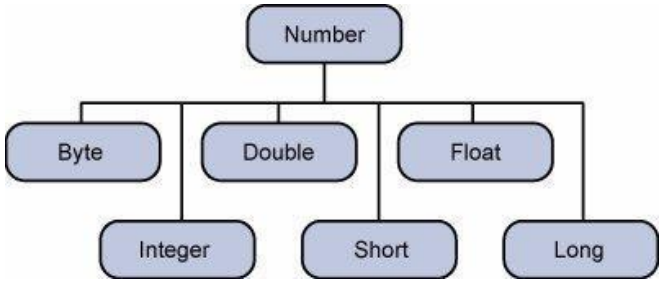
Wrapper class in Java

Wrapper classes in java are the Object representation of eight primitive types in java. All the wrapper classes in java are **immutable** and final. Java 5 **autoboxing** and **unboxing** allows easy conversion between primitive types and their corresponding wrapper classes in java programs.

All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.



The object of the wrapper class contains or wraps its respective primitive data type. Converting primitive data types into object is called **boxing**, and this is taken care by the compiler. Therefore, while using a wrapper class you just need to pass the value of the primitive data type to the constructor of the Wrapper class.

And the Wrapper object will be converted back to a primitive data type, and this process is called **unboxing.** The **Number** class is part of the ***java.lang*** package.

An example of boxing and unboxing:−

Example

```
public class Test {
  public static void main(String args[]) {
    Integer x = 5; // boxes int to an Integer object
    x =  x + 10;   // unboxes the Integer to a int
    System.out.println(x);
  }
}
```

Output: 15

When x is assigned an integer value, the compiler boxes the integer because x is integer object. Later, x is unboxed so that they can be added as an integer.

Autoboxing and unboxing lets developers write cleaner code, making it easier to read. The following table lists the primitive types and their corresponding wrapper classes, which are used by the Java compiler for autoboxing and unboxing:

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

The wrapper classes in java servers two primary purposes.

- To provide a mechanism to 'wrap' primitive values in an object so that primitives can do activities reserved for the objects like being added to ArrayList, Hashset, HashMap etc. collection (*Collection classes*).
- To provide an assortment of utility functions for primitives like converting primitive types to and from string objects, converting to various bases like binary, octal or hexadecimal, or comparing various objects.

Why do we need wrapper classes?

It was a smart decision to keep primitive types and Wrapper classes separate to keep things simple. We need wrapper classes when we need a type that will fit in the Object world programming like Collection classes. We use primitive types when we want things to be simple.

Wrapper classes are used in situations where objects are required, such as for elements of a Collection:

```
List<Integer> a = new ArrayList<Integer>();

methodRequiringListOfIntegers(a);
```