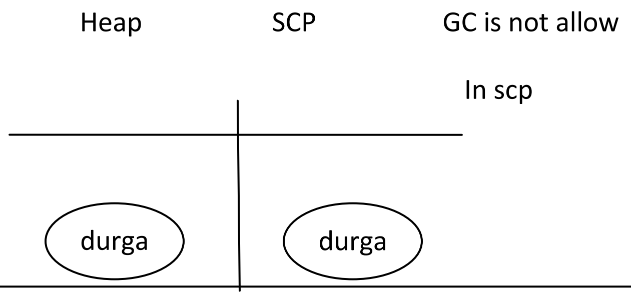
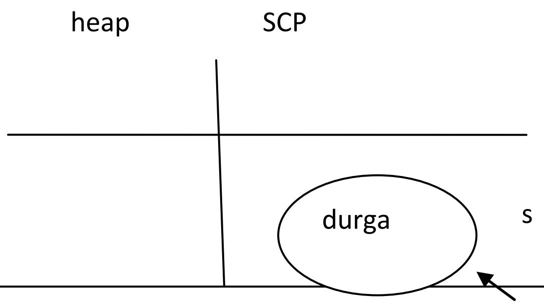
1)String s1=new String("durga");
 	
String s2=new String("durga");
s.o.pln(s1==s2) //false
s.o.pln(s1.equals(s2)); //true
	1)StringBuffer sb1=new StringBuffer("durga");
	SB sb2=new SB("durga");
	s.o.pln(sb1==sb2);//false
	s.o.pln(sb1.equals(sb2)); //false

2) In stringclass.equals() method is Overridden for content comparison. Hence .equals() method returns true if Content is same even though object are different.	2) In stringBuffer class .equals()method is not Overridden for content comparision. executed which is Hence object class .equals() method will be ment for reference comparison due to this .equals() Method returns false eventhough content is same If objects we different.

Case(3):-

Q) What is the difference between following ?

Ex(1).

String s=new String("durga")	String s="durga";
->In this case two object will be created one is in heap, and the other is in scp and 's 'is always pointing to Heap object 	-> In this case only one object will be created in scp and 's' is always pointing to that object 

Note:-

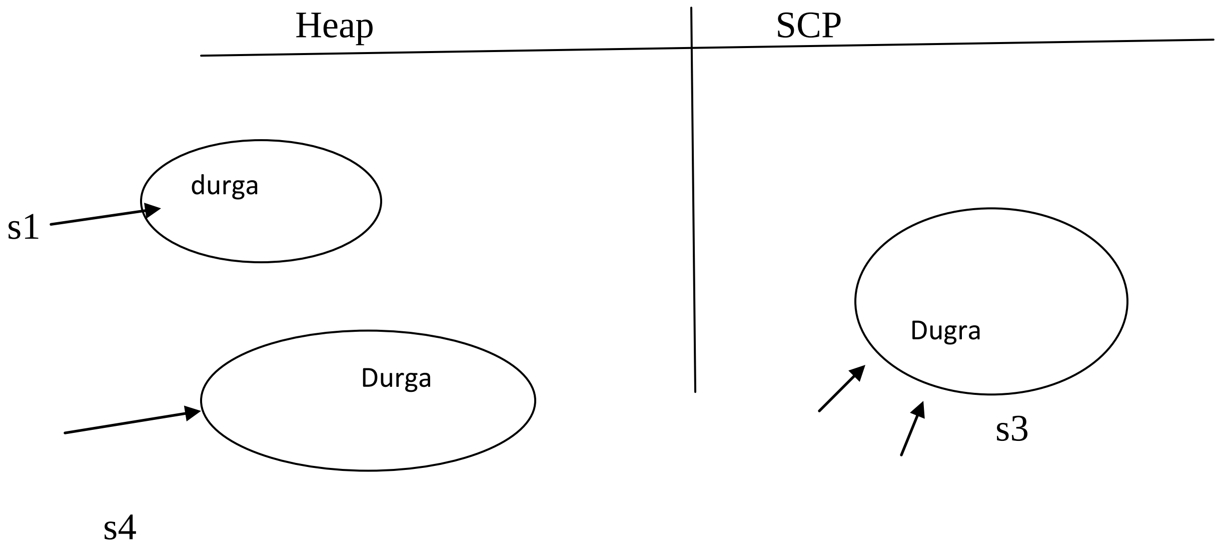
(1) G.C is not allowed to access in scp area hence eventhough object doesn't have any reference variable still it is not eligible for G.C and it is present in scp area.

(2) All objects present on scp will be destroyed automatically at the time of JVm shutdown.

(3) Object creation in scp is always options. First Jvm check is any object already present in scp with required content or not. If it is already available then it will reuse existing object instead of creating new object. If it is not already available then only a new object will be created. Hence , there is no chance of two object with the same content in scp. i.e Duplicate, object are not allowed in scp.

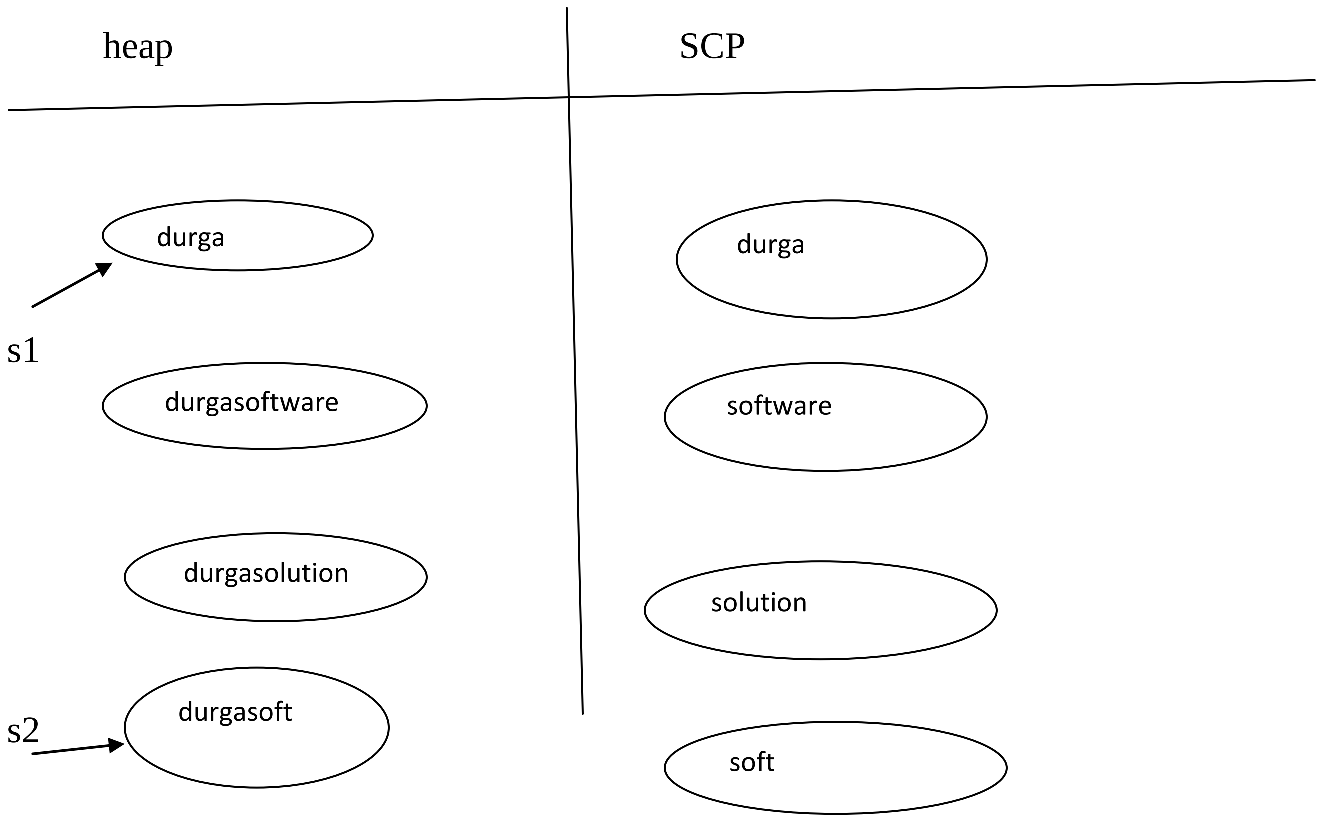
Ex(2).

