**Wrapper classes**

* The main objectives of wrapper classes are:
* To wrap primitives into object form so that we can handle primitives also just like objects.
* To define several utility functions which are required for the primitives.

**Constructors:**

* All most all wrapper classes define the following 2 constructors one can take corresponding primitive as argument and the other can take String as argument.

**Example:**

1. **Integer i=new Integer(10);**
2. **Integer i=new Integer(“10”);**

* If the String is not properly formatted then we will get runtime exception saying “**NumberFormatException**”.

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)throws Exception

{

Integer i=new Integer("ten");

System.out.println(i);//NumberFormatException

}

}

* Float class defines 3 constructors with float, String and double arguments.

1. **Float f=new Float (10.5f);**
2. **Float f=new Float (“10.5f”);**
3. **Float f=new Float(10.5);**
4. **Float f=new Float (“10.5”);**

* Character class defines only one constructor which can take char primitive as argument there is no String argument constructor.

**Character ch=new Character(‘a’);//valid**

Character ch=new Character(“a”);//invalid

* Boolean class defines 2 constructors with boolean primitive and String arguments.
* If we want to pass boolean primitive the only allowed values are true, false where case should be lower case.

**Example:**

Boolean b=new Boolean(true);

//Boolean b1=new Boolean(True);//C.E

//Boolean b=new Boolean(False);//C.E

* If we are passing String argument then case is not important and content is not important. If the content is case insensitive String of true then it is treated as true in all other cases it is treated as false.

**Example 1:**

class WrapperClassDemo

{

public static void main(String[] args)throws Exception

{

Boolean b1=new Boolean("true");

Boolean b2=new Boolean("True");

Boolean b3=new Boolean("false");

Boolean b4=new Boolean("False");

Boolean b5=new Boolean("bhaskar");

System.out.println(b1);//true

System.out.println(b2);//true

System.out.println(b3);//false

System.out.println(b4);//false

System.out.println(b5);//false

}

}

**Example 2(for exam purpose):**

class WrapperClassDemo

{

public static void main(String[] args)throws Exception

{

Boolean b1=new Boolean("yes");

Boolean b2=new Boolean("no");

System.out.println(b1);//false

System.out.println(b2);//false

System.out.println(b1.equals(b2));//true

System.out.println(b1==b2);//false

}

}

|  |  |
| --- | --- |
| **Wrapper class** | **Constructor summery** |
| Byte | byte, String |
| Short | short, String |
| Integer | Int, String |
| Long | long, String |
| Float | float, String, double |
| Double | double, String |
| Character |  |
| Boolean | boolean, String |

**Note:**

1. In all wrapper classes toString() method is overridden to return its content.
2. In all wrapper classes .equals() method is overridden for content compression.

**Utility methods:**

1. valueOf() method.
2. XXXValue() method.
3. parseXxx() method.
4. toString() method.

**valueOf() method:** We can use valueOf() method to create wrapper object for the given primitive or String this method is alternative to constructor.

**Form 1:** Every wrapper class except Character class contains a static valueOf() method to create wrapper object for the given String.

**public static wrapper valueOf(**[**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html)**s);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)throws Exception

{

Integer i=Integer.valueOf("10");

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

Boolean b=Boolean.valueOf("bhaskar");

System.out.println(i);//10

System.out.println(d);//10.5

System.out.println(b);//false

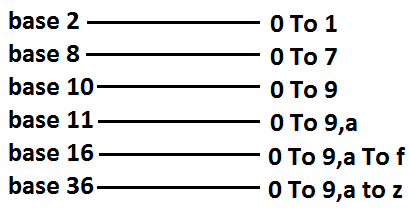
}

}

**Form 2:** Every integral type wrapper class (Byte, Short, Integer, and Long) contains the following valueOf() method to convert specified radix string to wrapper object.



**Note:** the allowed radix range is 2-36.



**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

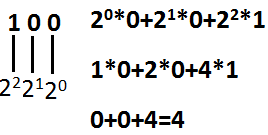
Integer i=Integer.valueOf("100",2);

System.out.println(i);//4

}

}

**Analysis:**



**Form 3:** Every wrapper class including Character class defines valueOf() method to convert primitive to wrapper object.

