Autoboxing and Unboxing :

Autobxing :

* It is a technique through which we can convert our PRIMITIVE DATA TYPE INTO corresponding object.

```
Primary Data type          Wrapper Object
      byte          :         Byte
      short         :         Short
      int           :         Integer
      long          :         Long
      float         :         Float
      double        :         Double
      char          :         Character
      boolean       :         Boolean
```

We have total 8 Wrapper classes.

* Autoboxing concept is available fromn JDK 1.5V which provides the facility to convert the primitive into Wrapper object as shown in the above diagram.

Autoboxing.java

```
void main()
{
    byte b = 12;
    Byte b1 = b;   //Autoboxing
    IO.println(b1);

    short s = 125;
    Short s1 = s;  //Autoboxing
    IO.println(s1);

  int x = 90;
    Integer x1 = x; //Autoboxing
    IO.println(x1);

  long y = 12L;
    Long y1 = y;   //Autoboxing
    IO.println(y1);

    float f = 12.5f;
    Float f1 = f;  //Autoboxing
    IO.println(f1);

  double d = 90.67;
    Double d1 = d; //Autoboxing
    IO.println(d1);


    char ch = 'A';
    Character ch1 = ch;   //Autoboxing
    IO.println(ch1);

    boolean a = true;
    Boolean a1 = a;  //Autoboxing
    IO.println(a1);
}
```

Unboxing :

It is a technique through which we can convert wrapper object back
to primitive data type. [Reverse of Autoboxing]

It is available from JDK 1.5V

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

Unboxing.java

```
void main()
{
    Byte b1 = 12;
    byte b = b1;   //Unboxing
    IO.println(b);

    Short s1 = 122;
    short s = s1;   //Unboxing
    IO.println(s);

    Integer i1 = 129;
    int i = i1;   //Unboxing
    IO.println(i);


    Long l1 = 12L;
    long l = l1;   //Unboxing
    IO.println(l);

    Float f1 = 12F;
    float f = f1;   //Unboxing
    IO.println(f);

    Double d1 = 12D;
    double d = d1;   //Unboxing
    IO.println(d);

    Character c1 = 'R';
    char c = c1;   //Unboxing
    IO.println(c);


    Boolean x1 = true;
    boolean x = x1;   //Unboxing
    IO.println(x);
}
```

Note : While working with primitive type, we can convert one type to another type but the same is not possible while working with Wrapper classes

```
Long l = 12;      //Invalid    [Long l = 12L; (Valid)]
Float f = 78;     //Invalid    [Float f = 78f; (Valid)]
Double d = 90;  //Invalid      [Double d = 90D; (Valid)]
```

Before JDK 1.5v How It was working :

Autoboxing.java

```
    int x = 90;
     //Convert the primitive into Object [Upto JDK 1.4]
     Integer obj = Integer.valueOf(x);
     IO.println(obj);

    //JDK 1.5V (Autoboxing)
     int y = 100;
     Integer y1 = y;
     IO.println(y1);
```

Unboxing.java

```
Integer obj = 100;
    //Converting Wrapper object to primitive [Upto JDK 1.4]
    int x = obj.intValue();
    IO.println(x);

    //JDK 1.5
    Integer y = 123;
    int z = y;
    IO.println(z);
```

Method Overloading Resolution :

In order to learn Method Overloading Resolution, We must have 3 basic concepts

Var args in Java
Widening (implicit type conversion) in Java
Autoboxing in Java

In case of ambiguity where compiler can select more than one method then compiler will provide the priority by using following rules :

1) Exact Match Rule :

First of all compiler will always search for exact match.

Example : 'A' [here It is character so compiler will search the method which accepts character as a parameter]

2) Most Specific Type :

Compiler alwyas provide more priority to most specific data type.
Example :

double > float [float is the most specific type]
float > long
long > int
int > char
int > short
short > byte

Note : THERE IS NO RELATION BETWEEN char and byte as well as char and short data type.

3) WAV [Widening -> Autoboxing -> Var Args]
Compiler gives the priority to select appropriate method by using the following sequence :
Widening ---> Autoboxing ----> Var args

//Programs on the above rules :

MOLR1.java

```
void main()
{
    accept(6);
}

public void accept(double d)
{
    IO.println("double");
}
public void accept(float d)
{
    IO.println("float");
}
```

Note : Here float will be executed becuase float is the most specific type.

MOLR2.java

```
void main()
{
  accept(6);
}

public void accept(int d)
{
 IO.println("int");
}
public void accept(char d)
{
 IO.println("char");
}
```

Here 6 is int type so int will be executed. [Exact match Rule]

MOLR3.java

```
void main()
{
 accept();
}

public void accept(int ...d)
{
 IO.println("int");
}
public void accept(char ...d)
{
 IO.println("char");
}
```

char will be executed becoz char is more specific type.

MOLR4.java

```
void main()
{
 accept();
}

public void accept(short ...d)
{
  IO.println("short");
}
public void accept(char ...d)
{
  IO.println("char");
}
```

Here we will get compilation error because there is no relation between char and short based on the specific type rule.

MOLR5.java

```
void main()
{
  accept();
}

public void accept(short ...d)
{
  IO.println("short");
}
public void accept(byte ...d)
{
  IO.println("byte");
}
```

Here byte will be executed because byte is the specific type.

MOLR6.java

```
void main()
{
 accept();
}

public void accept(double ...d)
{
 IO.println("double");
}
public void accept(long ...d)
{
 IO.println("long");
}
```

Here long will be executed because long is the most specific type.

MOLR7.java

```
void main()
{
 accept(9);
}

public void accept(int d)
{
 IO.println("int");
}
public void accept(long s)
{
 IO.println("long");
}
```

Note : Here int will be executed [exact match]

MOLR8.java

```
void main()
{
 accept(9);
}

public void accept(long s)
{
 IO.println("Widening");
}
public void accept(Integer i)
{
 IO.println("Autoboxing");
}
```

Here widening is having more priority

MOLR9.java

```
void main()
{
 accept(9);
}

public void accept(int ...s)
{
     IO.println("Var args");
}
public void accept(Integer i)
{
   IO.println("Autoboxing");
}
```

Here Autoboxing will be executed.

MORL10.java

```
void main()
{
     accept(9,2);
}

public void accept(int ...s)
{
     IO.println("Var args");
}
public void accept(long x , long y)
{
   IO.println("Widening");
}
```

Here Widening will be executed.

MORL11.java

```
void main()
{
     accept(9,2);
}

public void accept(int x, long y)
{
     IO.println("int - long");
}
public void accept(long x , int y)
{
   IO.println("long - int");
}
```

Here It is compilation error