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



ex(3).

```
String s1=new String("durga");
S1.concate("software");
S1.concate("solutions");
String s2=new s1.concate("soft");
```



Note:-

->For every String constant compulsory one object will be created in scp area.

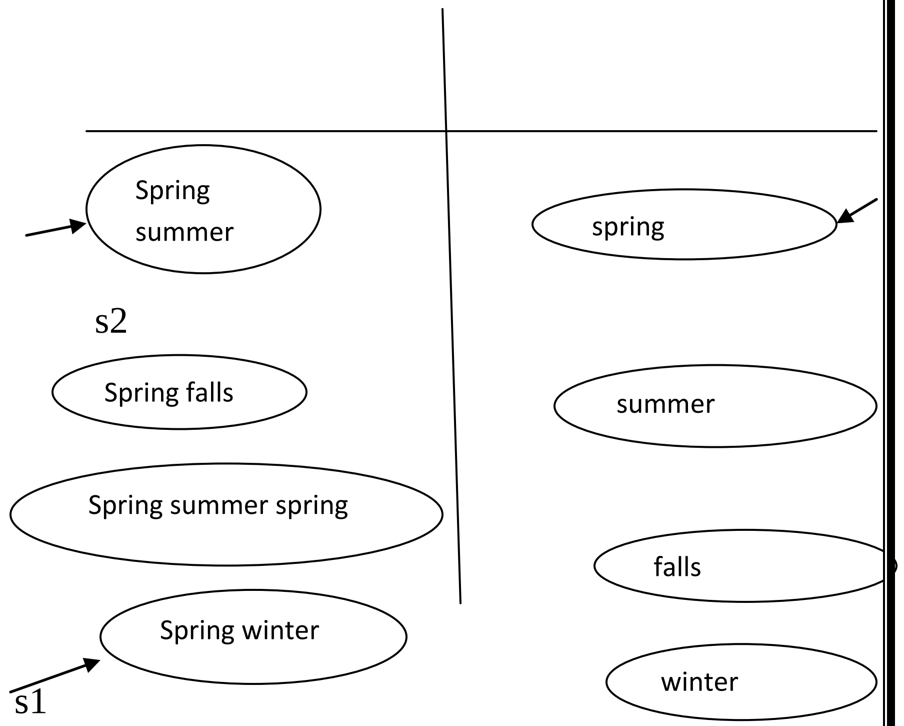
-> Because of same runtime operation if an object is required to created that object should be created only on heap but not in scp.

Ex. String s=”durga”+new String(“durga”);

SCP

Ex3.

```
String s1=”spring”;
String s2=s1+”summer”;
   s1
S1.concate(“falls”);
S3.concate(s1);
S1+=”winter”;
s.o.pln(s1);
s.o.pln(s2);
```



ex. Note:-

final string s=”raghu”; s is a constant

string s=”raghu”; s is a normal variable

```
String s1=new string("you cannot change me!");
String  s2=new String("you can not change me!");
s.o.pln(s1==s2)   //false
String s3="you cannot change me!";
String s4="you canot change me!";
s.o.pln(s1==s4);    //true
s.o.pln(s1==s3);    //false
String  s5="you cannot"+"change me!";
s.o.pln(s3==s5);   //true
String s6="you cannot";
String s7=s6+"change me!";
s.o.pln(s3==s7);  //false
final string s8="you cannot";
string s9=s8+"change me!";
s.o.pln(s3==s9);   //true
s.o.pln(s6==s3);   //true
```

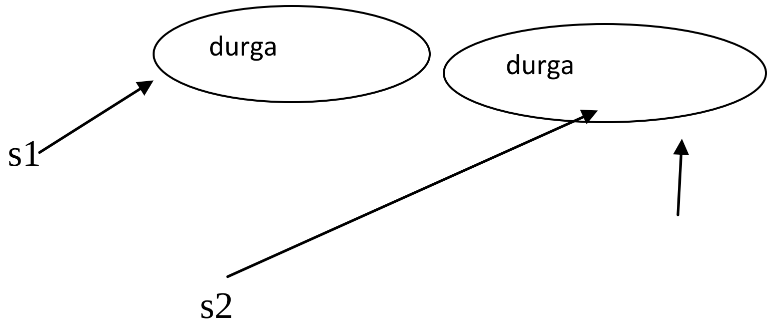
<u>Interning And String:-</u>

➔ By using heap object reference if you want to get corresponding scp object reference then we shold go for intern()