**public static wrapper valueOf(primitive p);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)throws Exception

{

Integer i=Integer.valueOf(10);

Double d=Double.valueOf(10.5);

Boolean b=Boolean.valueOf(true);

System.out.println(i);//10

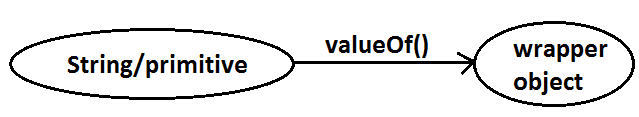
System.out.println(d);//10.5

System.out.println(b);//true

}

}

**Diagram:**



**xxxValue() method:** We can use xxxValue() methods to convert wrapper object to primitive.

* Every number type wrapper class (Byte, Short, Integer, Long, Float, Double) contains the following 6 xxxValue() methods to convert wrapper object to primitives.

1. **public byte byteValue()**
2. **public short shortValue()**
3. **public int intValue()**
4. **public long longValue()**
5. **public float floatValue()**
6. **pblic double doubleValue();**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)throws Exception

{

Integer i=new Integer(130);

System.out.println(i.byteValue());//-126

System.out.println(i.shortValue());//130

System.out.println(i.intValue());//130

System.out.println(i.longValue());//130

System.out.println(i.floatValue());//130.0

System.out.println(i.doubleValue());//130.0

}

}

**charValue() method:** Character class contains charValue() method to convert Character object to char primitive.

**public char charValue();**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

Character ch=new Character('a');

char c=ch.charValue();

System.out.println(c);//a

}

}

**booleanValue() method:** Boolean class contains booleanValue() method to convert Boolean object to boolean primitive.

**public boolean booleanValue();**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

Boolean b=new Boolean("bhaskar");

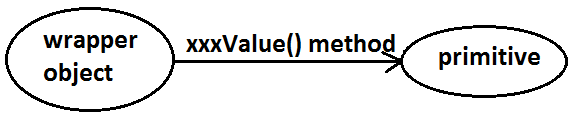
boolean b1=b.booleanValue();

System.out.println(b1);//false

}

}

**Diagram:**



* In total there are 38(6\*6+1+1) xxxValue() methods are possible.

**parseXxx() method:** We can use this method to convert String to corresponding primitive.

**Form1:** Every wrapper class except Character class contains a static parseXxx() method to convert String to corresponding primitive.

**public static primitive parseXxx(**[**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html)**s);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

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

boolean b=Boolean.parseBoolean("bhaskar");

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

System.out.println(i);//10

System.out.println(b);//false

System.out.println(d);//10.5

}

}

**Form 2:** integral type wrapper classes(Byte, Short, Integer, Long) contains the following parseXxx() method to convert specified radix String form to corresponding primitive.

**public static primitive parseXxx(**[**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html)**s,int radix);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

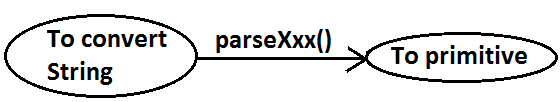
int i=Integer.parseInt("100",2);

System.out.println(i);//4

}

}

**Diagram:**



**toString() method:** We can use toString() method to convert wrapper object (or) primitive to String.

**Form 1:**

public [String](file:///D:\API\java%20SE\docs\api\java\lang\String.html) toString();

* Every wrapper class including Character class contains the above toString() method to convert wrapper object to String.
* It is the overriding version of Object class toString() method.
* Whenever we are trying to print wrapper object reference internally this toString() method only executed.

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

Integer i=Integer.valueOf("10");

System.out.println(i);//10

System.out.println(i.toString());//10

}

}

**Form 2:** Every wrapper class contains a static toString() method to convert primitive to String.

**public static** [**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html) **toString(primitive p);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

String s1=Integer.toString(10);

String s2=Boolean.toString(true);

System.out.println(s1);//10

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

}

}

**Form 3:**

* Integer and long classes contains the following static toString() method to convert the primitive to specified radix String form.

