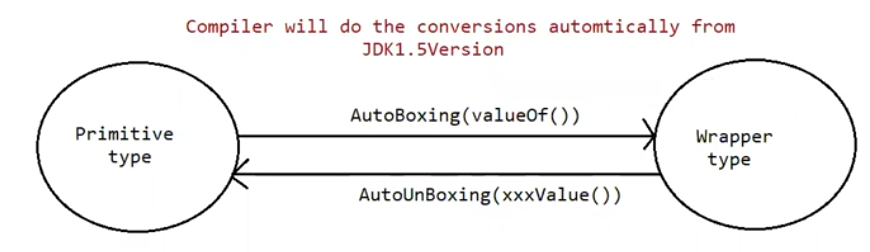
AutoBoxing and AutoUnBoxing :



Until 1.4 version , we cant provide wrapper class objects in place of primitive and primitive in place of wrapper class object all the required conversions should be done by the programmer.

But from jdk 1.5 version onwards , we can provide primitive in place of wrapper and wrapper in place of primitive . All the required conversions will be done by the compiler automatically, this mechanism is called “AutoBoxing” and “AutoUnBoxing”.

Eg: AutoBoxing\_UnBoxing\_Example

AutoBoxing :

Automatic conversion of primitive type to wrapper object is called autoboxing.

Integer i1 = 20;

|

| after compilation

|

Integer i1 = Integer.valueOf(10); // automatically done by compiler.

Note: AutoBoxing is done by the compiler using a method call valueOf().

AutoUnBoxing :

Automatic conversion of wrapper object to primitive type is called AutoUnBoxing.

Integer i1 = new Integer(10);

int i2 = i1;

|

| after compilation

|

int i2 = i1.intvalue(); // automatically done by the compiler.

AutoUnBoxing is done by the compiler using the method XXXvalue().

Integer i1 = new Integer(“10”) // from jdk 11 it is deprecated . means in future this may be removed.

Integer i2 = Integer.valueOf(“10”);

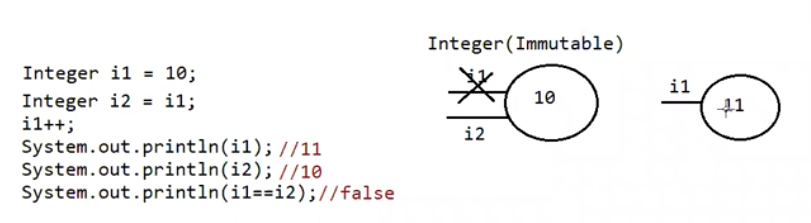
Eg: AutoBoxing\_UnBoxing\_Eg2

// go through the code.

Eg: AutoBoxing\_UnBoxing\_Eg3

//go through the code

Eg: AutoBoxing\_Unboxing\_Eg4



Note : all the wrapper classes are immutable by default.

i++ // here it is not the increment of object , actually what is done by jvm is i1 = i1+1.

Eg: AutoBoxing\_Unboxing\_Eg5

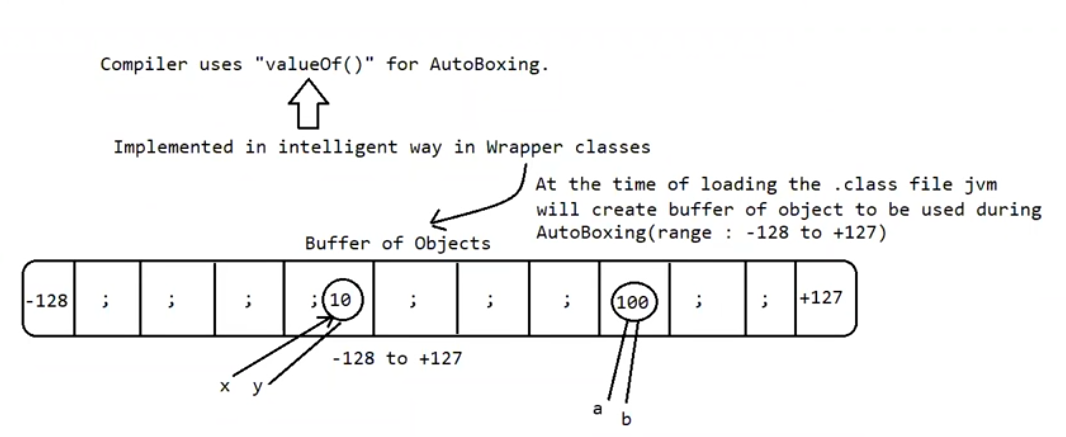
// go through the code.