Ex.-

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

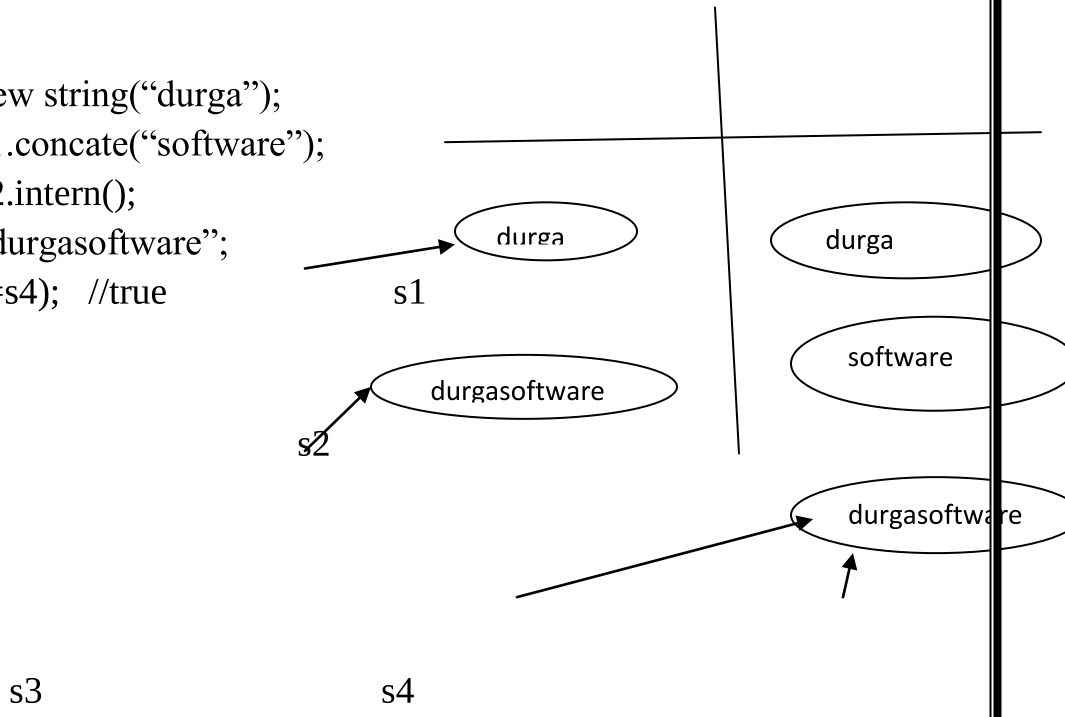
String 2=s1.intern();
s.o.pln(s1==s2); //false
string s3=”durga”;
s.o.pln(s2==s3); //true



➔ If the corresponding object is not available in scp, then intern() creates that object and return it.

Ex.

```
String s1=new string(“durga”);
String s2=s1.concate(“software”);
String s3=s2.intern();
String s4=”durgasoftware”;
s.o.pln(s3==s4); //true
```



Constructors of the string class:-

(1) String s=new string();
(2) String s=new string(string constant);
(3) String s=new string(stringBuffer sb);
(4) String s=new string(char[] ch);

Ex:-

```
Char[] ch={‘a’,’b’,’c’,’d’};
String s=new string(ch);
s.o.pln(s);   //abcd
```

(5) String s=new string(byte[] b)

Eg:-

```
byte[] b={100,101,102,103};

string s=new string(b);

s.o.pln(s);
```

<u>Important methods of string class:-</u>

(1) public char charAt(int index);

```
Eg:- string s=”durga”;
s.o.pln(s.charAt[3]);    //g
s.o.pln(s.charAt[30]);      //R.E:-StringIndexOutOfBoundsException
```

(2) public string concat(string s)

Ex.

```
String s=”durga”;

S=s.concate(“software”);

//s=s+”software”;

//s+=”software”;

s.o.pln(s);     //gurgasoftware
```

-> The overloaded +,+=operators also ment for concatenation only

(3) <u>public Boolean equals(object obj)</u> ment for content comparison where the case is also important.

(4) <u>Public Boolean equalIgnoreCase(String s)</u> meant for content comparison where the case is not important.

<u>Ex.</u>

String s=”Java”;

s.o.pln(s.equals(“java”)); //false

s.o.pln(s.equalsIgnoreCase(“java”)); //true

NOTE:-

In general to perfom validation of username we have to go for equalsIgnoreCase method where the case is not important. Where as to perform password validation we have to else quals() where the case is important.

(5) Public string substring(int begin); reference the substring form begin index to end of the string.
(6) Public string substring(int begin,int end); returns the substring form begi index to end index.

Ex. String s=”abcdefg”;

s.o.pln(s.substring(3))// defg

s.o.pln(s.substring(2.6))// cdef

(7) public int length();

Eg. String s=”aabbb”;

s.o.pln(s.length); ->C.E: can’t find symbol

Symbol : variable length

Location: class java.lang.string

Note:-
Length variable applicable for arrays where as length() is applicable for string objects

(8)public string replace(char old, char new);

Eg. String s=”abbb”;

s.o.pln(s.replace(‘a’,’b’));

(9) public string tolowercase();

(10)public string touppercase();

(11)<u>public string trim():-</u>

->To remove the bank space present at beginning and end of the string bu not blankspace present at middle of the string.

(12) <u>public int indexOf(char ch):-</u>

-> If returns indexOf first occurance of the specified character

(13) public int l

astIndexOf(char ch);

Importance of the string constant pool(scp):-

Vector registration form

Name of the consistency:chpt

Name:Ravindra

Fathername:Mohan

Age:25

DOB:

H.No:9-133

Strect:Ramnagar

Substrect:ramnagar

City:pune

Dist:pune

State:maharastra

Country:India

PIN:1234

Identifacation name:****

- ➔ In our program if any string object required to use separately, it is not recommended to create a separate object for every requirement. This approach reduces performed & memory utilization.
- ➔ We can resolve this problem by creating only one object and share the same object with all required references.

- ➔ This approach improve memory utilization and performance we can achive this by using string constant pool.
- ➔ In scp, a single object will be shared for all required reference. Hence the main advantages of scp are memory utilization and performances will be improved,
- ➔ But the problem in this approach is, as several references pointing to the same object by using one reference. If we are perform any change all remaining references will be improved.
- ➔ To resolve there sum people declare string objects as immutable().
- ➔ According to that once we creted a string object we can't perform any change in the existing object. If we are trying to perform any change with. So, that there is no effect on remaining references.
- ➔ Hence, "The main disadvantages of scp is we should compulsory maintain String object as immutable".

Q) **Why scp like concept is defined only for string object but not for StringBuffer.?**

-> In any java program,the most commonly used on=bject is string. Hence with repeate to memory and performances special arrangement is required, for this scp concept required.

-> But StringBuffer is commonly used object. Hence special concepts like scp is not required.

Q) **What are the advantages of scp?**

->Instead of creating a separate object for every requirement we can create only one object in scp and we can reuse the same object for every requirement. So that performance and memory utilization will be increased.

