

Integer.valueOf (Strings) and int Value () methods from the java.lang.Integer class. To convert the string “22” into the int .

```
int I = Integer.valueOf (“22”).intValue();
```

Doubles, floats and long are converted similarly. To convert a String like “22” into the long value 22, we would write

```
long1 = Long.valueOf(“22”).longValue();
```

To convert “22.5” into a float or a double, we would write:

```
double x = Double.valueOf(“22.5”).doubleValue();
```

```
float y = Float.valueOf(“22.5”).floatValue();
```

The various **valueOf()** methods are relatively intelligent and can handle plus and minus signs, exponents, and most other common number formats. However, if we pass one of these methods, something completely non-numeric, like “hello world,” it will throw a **NumberFormatException**.

```
class area
```

```
{  
public static void main (String args[])  
{  
double radius;  
double pi = 3.14;  
double A  
radius = Double.valueOf(args[0]).doubleValue ();  
A = pi*radius*radius;  
System.out.println(A + “square meters”)  
}  
}
```

WRAPPER CLASSES

In Java, primitive data types such as long, int, char etc. are not treated as objects. This is for simplicity and efficiency; but many times, we need to treat these primitive type as objects, for example, when we want to store primitive data types along with other objects in an array. The Java programming language provides “**Wrapper**” to manipulate primitive data elements as objects. Each primitive data type has a corresponding wrapper class in the java.lang package. These classes are:

Primitive Data Type	Wrapper class
Boolean	Boolean
Byte	Byte
Char	Character
Short	Short
Int	Integer
Long	Long
Float	Float

A wrapper class object is constructed by passing the value to be wrapped into the appropriate constructor. For example:

```
int ptyp=10;  
Integer pobj = new Integer(ptyp);
```

INNER CLASSES

A class that is declared and defined inside some other class is called an **inner class**. Sometimes, it is also called a nested class. The inner classes give additional functionality to the program, and make it clearer.

```
public class Outer  
{  
    int i=10;  
    public class Inner  
    {  
        int j=20;  
        public void innerFn()  
        {  
            System.out.println("j is "+ j);  
        }  
    }  
    public void outerFn()  
    {  
        System.out.println("I is "+I);  
    }  
    public static void main(String s [ ])  
    {  
        Outer out =new Outer();  
        out.outerFn();  
    }  
}
```

Enclosing the ‘this’ reference of Inner Classes.

Inner classes have access to their enclosing class’s scope. The access to enclosing class’s scope is possible because the **inner class** actually has a hidden reference to the outer class **this reference**.

Example:

```
public class Outer  
{  
    int i=10;  
    public class Inner  
    {  
        int j;  
        public void innerFn()  
        {  
            System.out.println("I is "+I);  
            System.out.println("j is "+j);  
        }  
    }  
}
```

```

    }
    }
    public void innerCreate()
    {
        Inner in=new Inner();
        in.innerFn();
    }
    public void outerFn()
    {
        System.out.println("I is "+ i);
    }
    public static void main(String s[ ] )
    {
        Out out=new Outer();
        out.innerCreate();
    }
}

```

ADDING CLASSES TO EXISTING PACKAGES

Suppose a package A contains a public class, Class A, and we want to add another public class, Class B to this package.

```

package packageA;
public class ClassA;
{
//(body of classA)
}

```

```

package packageB;
public class ClassB
{
//(body of ClassB)
}

```

Store the source and compiled files ClassB.java and ClassB.class in the directory **packageA**. we can add non-public classes also in this manner. When the package, **packageA**, is imported, both the classes **ClassA** and **ClassB** are imported, as they are contained in that package.

A **.java** file can have only one public class. So, if we want to create a package with multiple public classes in it, create the classes in separate source files and declare the package statement

package packagename;

at the top of each source file. Switch to the subdirectory created with the package name, and compile each source file. Now, the package contains **.class** files of all the source files.