# Autoboxing and Unboxing in Java

In Java 1.5, Java added concept of autoboxing and unboxing that deal with conversion of primitive type to object and vice versa.

The term **autoboxing** refers to the **auto conversion** of **primitive** type to its correspond **object** type. For example, conversion of int type to Integer object or char type to Character object. This conversion is done implicitly by the Java compiler during program execution.

In the same context, when an object coverts to its correspond primitive type then it is called **unboxing**. For example, conversion of Integer type to int type or Byte to byte type etc. Java automatically performs this conversion during program execution.

Example of Autoboxing

In this example, we are assigning an int type value to Integer object and notice compiler does not report any error because it performs autoboxing here.

```
class Demo

{

     public static void main(String[] args)

     {

          Integer i = 100; // Auto-boxing of int i.e converting
primitive data type int to a Wrapper class Integer

          System.out.println(i);



          Character ch = 'a';

          System.out.println(ch);



          Byte b = 12;

          System.out.println(b);



     }

}
```

Copy

```
100
```

```
a

12
```

Explanation:

Whenever we use object of Wrapper class in an expression, autoboxing is done by JVM.

This will happen always, when we will use Wrapper class objects in expressions or conditions etc.

Example: Autoboxing in Collection

```
import java.util.ArrayList;


class Demo

{

     public static void main(String[] args)

     {

          ArrayList<Integer> arrayList = new ArrayList<Integer>();

          arrayList.add(100); // autoboxing int to Integer

          arrayList.add(200);

          arrayList.add(300);

          for(Integer i : arrayList) {

               System.out.println(i);

          }

     }

}
```

Copy

```


100

200

300
```

In the same manner, we can perform unboxing. Lets see an example.

Example : Unboxing

In this example, we using arraylist to store int type values. Since arraylist stores only object then it automatically converts int type to Integer and store the elements. If we fetch the elements of it

returns object type and if we store it into primitive int type then it automatically converts Integer to int type. See the below example.

```
import java.util.ArrayList;


class Demo
{
     public static void main(String[] args)
     {
          ArrayList arrayList = new ArrayList();
          arrayList.add(100); // autoboxing int to Integer
          arrayList.add(200);
          arrayList.add(300);
          for(Integer i : arrayList) {
               System.out.println(i);
          }
          // unboxing Integer to int type
          int first = arrayList.get(0);
          System.out.println("int value "+first);
     }
}
```

Copy

```
100
200
300
int value 100
```

Benefits of Autoboxing / Unboxing

1. Autoboxing / Unboxing lets us use primitive types and Wrapper class objects interchangeably.
2. We don't have to perform Explicit **typecasting**.

3. It helps prevent errors, but may lead to unexpected results sometimes. Hence must be used with care.
4. Auto-unboxing also allows you to mix different types of numeric objects in an expression. When the values are unboxed, the standard type conversions can be applied.

Example:

```
class Demo {

     public static void main(String args[]) {

          Integer i = 35;

          Double d = 33.3;

          d = d + i;

          System.out.println("Value of d is " + d);

      }

}
```

Copy

```
Value of d is 68.3
```

**Note:** When the statement **d = d + i;** was executed, i was auto-unboxed into int, d was auto-unboxed into double, addition was performed and then finally, auto-boxing of d was done into Double type Wrapper class.

Type Wrappers

Java uses primitive data types such as int, double, float etc. to hold the basic data types for the sake of performance. Despite the performance benefits offered by the primitive data types, there are situations when you will need an object representation of the primitive data type. For example, many data structures in Java operate on objects. So you cannot use primitive data types with those data structures. To handle such type of situations, Java provides **type Wrappers** which provide classes that encapsulate a primitive type within an object.

Java provides wrapper classes correspond to each primitive type to deal with objects that are tabled below.

| Primitive | Wrapper class |
|---|---|
| int | Integer |
| byte | Byte |

| | |
|---|---|
| short | Short |
| float | Float |
| double | Double |
| char | Character |
| long | Long |
| boolean | Boolean |

Above mentioned Classes comes under Numeric type wrapper. These classes encapsulate byte, short, int, long, float, double primitive type.

Example: Autoboxing and Unboxing in method

We can see how autoboxing and unboxing take place in a method. Notice, during method call we passed int type primitive value but method accepts only Integer objects so here JVM does autoboxing implicitly. And notice, this method returns an int type primitive which is another conversion (unboxing) from Integer to int type.

```
class Demo {

     // autoboxing in parameters

     static int add(Integer a, Integer b) {

          // unboxing in return

          return a+b;

     }


    public static void main(String args[]) {


          int sum = add(10,20);


          System.out.println("sum = "+sum);


          }
```

```
}
```

Copy

```
sum = 30
```