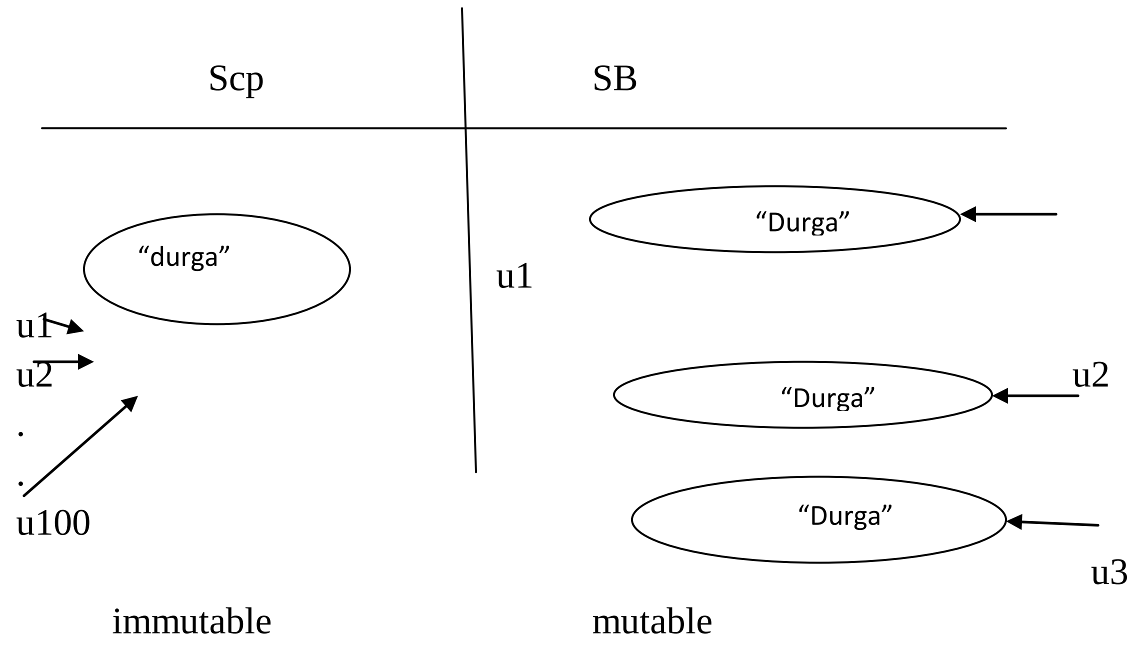
Q) **What is the disadvantages of scp?**

-> compulsory we should make string object as immutable.

Q)**Why string objects are immutable where as stringBuffer object are mutable?**

-> In this case of string several reference can pointing to the same object. By using one reference,if we are performing any change in the existing object the remaining reference will be impacted to resolve this problem sun people declared as string objects are immutable . According to this one we created a

string object we can’t perform any changes in the existing object. If we are trying to perform any changes with those changes a new object is created i.e scp is the only reason why the string object are immutable.
-> But in case of stringBuffer for every requirement compulsory a seprate object will be created. Revising tfhe same Stringbuffer object. There is no changes In one stringbuffer object if we are performing any changes there is no impact of remaining references. Hence we can perform any changes in the stringbuffer object and stringbuffer objects are mutable…..



Q)**It is possible to create our own immutable class?**
->Yes
Note:-
->Once we created a string object we can’t perform any changes in the existing object. If we are trying to perform any changes with those changes a new object will be created on the heap.

-> Because of our runtime method call if there is a change in content then only new object will be created.

-> If there is no change in content existing object only will be reused

Ex.(1)…

String s1=”durga”;

String s2=s1.touppercase();

String s3=s1.tolowercase();

s1

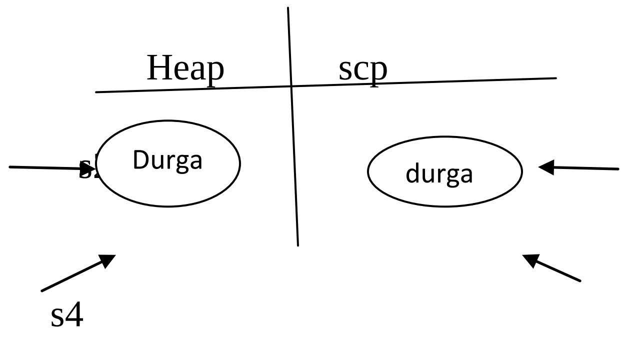
String s4=s2.touppercase();

s.o.pln(s1==s2) //false

s3

s.o.pln(s1==s3) //true

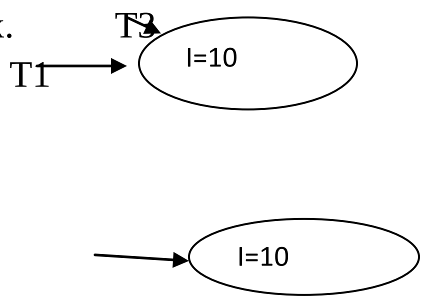
s.o.pln(s2==s4) //true



Creation of our own Immutable class:-

- We can create our own immutable class also
- Once we created an object we can’t perform any change in the existing object. If we are trying perform any change with those changes a new object will be created.
- Because of our runtime method call if these is no change in the content then existing object onky will be returned

Ex.



```
Ex.    Final class Test
        {
        private int I;
        Test(int i)
        {
        This.i=I;
        }
        public test modify(int i)
        {
        if(this.i==i)
        return this;
        return(new Test(i));
        }
        }
        Test t1=new Test(10);
        Test t2=new Test(100);
        Test t3=new Test(10);
        s.o.pln(t1==t2);    //false
        s.o.pln(t1==t3);   //true
```

Q) In java which object are immutable?

(1) string object

(2) All wrapper object are immutable

StringBuffer:-

- ➔ If the content will change frequently then it is never recommended to go for string. Because for every changes compulsory a new object will be created.
- ➔ To handle this required commpulsary we should go for stringbuffer where all changes will be performed in existing object only instead of creating new object.

Constructor:-

(1) Stringbuffer sb=new stringbuffer();

- ➔ Create an empty stringbuffer object with default initialcapacity
- ➔ Once stringbuffer reaches its max capacity a new sb object will be created with

  New capacity=(current capacity+1)*2

Ex.

StringBuffer sb=new StringBuffer();

s.o.pln(sb.capacity()); //16

sb:append("abcdefghijklmnop");

s.o.pln(sb.capacity()); //16

sb.append("g");

s.o.pln(sb.capacity()); //34

(2) StringBuffer sb=new stringBuffer(int initialcapacity);

-> create an empty sb object with specified initilcapacity

(3)StringBuffer sb=new StrinBuffer(string s);

-> create an equivalent SB object for the given string with capacity=16+s.lenght();

<u>Important methods & StringBuffer class:-</u>

(1) Public int length()
(2) Public int capacity()
(3) Public class charAt(int index);

Eg. StringBuffer sb=new StringBuffer("durga");

s.o.pln(charAt(3));
s.o.pln(charAt(30)); //Reason:- StringIndexoutOfBoundException

s.o.pln(charAt(5)); ////Reason:- StringIndexoutOfBoundException

(4) public void setcharAt(int index,char ch);

-> To replace the character locating of specified index with the provided character.