**public static** [**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html) **toString(primitive p,int radix);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

String s1=Integer.toString(7,2);

String s2=Integer.toString(17,2);

System.out.println(s1);//111

System.out.println(s2);//10001

}

}

**Form 4:** Integer and Long classes contains the following toXxxString() methods.

**public static** [**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html) **toBinaryString(primitive p);**

**public static** [**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html) **toOctalString(primitive p);**

**public static** [**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html) **toHexString(primitive p);**

**Example:**

class WrapperClassDemo

{

public static void main(String[] args)

{

String s1=Integer.toBinaryString(7);

String s2=Integer.toOctalString(10);

String s3=Integer.toHexString(20);

System.out.println(s1);//111

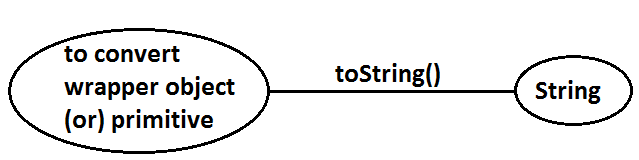
System.out.println(s2);//12

System.out.println(s3);//14

}

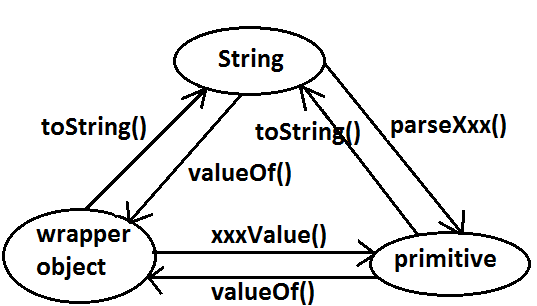
}

**Diagram:**



**Dancing between String, wrapper object and primitive:**

**Diagram:**



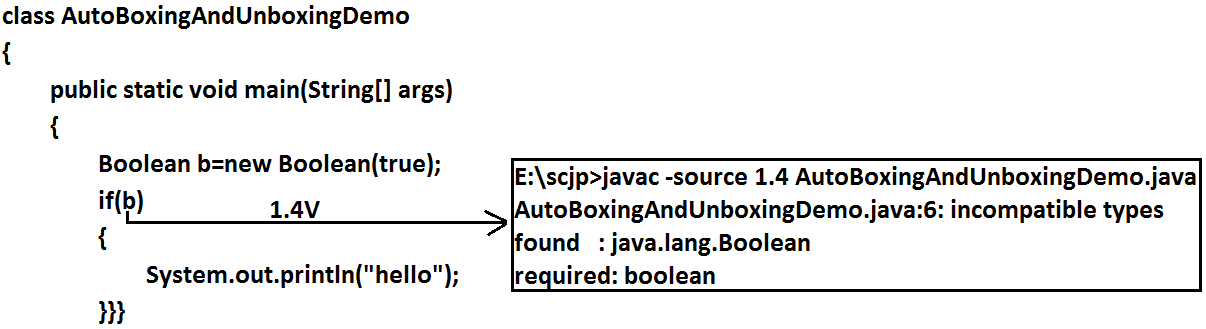
* String, StringBuffer, StringBuilder and all wrapper classes are final classes.
* The wrapper classes which are not child class of **Number** of Boolean and Character.
* The wrapper classes which are not direct class of Object of Byte, Short, Integer, Long, Float, Double.
* Sometimes we can consider **Void** is also as wrapper class.
* In addition to String all wrapper objects also immutable in java.

**Autoboxing and Autounboxing**

* Until 1.4 version we can’t provide wrapper object in the place of primitive and primitive in the place of wrapper object all the required conversions should be performed explicitly by the programmer.

**Example 1:**

**Program 1:**

****

**Program 2:**

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Boolean b=new Boolean(true);

if(b)

{

System.out.println("hello");

