Unboxing :
---

* It is a technique through which we can convert Wrapper object back to the primitive data type.

Conversion of Wrapper Object to primitive before 1.5V

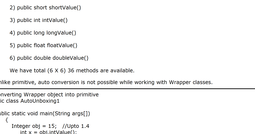
```
Integer obj = 25;
int x = obj.intValue();
```

Conversion of Wrapper object to primitive from JDK 1.5V onwards :

```
Integer obj = 55;
int x = obj;
```

Wrapper Object		Primitive type
Byte	-	byte
Short	-	short
Integer	-	int
Long	-	long
Float	-	float
Double	-	double
Chracter	-	char
Boolean	-	boolean

* As we know we have total 8 primitive data types in java. Among all these 8 primitive data 6 data types i.e byte, short, int, long, float and double represents numeric type so java has provided a predefined class called **java.lang.Number which support numeric type hence this class java.lang.Number contains 6 sub classes as shown below :**



All the above six warpper classes have provided the following methdos :

1) public byte byteValue()

2) public short shortValue()

3) public int intValue()

4) public long longValue()

5) public float floatValue()

6) public double doubleValue()

We have total (6 X 6) 36 methods are available.

* Unlike primitive, auto conversion is not possible while working with Wrapper classes.

```
//Converting Wrapper object into primitive
public class AutoUnboxing1
{
  public static void main(String args[])
    {
       Integer obj = 15;   //Upto 1.4
          int x = obj.intValue();
          System.out.println(x);
    }
}
```

```
public class AutoUnboxing2
{
    public static void main(String[] args)
    {
          Integer x = 25;
          int y = x;        //JDK 1.5 onwards
          System.out.println(y);
    }
}
```

```
public class AutoUnboxing3
{
    public static void main(String[] args)
    {
          Integer i = 15;
          System.out.println(i.byteValue());
          System.out.println(i.shortValue());
          System.out.println(i.intValue());
          System.out.println(i.longValue());
          System.out.println(i.floatValue());
          System.out.println(i.doubleValue());
    }
}
```

```
public class AutoUnboxing4
{
    public static void main(String[] args)
    {
          Character c1 = 'A';
          char ch = c1.charValue();
          System.out.println(ch);
    }
}
```

```
public class AutoUnboxing5
{
    public static void main(String[] args)
    {
          Boolean b1 = true;
          boolean b = b1.booleanValue();
          System.out.println(b);
    }
}
```

```
public class Test
{
    public static void main(String[] args)
    {
          Long a = 12L;
          System.out.println(a);

          Float b = 15F;
          System.out.println(b);

          Double c = 1.0;
          System.out.println(c);

          Double d = 1D;
          System.out.println(d);
    }
}
```

Note : From the above program , It is clear that auto-conversion is not possible while working with Wrapper classes but same is possible with primitive.

```
      long x = 12;  //Automatic Type Casting OR Widening [int is converted to long]
```

Polymorphism :

Poly = Many
Morphism = forms

* Poly means "many" and morphism means "forms".
* It is a Greek word whose meaning is "SAME OBJECT HAVING DIFFERENT BEHAVIOR".
* In our real life a person can perform so many task as shown below :

```
 Example :
 void person(Walkig w)
 void person(Running w)
 void person(Reading w)
```

* In the same way, In our programming languages, A method OR a constructor can perform so many task.

```
 Example :
 void add(int x, int y)
 {
 }

 void add(float x, float y)
 {
 }

 void add(String x, String y)
 {
 }
```

Method names are same but parameters are different so add method can perform so many task.

* Types of Polymorphism :

We have 2 types of Polymorphism :

1) Compile time OR Static Polymorphism OR Early Binding
2) Runtime OR Dynamic Polymorphism OR Late Binding

Static Polymorphism :
---

* The polymorphism which **exist at the time of compiliation** is called static polymorphism.

* In static polymorphism, compiler has very good idea regarding method call based on the **method parameter** type.

* The **binding of the method** is done at the time of compilation that is the reason It is also known as Early binding.

* We can achieve static polymorphism by using "Method Overloading".

Dynamic Polymorphism :