(5) public StringBuffer append(String s);

append(int i);

append(Boolean b); // this and another two methods is overloaded method

append(double d); //overload

append(object o); //overload

ex. StringBuffer sb=new StringBuffer();

sb.append("pi value is");

sb.append(3.14);

sb.append("It is exactly");

sb.append(true);

s.o.pln(sb);

(6) public stringbuffer insert(int index,String s);

(int index, int i);

(" boolean b);

(" double d);

Ex.

StringBuffer sb=new StringBuffer("durga");

Sb.insert(3,"ravi");

s.o.pln(sb); //durgaravi

(7) public stringbuffer delete(int begin,int end);

-> To delete the characters present at begin index to end-1 index

(8) public StringBuffer deletecharAt(int index);

->To delete the character locating at specified index

(9) public StringBuffer reverse()

Eg. Sb=new SB("durga");

s.o.pln(sb.reverse()); //agrud

(10)public void setlength(int length)

Ex. StringBuffer sb=new StringBuffer("durga12345");

Sb.length(8);

s.o.pln(sb); //durga123

(11) public void ensure(int capacity);

-> To get the capacity based on ourrequirement

Ex. StringBuffer sb=new StringBuffer();

s.o.pln(sb.capacity());

sb.ensureCapacity(2000);

s.o.pln(sb.capacity()); //2000

(12) public void trimInSize()

-> To release extra allocated free memory after calling. This method length and capacity will be equal

Ex. StringBuffer sb=new StringBuffer();

Sb.encurecapacity(2000);

```
Sb.append("durga");

Sb.trimToSize();

s.o.pln(sb.capacity());    //5
```

StringBuffer:-

->Every method present in stringbuffer is synchronized. Hence at a time any one thread is allowed to access StringBuffr object. If Increases waiting time of the thread and effects performances of the system.

-> To resolve this problem SUN people introduced StringBuffer in 1.5 version.

-> StringBuffer is exactly same as StringBuffer(including method and constructor) except the following difference.

StringBuffer	StringBuilder
(1) Every method is synchronized	(1) No method is synchronized
(2) SB object is thread safe. Because SB safe because it can be object can be accessed by only one simultaneously thread of time	(2)String builder is not thread accessed by multiple thread
(3) Relatively performance is low high	(3) Relatively performance is
(4) Introduced in 1.0 version	(4) Introduced in 1.0 version

String vs StringBuffer vs StringBuilder:-

->if the content will not change frequently then we should go for String.

-> If the content will chnge frequently and thread Safety is required then we should go for StringBuffer.

-> If the content will change Frequently and threadsafety is not required then we should go for StringBuilder.

Method Chaining:-

->For most of the method in string, stringbuffer and stringbuilder the return type is same type onle. Hemce after applying a method an the result we can call another method will forms methodchaining

Sb.m1()…m2()….m3()….m4()…………..

->In mehod cahining all method will be executed form left to right.

Ex. Stringbuffer sb=new stringbuffer();

Sb.append("durga").insert(2,"xyz").reverse().delete(2,7).append("solutions";

s.o.pln(sb));

Final vs Immutable:-

-> If a reference variable declared as the final then we can't reassign that reference variable to some other object.

Ex.

Final StringBuffer sb=new StrinBuffer("durga");

Sb=new StringBuffer("software");

C.E:- can’t assign a value to final variable sb

-> Declaring a reference variable as final we want get any immutably,in the corresponding object we can performing any type of change eventhough reference variable declared as final.

Ex. Final StringBuffer sb=new StringBuffer(“durga”);

Sb.append(“software”);

s.o.pln(sb); durgasoftware

-> Hence final variable and immutability both concept are differect.

Wrapper classes:-
->The main objectives of wrapper classes are
(I)To wrap primitives into object form. So that we can handle primitive just like objects.
(II) To define several utility methods for the primitives

Constructor of wrapper classes:
Creation of wrapper object:-
->All most all wrapper classes contains two constructors. One can take corresponding primitives are arguments and the others can take string as arrangement.
Ex. Integer i=new Integer(10);
Integer i= new Integer(“10”);

Double d=new double(10.5);
Double d=new Double(“10.5”);

->If the string is not properly formatted then we will get R.E saying NumberFormstException

Ex. Integer i=new Integer("ten"); R.E! NFE

-> Float class contain 3 constructors one can take float primitives and the other can take string and 3rd one can take double argument.

Ex. 1) float f=new float(10.5f);
2) float f=new float("10.5f");
3) float f=new float(10.5); //double

- Character class contains only on constructor which can take char primitive as arguments.
  Ex. 1) character ch=new character('a');
  2) character ch=new character("a");

* Boolean class contains two constructors one can take Boolean primitive as the arguments and other can take string as arguments.

-> If we are passing Boolean primitive as arguments the only allowed values are true, false. By mistake if we providing any other we will get compile time error.

Ex.

1) Boolean B=new Boolean(true);
2) Boolean b=new Boolean(True);

->If we are passing string arguments to the Boolean constructors then the case is not important and content also not important.

-> If the content case insensitive string true, otherwise it is treated as false.

Ex.

1) Boolean B=new Boolean(true); //true
2) Boolean B=new Boolean("True"); //true
3) Boolean B=new Boolean("TRUE"); //true
4) Boolean B=new Boolean("durga"); //false
5) Boolean B=new Boolean("yes"); //false

Wrapper classes argument	**corresponding constructor**
Byte	byte as string
Short	Short as string
Integer	int as string
Long	long as string
*Float	float or string or double
Double	double or string
*character	char
*Boolean	Boolean or string

Q) which one is true and false?

(1) Boolean b1=new Boolean(“yes”);

(2)Boolean b2=new Boolean(“no”);

s.o.pln(b1.equals(b2)); -→ true

s.o.pln(b1==b2); → flase

s.o.pln(b1); false

s.o.pln(b2); false

Notes:-

-> In every wrapper class tostring() is overridden to return it’s contents

->In every wrapper class .equals() is overridden for content comparison

Utility Method:-

There are four methods

(I) valueOf()
(II) xxxvalue()
(III) parsevalue()
(IV) toString()

(I)valueOf:-

-> we can use value of method for creating wrapper object as alternate to constructor

Form 1:

->Every wrapper class exact character class contains a static valueOf() method for converting for converting string to the wrapper object.

public static wrapper valueOf(String s)

Eg. Integer I1=Integer.valueOf("10");

Boolean b1=Boolean.valueOf("true");

Double d=Doubl.valueOf("p.s");

Form(2):-

-> Every integer type wrapper class(Byte.shrt.Integer.lang) contains the following valueOf() method to convert specified Radix String form to corresponding wrapper object.

```
Public static wrapper valueOf(String s,int radix);
```

Ex.