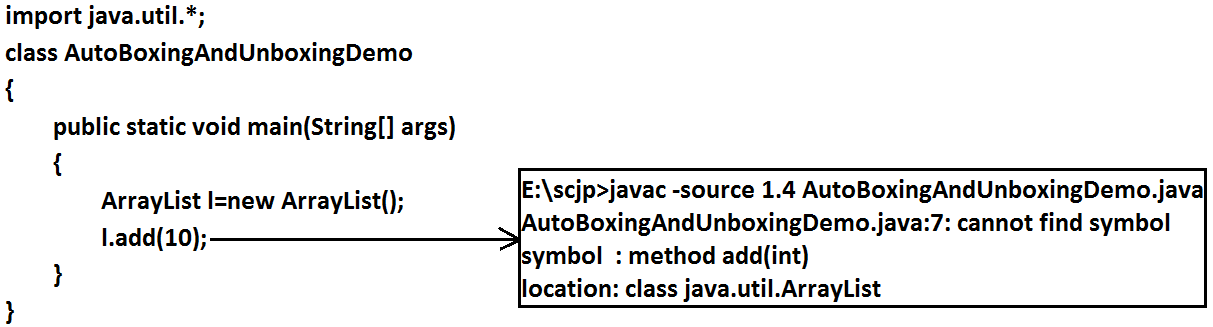
}}}

**Output:**

Hello

**Example 2:**

**Program 1:**

****

**Program 2:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

ArrayList l=new ArrayList();

Integer i=new Integer(10);

l.add(i);

}

}

* But from 1.5 version onwards we can provide primitive value in the place of wrapper and wrapper object in the place of primitive all required conversions will be performed automatically by compiler. These automatic conversions are called Autoboxing and Autounboxing.

**Autoboxing:** Automatic conversion of primitive to wrapper object by compiler is called Autoboxing.

**Example:**

Integer i=10; [compiler converts “int” to “Integer” automatically by Autoboxing]

* After compilation the above line will become.

Integer i=Integer.valueOf(10);

* That is internally Autoboxing concept is implemented by using valueOf() method.

**Autounboxing:** automatic conversion of wrapper object to primitive by compiler is called Autounboxing.

**Example:**

Integer i=new Integer(10);

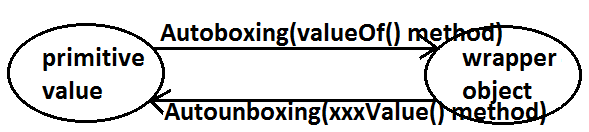
Int i=I; [compiler converts “Integer” to “int” automatically by Autounboxing]

* After compilation the above line will become.

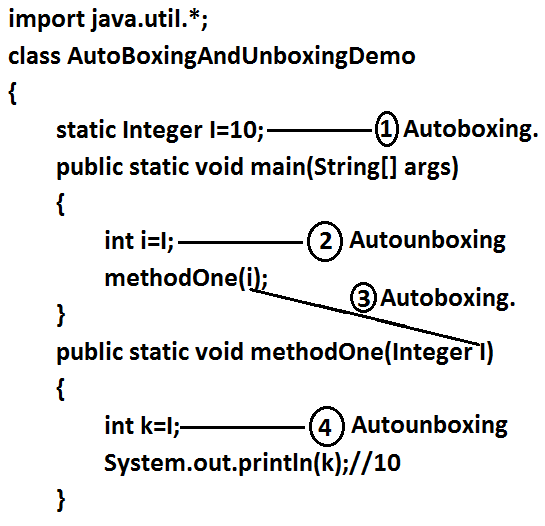
Int i=I.intValue();

* That is Autounboxing concept is internally implemented by using xxxValue() method.

**Diagram:**



**Example:**



**Note:** From 1.5 version onwards we can use primitives and wrapper objects interchangly the required conversions will be performed automatically by compiler.

**Example 1:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

static Integer I=0;

public static void main(String[] args)

{

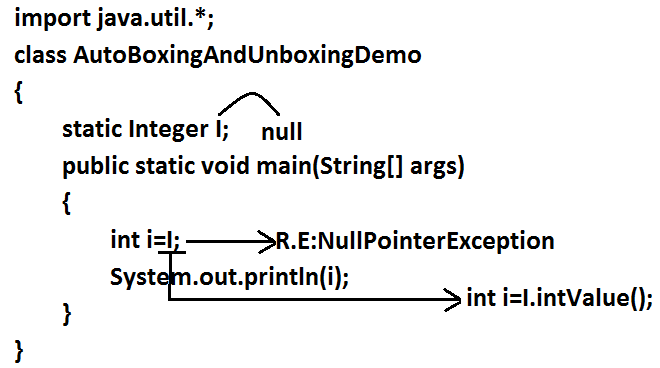
int i=I;

System.out.println(i);//0

}

}

**Example 2:**



**Example 3:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=10;

Integer y=x;

++x;

System.out.println(x);//11

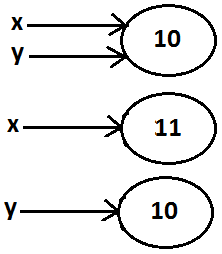
System.out.println(y);//10

System.out.println(x==y);//false

}

}

**Diagram:**



**Note:** All wrapper objects are immutable that is once we created a wrapper 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.

