**Java Autoboxing and Unboxing with examples**

[**JAVA TUTORIALS**](http://beginnersbook.com/category/technology/java-guide/java-tutorials/)

Java 1.5 introduced a special feature of auto conversion of primitive types to the corresponding Wrapper class and vice versa.

**Autoboxing**: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

**Unboxing**: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Primitive type Wrapper class

boolean Boolean

byte Byte

char Character

float Float

int Integer

long Long

short Short

double Double

**When does the autoboxing and unboxing happens in Java**

**Autoboxing**

: Lets see few cases with examples, where autoboxing happens.  
**Case 1**: When a method is expecting a wrapper class object but the value that is passed as parameter is a primitive type. For example in the below code, the method myMethod() is expecting an object of Integer wrapper class, however we passed a primitive int type. The program ran fine as compiler does the autoboxing (conversion of int to Integer)

class AutoboxingExample1

{

public static void myMethod(Integer num){

System.out.println(num);

}

public static void main(String[] args) {

/\* passed int (primitive type), it would be

\* converted to Integer object at Runtime

\*/

myMethod(2);

}

}

Output:

2

**Case 2**: When at some point of time, you are assigning a primitive type value to an object of its wrapper class. For example: The below statements are valid because compiler does the autoboxing at runtime.

Integer inum = 3; //Assigning int to Integer: Autoboxing

Long lnum = 32L; //Assigning long to Long: Autoboxing

**Case 3**: When dealing with collection framework classes:

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

arrayList.add(11); //Autoboxing - int primitive to Integer

arrayList.add(22); //Autoboxing

Here ArrayList class is expecting an Integer wrapper class object but we are providing int primitive.

**Unboxing**

**Case 1**: Method is expecting Integer object (parameter) but we have supplied int. Auotmatic conversion(unboxing) happened that converted Integer to int.

class UnboxingExample1

{

public static void myMethod(int num){

System.out.println(num);

}

public static void main(String[] args) {

Integer inum = new Integer(100);

/\* passed Integer wrapper class object, it

\* would be converted to int primitive type

\* at Runtime

\*/

myMethod(inum);

}

}

Output:

100

**Case 2**: Assignments

Integer inum = new Integer(5);

int num = inum; //unboxing object to primitive conversion

**Case 3**: While dealing with collection classes:

ArrayList arrayList = new ArrayList()

int num = arrayList.get(0); // unboxing because get method returns an Integer object

**What happens behind the scenes?**

In the above section we learnt how java compiler performs automatic conversion between primitive type and corresponding Wrapper objects. Lets discuss what compiler actually does during autoboxing and unboxing. The best way to understand this is to compare things before java 1.5 and after java 1.5 (boxing and unboxing introduced in java 1.5).

**Autoboxing**:  
What we see:

Integer number = 100;

What compiler does (or what we used to do before java 1.5):

Integer number = Integer.valueOf(100);

**Unboxing**:  
What we see:

Integer num2 = new Integer(50);

int inum = num2;

What compiler does:

Integer num2 = new Integer(50);

int inum = num2.intValue();

Similar things happen with the other wrapper classes and primitive types such as long, double, short etc.

**Few things you should take care:**

Do not mix primitives and objects while doing comparisons. You might get unpredictable results for such comparisons. Better thing to do is: compare object with objects (using equals() method) and compare primitive with primitives(using logical operators such as “==”, “<” etc).