Integer I1=Integer.valueOf("1010",2);

s.o.pln(I1); //10

Integer I2=Integer.valueOf("1111",2);

s.o.pln(I2);   //15

form(3):-

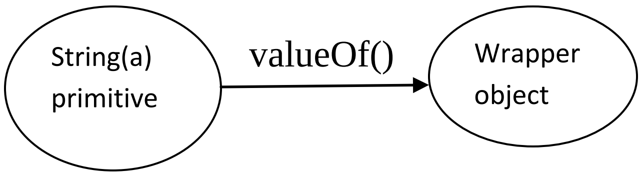
->every Wrapper class including character class contains the following valueOf() to convert primitives to corresponding wrapper object.

```
Public static wrapper valueOf(primitive p)
```

Ex.

1) Integer i=Integer.valueOf(10);
2) Character ch=character.valueOf('a');
3) Boolean B=Boolean.valueOf(true);

Note:



(II)xxxvalue();-

->We can use xxxvalue() methods to convert wrapper object to primitives.

->Every number type wrapper class contains the following size(6) xxxvalue() methods

->The methods are

Public byte byteValue();

Public int intValue();

Public short shortValue();

Public long longValue();

Public float floatValue();

Public double doublevalue();

Ex.

```
(1)Double d=new Double(130.456);
   s.o.pln(D.byteValue());     //-126
   s.o.pln(D.shortValue());    //130
   s.o.pln(D.intValue());       //130
   s.o.pln(D.longValue());    //130
   s.o.pln(D.floatValue());   //130.0
   s.o.pln(D.doubleValue());   //130.0
```

(a)charvalue():-

->character class contains char value method to convert character object to the char primitive.
Public char charValue();

Eg. Character ch=new character('@');
Char ch=ch.charValue();
s.o.pln(ch); // '@'

(b)BooleanValue():-
->Boolean class contains booleanValue to find boolean primitives for the given Boolean object.

```
Public Boolean booleanValue();
```

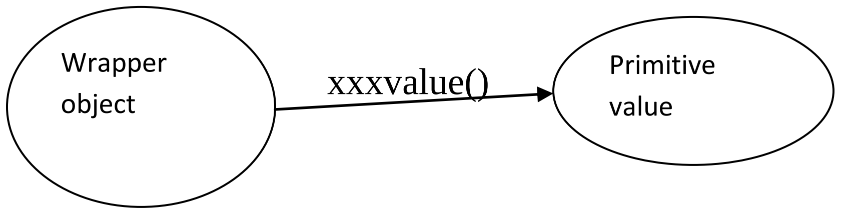
Eg. Boolean B=Boolean.valueOf("durga");

Boolean b=b.booleanValue();

s.o.pln(b); //false

Note:-

->Total 38=(6*6*+1+1) xxxValue() are variable



(III) parsexxx():-

->We can use parsexxx() to convert string to corresponding primitive

Form1:-

-> Every wrapper class except char class contains following parsexxx() to,convert string to corresponding primitive.

```
Public static primitive parseXXX(String s);
```

Eg.int i=Integer.parseInt("10");

Double d=Double.parseDouble("10.5");

Long l=long.parseLong("102");

Booleaan b=Boolean.parseBoolean("durga"); //false

Form2:-

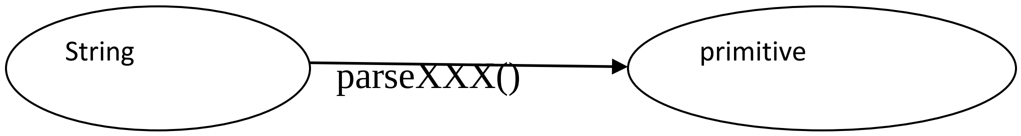
->every Integer types wrapper class contains the following parseXXX() to convert specified radix string to corresponging primitive.

Eg. Public static primitive parseXXX(String s,int radix);

Int i=Integer.parseInt("111",2);

s.o.pln(i); //15

note:-



(iv) toString:-

->we can use tostring() to convert wrapper object or primitive to string

Form(1):

->Every wrapper class contains the following tostring() to convert wrapper object to string type.

```
public string tostring();
```

-> It is overriding version of object class tostring()

Ex.

Integer I=new Integer(10);

s.o.pln(I.tostring());

form2:-

->Every wrapper class contains a static tostring(), to convert primitive to string form.

```
Public static tostring(primitive p);
```

String s=Integer.tostring(10);

String s=Boolean.toString(true);

Form (3):

->Integer and Long classes contains toString() to convert primitive to specified radix string form.

```
Public static string tostring(primitive p,int radix);
```

Eg. String s=Integer.tostring(15,2);
s.o.pln(3); //1111

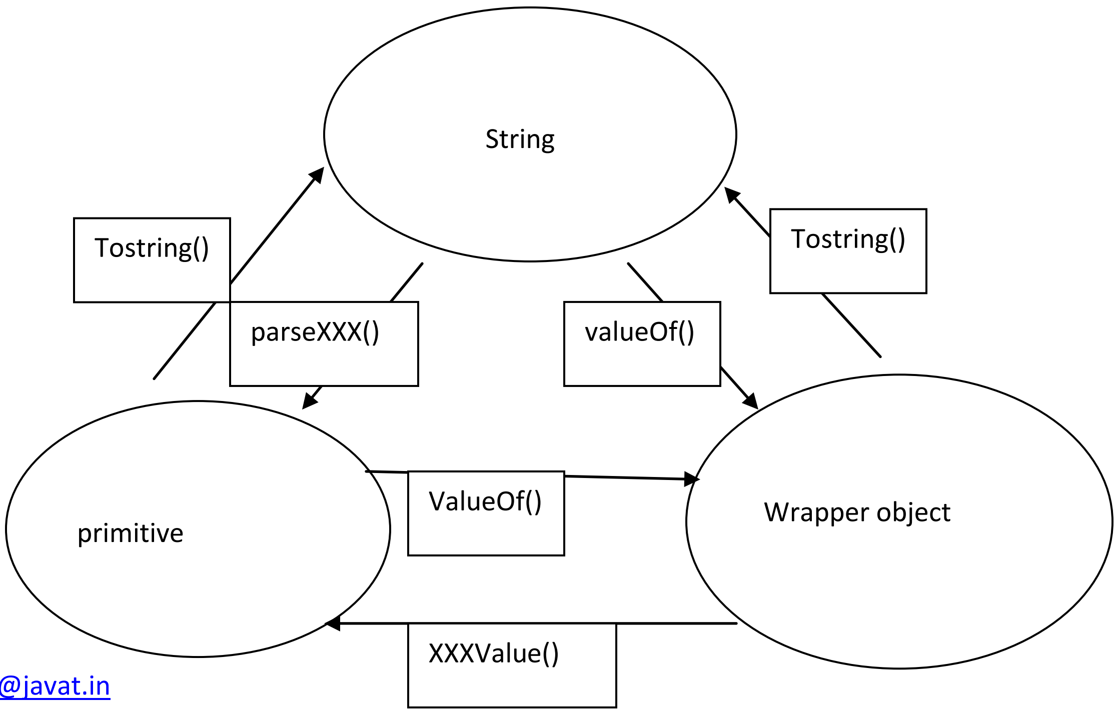
form 4:-

-> Integer and long classes contains the following toXXXString();