**Example 4:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=new Integer(10);

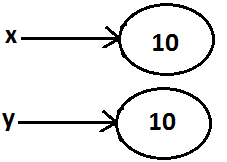
Integer y=new Integer(10);

System.out.println(x==y);//false

}

}

**Diagram:**



**Example 5:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=new Integer(10);

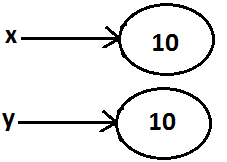
Integer y=10;

System.out.println(x==y);//false

}

}

**Diagram:**



**Example 6:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=new Integer(10);

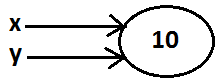
Integer y=x;

System.out.println(x==y);//true

}

}

**Diagram:**



**Example 7:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=10;

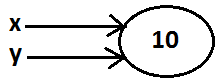
Integer y=10;

System.out.println(x==y);//true

}

}

**Diagram:**



**Example 8:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=100;

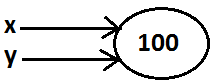
Integer y=100;

System.out.println(x==y);//true

}

}

**Diagram:**



**Example 9:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void main(String[] args)

{

Integer x=1000;

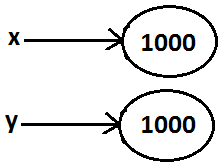
Integer y=1000;

System.out.println(x==y);//false

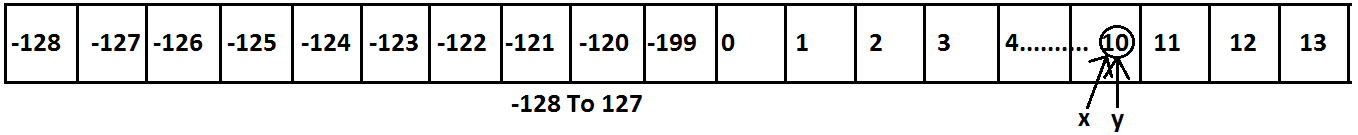
}

}

**Diagram:**



**Diagram:**



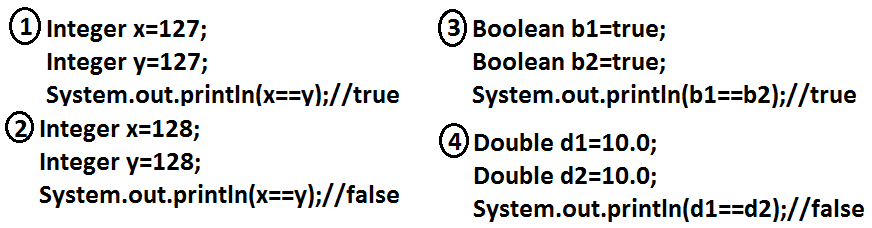
**Conclusions:**

* To implement the Autoboxing concept in every wrapper class a buffer of objects will be created at the time of class loading.
* By Autoboxing if an object is required to create 1st JVM will check whether that object is available in the buffer or not. If it is available then JVM will reuse that buffered object instead of creating new object. If the object is not available in the buffer then only a new object will be created. This approach improves performance and memory utilization.
* But this buffer concept is available only in the following cases.

|  |  |
| --- | --- |
| Byte | Always |
| Short | -128 To 127 |
| Integer | -128 To 127 |
| Long | -128 To 127 |
| Character | 0 To 127 |
| Boolean | Always |

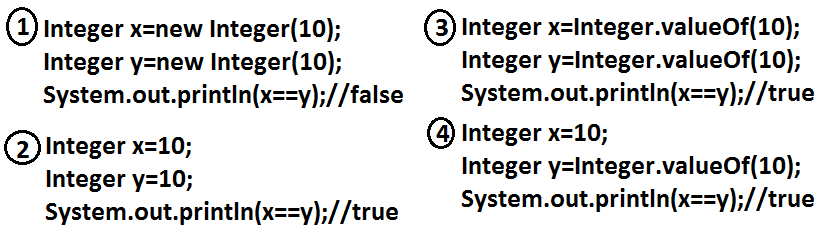
* In all the remaining cases compulsory a new object will be created.

