ECAP615

Programming in Java

Harjinder Kaur

Assistant Professor

Learning Outcomes

After this lecture, you will be able to

- Learn the basic concept of wrapper classes.
- Understand the use and need of wrapper classes.
- Analyze the difference in primitive types and wrapper object.
- Learn the concept of boxing and unboxing.

Wrapper Classes

- Java uses primitive types, such as int, char,double to hold the basic data types supported by the language.
- Sometimes it is required to create an object representation of these primitive types.
- These are collection classes that deal only with such objects.
- One needs to wrap the primitive type in a class.

Wrapper Classes

- The wrapper class in Java provides the mechanism *to* convert *primitive into object and object into primitive*.
- autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Wrapper Classes

- To satisfy this need, java provides classes that correspond to each of the primitive types.
- Basically, these classes encapsulate, or wrap, the primitive types within a class.
- Thus, they are commonly referred to as type wrapper. Type wrapper are classes that encapsulate a primitive type within an object.
- The wrapper types are Byte, Short, Integer, Long, Character, Boolean, Double, Float.

Use of Wrapper classes in Java

- Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc.
- Let us see the different scenarios, where we need to use the wrapper classes.

✓Change the value in Method

✓Serialization

✓Synchronization

✓Collection Framework

Wrapper Classes

The eight classes of the *java.lang* package are known as wrapper classes in Java.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Difference b/w Primitive Data Type and Object of a Wrapper Class

- The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

```
int x = 25;

Integer y = new Integer(33);
```

- The first statement declares an int variable named x and initializes it with the value 25.

Difference b/w Primitive Data Type and Object of a Wrapper Class

- The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.

```
int x = 25;

Integer y = new Integer(33);
```

Clearly x and y differ by more than their values:

- x is a variable that holds a value;
- y is an object variable that holds a reference to an object.

Need of Wrapper Classes

- Wrapper classes are used to be able to use the primitive data types as objects.
- Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.

Need of Wrapper Classes

- An object is needed to support synchronization in multithreading.
- The classes in java.util package handles only objects and hence wrapper classes help in this case also.

Boxing and Unboxing

- The wrapping is done by the compiler.
- If we use a primitive where an object is expected, the compiler boxes the primitive in its wrapper class.
- Similarly, if we use a number object when a primitive is expected, the compiler un-boxes the object.

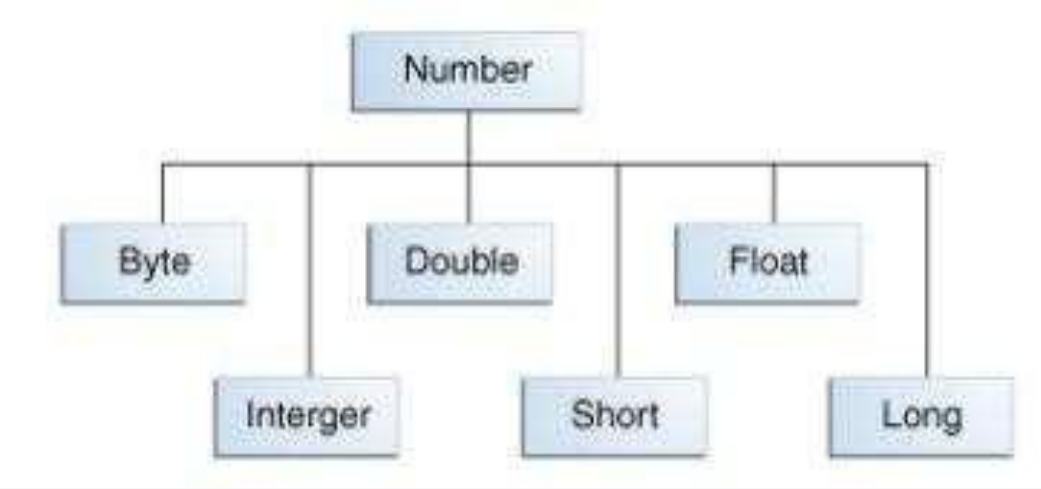
Example of boxing and unboxing

```
Integer x, y;

x = 12; y = 15; //boxing

System.out.println(x+y);//Unboxing
```

- When x and y are assigned integer values, the compiler boxes the integers because x and y are integer objects.
- In the println() statement, x and y are unboxed so that they can be added as integers.

Numeric Wrapper Classes

- All of the numeric wrapper classes are subclasses of the abstract class Number.
- Short, Integer, Double and Long implement Comparable interface.



Features of Numeric Wrapper Classes

- All the numeric wrapper classes provide a method to convert a numeric *string into a primitive value*.

Syntax: public static type parseType (String Number)

- parseInt()
- parseFloat()
- parseDouble()
- parseLong()

Features of Numeric Wrapper Classes

- All the wrapper classes provide a static method toString to provide the *string representation of the primitive values.*

Syntax: public static String toString (type value)

Example: public static String toString (int a)

Features of Numeric Wrapper Classes

- All numeric wrapper classes have a static method valueOf, which is used to *create a new object initialized to the value* represented by the specified string.

Syntax: public static DataType valueOf (String s)

Example:

- Integer i = Integer.valueOf (“135”);
- Double d = Double.valueOf (“13.5”);

Methods implemented by subclasses of Number

- Compares this Number object to the argument.
- int compareTo(Byte anotherByte)
- int compareTo(Double anotherDouble)
- int compareTo(Float anotherFloat)

Methods implemented by subclasses of Number

- int compareTo(Integer anotherInteger)
- int compareTo(Long anotherLong)
- int compareTo(Short anotherShort)
- returns int after comparison (-1, 0, 1).

Character Class

- Character is a wrapper around a char.
- The constructor for Character is :

Character(char ch)

- Here, ch specifies the character that will be wrapped by the Character object being created.

Character Class

- To obtain the char value contained in a Character object, call charValue( ), shown here:

```
char charValue( );
```

- It returns the encapsulated character.

Boolean Class

- Boolean is a wrapper around boolean values.
- It defines these constructors:
  - Boolean(boolean boolValue)
  - Boolean(String boolString)
- In the first version, boolValue must be either true or false.
- In the second version, if boolString contains the string “true” (in uppercase or lowercase), then the new Boolean object will be true. Otherwise, it will be false.

Boolean Class

- To obtain a boolean value from a Boolean object, use booleanValue( ), shown here:

```
boolean booleanValue( )
```

- It returns the boolean equivalent of the invoking object.

Autoboxing

- The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing.
- For example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Example

```
public class AutoBoxingTest {

   public static void main(String args[]) {

      int num = 10; // int primitive

      Integer obj = Integer.valueOf(num); // creating a wrapper class object

      System.out.println(num + " " + obj);

   }

}
```

Output: 10 10

Unboxing

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.
- It is the reverse process of autoboxing.

Example

```
public class UnboxingTest {
   public static void main(String args[]) {
      Integer obj = new Integer(10); // Creating Wrapper class object
      int num = obj.intValue(); // Converting the wrapper object to primitive datatype
      System.out.println(num + " " + obj);
   }
}
```

Output: 10 10


That’s all for now…