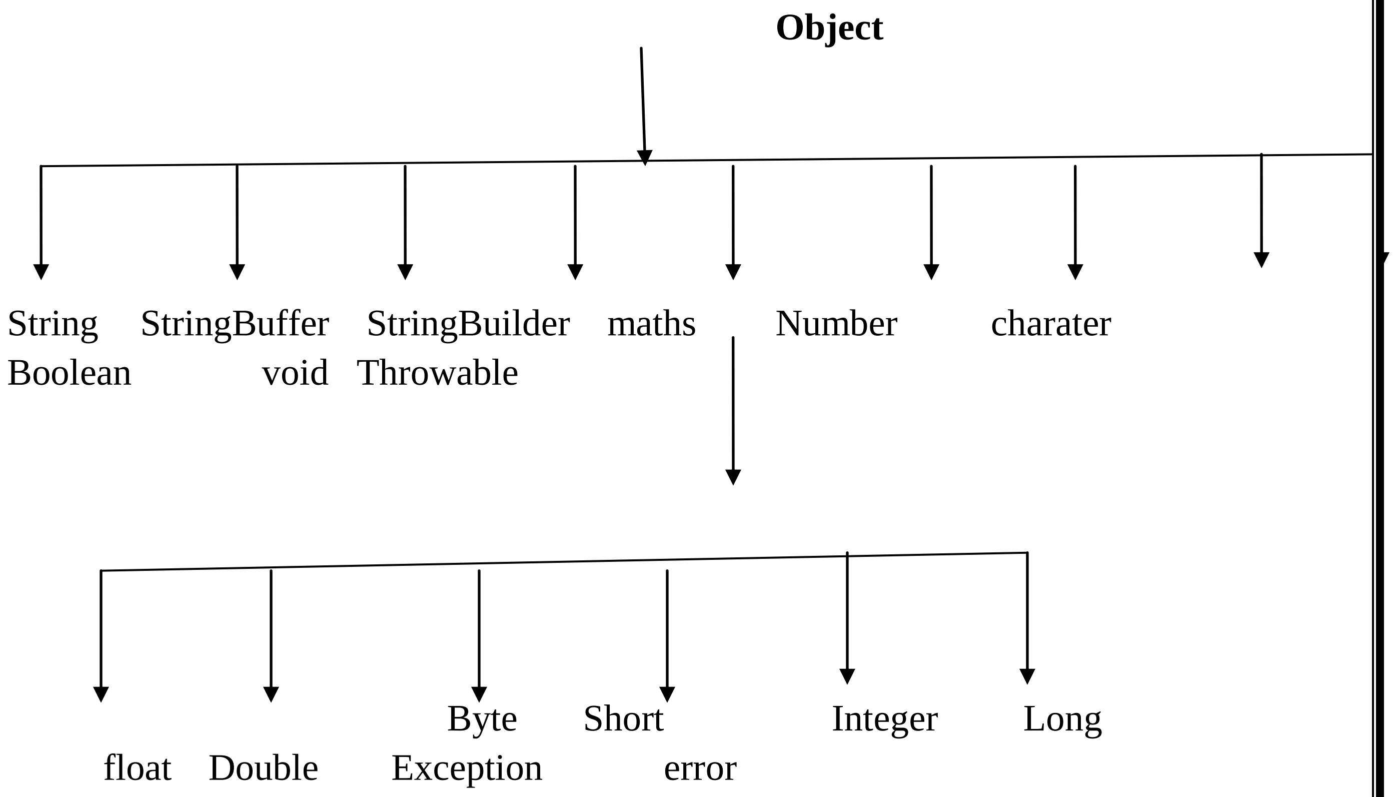
1. public static toBinaryString(primitive p);
2.public static string toOctalString(primitive p);
3.public static string toHexString(primitive p);

Ex. String s=Integer.toHexString(123);
s.o.pln(3);

Dancing between String,wrapper object, and primitive value:-



Partial hierarchy of java.lang package:-



->String,StringBuffer, StringBuilder all wrapper classes are final.
->The wrapper classes which are not child classes of number, character and Boolean
-> The wrapper classes which are not child classes of object are byte,short,Integer,Long,float,double
-> Sometimes we can consider void also as wrapper classes.
-> In addition to String object all wrapper object are Immutable.

Autoboxing and Autounboxing:-

-> until 1.4 version we can't provide primitive value in the place of wrapper objects and wrapper objects in the place of primitive. All the required conversion should be performed explicitly by the program

Ex.

```
(1) ArrayList l=new ArrayList();
    l.add();   X C.E
    Integer I=new Integer(10);
    l.add(i);

(2) Boolean b=new Boolean(true);

If(B)  →  C.E:- Incompatible types
{
found:Boolean,required:boolean

s.o.pln("hello");

}

Boolean b=B.booleanValue();

If(b)

{

s.o.pln("Hello");

}
```

->But from 1.5 version on words in the place of wrapper object we can provide primitive value and in the place of primitive value we can provide wrapper object. All the required conversions will be performed automatically by the compiler.

There automatic conversion are called Autoboxing and Autounboxing

Autoboxing:-

-> Automatic conversion of primitive value to the wrapper object by compiler is called autoboxing.

Ex. Integer i=10; [compiler convert int to integer automatically by autoboxing]

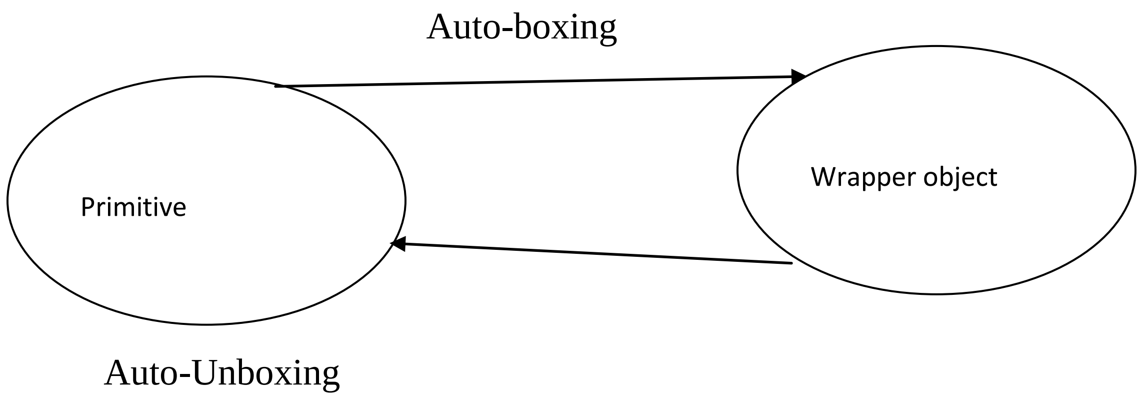
Auto-Unboxing:-

->Automatic conversion of wrapper object to the primitive type by compiler is called “Auto-Unboxing”.