Eg: AutoBoxing\_UnBoxing\_Eg6

// go through the code

Eg: AutoBoxing\_UnBoxing\_Eg7

//go through the code



Note :

1. To implement the autoboxing concept in wrapper class a buffer object will be created at the time of class loading.
2. During autoboxing if an object has to be created first jvm will check whether the object is already available buffer or not.
3. If it is available jvm will reuse the buffered object instead of creating a new object.
4. If the object is not available in the buffered area , then jvm will create a new object in the heap area. This approach improves the performance and memory utilization.

But this buffer concept is available only for few cases.

Byte => -128 to +127

Short => -128 to +127

Integer => -128 to +127

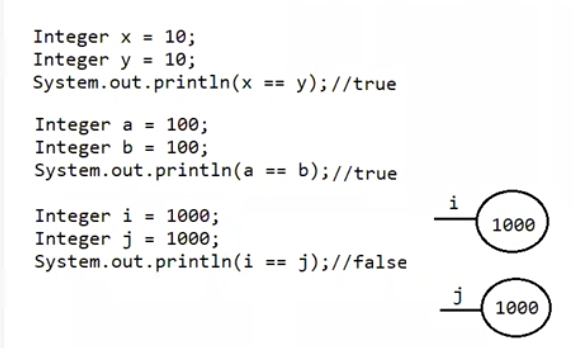
Long => -128 to +127

Character => 0 to 127

Boolean => true, false

In the remaining cases new objects will be created in the heap. If a wrapper object is created using the constructor then object is created in the heap.

Eg: Buffer\_Of\_Object



// here buffer of object concept is used . new object is created only if the data is not in the range of buffer of object.

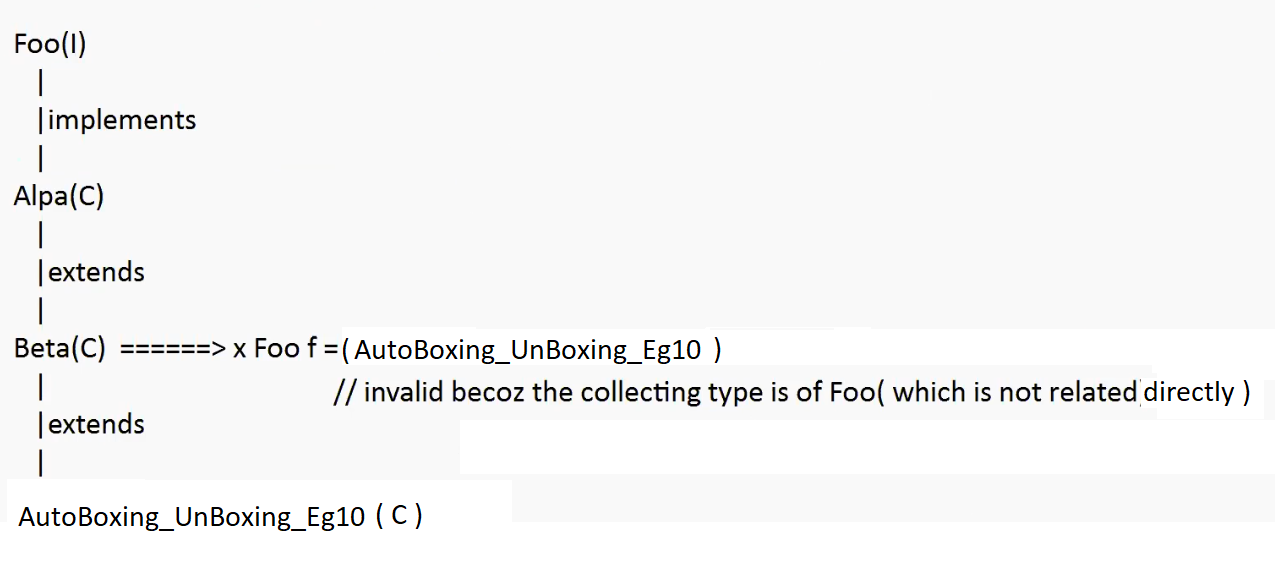
Eg: AutoBoxing\_UnBoxing\_Eg8

// go through the code

Eg: AutoBoxing\_UnBoxing\_Eg9

// go through the code

Eg: AutoBoxing\_UnBoxing\_Eg10



// go through the code also.

Eg:AutoBoxing\_UnBoxing\_Eg11

// here program parameters args[0] = “1” and args[1] = “null” .

The valueOf() converts string or primitive to wrapper type so the result will be

new Boolean (“1) = > new Boolean (false) => false

and auto unboxing is happening to store the data in the primitive type boolean variable a.

similarly new Boolean ( “null”) => new Boolean(false) =>false

and auto unboxing is happening to store the data in the primitive type boolean variable b.

Eg: AutoBoxing\_UnBoxing\_Eg12

// go through the code

Eg: AutoBoxing\_UnBoxing\_Eg13

// go through the code

Eg: AutoBoxing\_UnBoxing\_Eg14

// go through the code

Eg: AutoBoxing\_UnBoxing\_Eg15

// go through the code