**Examples:**



* Internally Autoboxing concept is implemented by using valueOf() method hence the above rule applicable even for valueOf() method also.

**Examples:**



**Note:** When compared with constructors it is recommended to use valueOf() method to create wrapper object.

**Overloading with respect to widening, Autoboxing and var-arg methods:**

**Case 1:** **Widening vs Autoboxing.**

**Widening:** Converting a lower data type into a higher data type is called widening.

**Example:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void methodOne(long l)

{

System.out.println("widening");

}

public static void methodOne(Integer i)

{

System.out.println("autoboxing");

}

public static void main(String[] args)

{

int x=10;

methodOne(x);

}

}

**Output:**

Widening

* Widening dominates Autoboxing.

**Case 2:** **Widening vs var-arg method.**

**Example:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void methodOne(long l)

{

System.out.println("widening");

}

public static void methodOne(int... i)

{

System.out.println("var-arg method");

}

public static void main(String[] args)

{

int x=10;

methodOne(x);

}

}

**Output:**

Widening

* Widening dominates var-arg method.

**Case 3:** Autoboxing vs var-arg method.

**Example:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void methodOne(Integer i)

{

System.out.println("Autoboxing");

}

public static void methodOne(int... i)

{

System.out.println("var-arg method");

}

public static void main(String[] args)

{

int x=10;

methodOne(x);

}

}

**Output:**

Autoboxing

* Autoboxing dominates var-arg method.
* In general var-arg method will get least priority. That is if no other method matched then only var-arg method will get chance. It is exactly same as “default” case inside a switch.

1. **Widening**
2. **Autoboxing**
3. **Var-arg method.**

**Case 4:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void methodOne(Long l)

{

System.out.println("Long");

}

public static void main(String[] args)

{

int x=10;

methodOne(x);

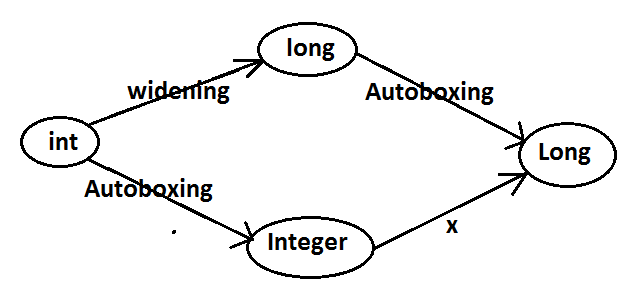
}

}

**Output:**

* methodOne(java.lang.Long) in AutoBoxingAndUnb oxingDemo cannot be applied to (int)

**Diagram:**



* Widening followed by Autoboxing is not allowed in java but Autoboxing followed by widening is allowed.

**Case 5:**

import java.util.\*;

class AutoBoxingAndUnboxingDemo

{

public static void methodOne(Object o)

{

System.out.println("Object");

}

public static void main(String[] args)

{

int x=10;

methodOne(x);

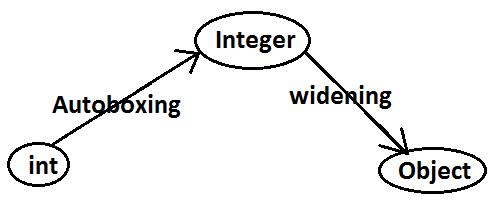
}

}

**Output:**

Object

**Diagram:**



**Which of the following declarations are valid?**

1. Longl=10;(valid)
2. Long l=10;(invalid)(C.E)
3. Long l=10l;(Autoboxing)
4. Number n=10; (valid)
5. Object o=10.0; (valid)
6. int i=10l; (invalid)(C.E)