Ex.

1) int i=new Integer(10); [compiler converts integer to int automatically by auto-unboxing]

Note:-



ex.
(1) Integer i=10;
After compilation this line will become
Integer I=Integer.valueOf(10);

i.e Autoboxing concept internally implemented by using valueOf()

ex.

```
        Integer I=new Integer(10);
int i=I;
```

-> After completion this line will become

```
        int i=I.intValue();
```

i.e Autounboxing concat internally implemented by using xxxValue()

Exam purpose:-

Ex

```
class Test
{
Static Integer I=10;                -> (1) A.B
p.s.v.m(String[] args)
{
int i=I;                            ->(2) A.U.B
m1(i); ------------>            (3) A.B
}
p.s.v.m1(Integer I)
{
int k=I;    ->(4) A.U.B
s.o.pln(k);   //10
}
}
```

Note:-

->Because of autoboxing and Auto-unboxing from 1.5 version onwards there there is no diff between primitive value and wrapper objects. We can use interchanging.

Ex.

```
 class Test
        {
Static Integer I=0;
```

```
Class Test
{
static Integer I;
p.s.v.m(String agrs[])
p.s.v.m(String agrs[])
{
NullPointerException
int i=I;
s.o.pln(i);   //0
}
}
 int i=I.intValue();
```

```
{
int i=I;     -->R.E:-

            s.o.pln(i);
}    }
int i=I.intvalue()
```

Ex3:-

```
        Integer x=10;
Integer y=x;
  s.o.pln(x);     //11
  s.o.pln(y);   //10
  creating an object
```

$$\begin{tikzcd} x \arrow[r] & (10) \\ y \arrow[ru] & \end{tikzcd}$$

note:- Because if we want to change after then that few changed object is created with the same reference name

```
ex4:- (1) Integer x=new Integer(10);
       Integer y=new Integer(10);
       s.o.pln(x==y);       //false
       (2) Integer x=new Integer(10);
       Integer y=10;
       s.o.pln(x==y);                //false
       (3) Integer x=10;
           Integer y=10;
       s.o.pln(x==y);  //true
       (4) Integer x=100;
       Integer y=100;
       s.o.pln(x==y);                //true
       (5) Integer x=1000;
       Integer y=1000;
       s.o.pln(x==y);                //false
```

Conclusion:-

->By autoboxing if an object is required to create compiler want create that object immidiatly first check is any object already created.
->If it is already created then it will reuse existing object instead of created new one.
->If it is not already there, then only a new object will be created .
->But this rule is applicable only in the following cases.
(1) Byte->Always
(2)Short-> -128 to 127
(3)Integer-> -128 to 127
(4) Long-> -128 to 127
(5) Character -> 0 to 127
(6) Boolean->always
->Except the above range in all other cases compulsory a new object will be created.

Ex:-

```
(1)Integer I1=127;          (1) Byte->Always
(2)Integer I2=127           (2)Short-> -128 to 127
    s.o.pln(I1==I2);//true  (3) Integer-> -128 to 127
                            (4)Long-> -128 to 127
                            (5)Character->  -128 to 127
                            (6)Boolean -> Always
(2) Integer I1=128;
Integer I2=128;
s.o.pln(I1==I2);//true

(3)Float f1=10.0f;
    Float f2=10.0f;
    s.o.pln(f1==f2);  //false
(4) Boolean b1=true;
Boolean b2=true;
S.o.pln(b1==b2);  //true
```

->Overloading write a auto-boxing, widing and var-arg methods:-

Case(1):-
 Widening vs Auto-boxing:-
Ex:-

```
 class Test
{
p.s.v.m1(long 2)
{
s.o.pln("widing");
}
p.s.v.m2(Integer I)
{
s.o.pln("Autoboxing");
}
p.s.v.m(String args[])
{
int x=10;
m1(2);  //widening
}
}
```

->Widening dominates Auto-boxing

Case(2):-
->widening vs var-arg():-
Ex.

```
 class Test
{
p.s.v.m1(long l)
{
s.o.pln("widening");
}
p.s.v.m2(int…i)
{
s.o.pln("var-arg");
}
p.s.v.main(string args[])
{
int x=10;
m1(x); // o/p widening
}
}
```

->Widening dominates var-arg()

Case3:-
Auto-boxing vs var-arg:-

Ex.

```
    class Test
{
p.s.v.m1(Integer I)
{
s.o.pln("Autoboxing");
}
p.s.v.m1(int..i)
{
s.o.pln("var-arg");
}
p.s.v.m(String args[])
{
int x=10;
m1(x);                    // o/p- Autoboxing
}
}
```

->In general var-arg() will get least priority,If no other method matched then only var-arg() will be executed.

->While resolving overloaded methods compiles will always keeps the precedence in the following order.

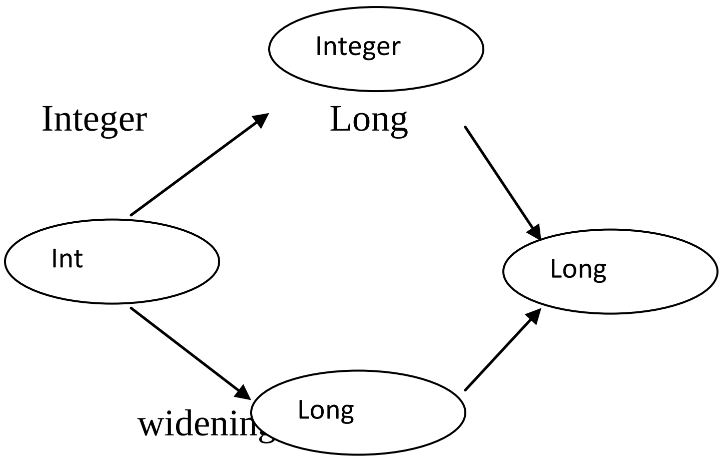
(I)Widening

(II)Auto-boxing

(III) Var-arg()

Case4:-
object

```
class Tesautoboxing
{
p.s.v.m1(Long l)
{
S.o.pln("Long");
Autoboxing
}
p.s.v.main(string arg[])
```



```
{
int x=10;
m1(x);
}
}C.E:-
            M1(java.lang.long) in test cannot be applied to(int)
```

->Widening followed by auto-boxing is not allowed in java where as autoboxing followed by widening is allowed

Ex.

```
class Test
{
p.s.v.m1(object o)
{
s.o.pln("object");
}
p.s.v.main(string args[])
{
int x=10;
m1(x);   //object
}
}
```

Q) Which of the following declarations are valid

(1) long l=10;   //write

(2)long l=10; //wrong

(3)Object o=10; //write

(4)double d=10; //write

(5)double d=10; //wrong

(6)Number n=10; //write