Wrapper Classes: Primitive data types may be converted into object type by using the wrapper classes. These wrapper classes are available in the java.lang package. So we can use it without an import statement.

There are eight wrapper classes:

Byte
Boolean
Character
Short
Integer
Long
Float
Double

Primitive types

In both Java and C++, there are a large number of primitive types. In both languages, the declaration and instantiation of a variable of a primitive type uses a statement such as the following to cause memory to be set aside for the variable and a name to be associated with the variable.

int myVar;

Combining all three steps for primitive types

Also, in both languages, you can initialize the value of the variable when it is declared, thereby accomplishing *declaration*, *instantiation*, and *initialization* all in one statement such as the following.

int myVar = 6;

What does the compiler know?

Perhaps even more important, everything that happens in the above statements regarding primitive variables is known to the compiler at compile time.

C++ also allows you to declare and instantiate variables of primitive types or objects from classes at runtime in such a way that the compiler doesn't know at compile time where the variable or object is stored

Static vs. dynamic memory in Java

In Java, all primitive variables must be allocated to static memory at compile time. Java does not allow primitive variables to be instantiated into dynamic memory at runtime.

However, there are wrapper classes for primitive types that can be used to turn them into objects for this purpose. In Java, all objects must be instantiated into dynamic memory at runtime. Java does not allow objects to be instantiated in static memory at compile time.

An array of objects

When C++ instantiates an array of objects in dynamic memory, it actually instantiates an array of objects and returns a pointer to the first object.

When Java instantiates *"an array of objects"* (which is always in dynamic memory), it actually instantiates an array of *references* to objects.

An additional step is required to create the objects pointed to by the references.

To review, in C++, an array of objects instantiated in dynamic memory consists of an actual array of objects and a single pointer is returned which points to the first object in the array. Pointer arithmetic can then be used to access the other objects.

An *array of objects* in Java is instantiated as an array of reference variables where each reference variable can then be used to instantiate an object and have it referred to by the reference variable.

Multiple use of the new operator is required in Java

In Java, multiple usage of the **new** operator is required to instantiate an *"array of objects"* because the array doesn't actually contain the objects; it contains references to the objects.

C++ requires only one use of the new operator

Only one usage of the **new** operator will instantiate a (single dimensional) array of objects in C++. Instantiation of an array of objects in Java is very similar to the instantiation of an array of pointers in C++, which is often used to point to a group of strings. However, from a syntax viewpoint, the array of *references* in Java is much easier to use than an array of pointers in C++.

- In wrapper class first character is upper case
- Character and Integer is not abbreviated as the type char and int.
- Byte,Boolean,Short,Integer,Long,Float,Double are subclass of the abstract class number.
- These wrapper classes encapsulate(or) 'wraps up' a primitive type. So that a variable can be represented by an object when necessary.
- Theses classes also have a constants MAX_VALUE and MIN_VALUE.

e.g. **Integer Class:** The Integer class is a wrapper class for primitive data type int.

- It wraps a value of the primitive type int in an object.
- This class provides methods for converting an int to String object and String to an int.
- parseInt(String s) //convert string to integer n return value int
- toBinaryString(int n) // returns a String that contain binary equivalent of n
- toHexString(int n) // returns a String that contain hexadecimal equivalent of n
- toOctalString(int n) // returns a String that contain Octal equivalent of n,

Wrapper:That in which an object is wrapped or covered

How do I convert numeric values to Strings with Java?

Converting numeric values to Strings with Java is pretty easy. The String class contains a method named "valueOf()" that can be used, and the numeric classes also contain "toString()" methods that can be used for the conversion.

Assuming that the variable i is an int, f is a float, d is a double, and l is a long, the following examples show how you can convert each numeric type to a String.

Using String.valueOf() for the conversions, you'd write the Java code this way:

```
String s = String.valueOf(i);
String s = String.valueOf(f);
String s = String.valueOf(d);
String s = String.valueOf(l);
```

Using the toString() method of the numeric classes (Integer, Float, Double, and Long), you'd write the conversions this way:

```
String s = new Integer(i).toString();
String s = new Float(f).toString();
String s = new Double(d).toString();
String s = new Long(l).toString();
```

```
class Convert
{
  public static void main(String[] args)
  {
    int i    = 100;
    float f  = (float) 200.0;
   double d = 400.0;
   long l   = 100000;
   String s = new String();
//-----------------------------------------------//      // Examples using
   "valueOf()" method of the   //      // String class.
   //      //-----------------------------------------------//
 s = String.valueOf(i);
 System.out.println ("int i = " + s);
```



```
s = String.valueOf(f);
 System.out.println ("float f = " + s);
s = String.valueOf(d);
System.out.println ("double d = " + s);
s = String.valueOf(l);      System.out.println ("long l = " + s);
//-------------------------------------------------//      // Examples using the
   "toString()" method of the  //      // numeric classes
   //      //-------------------------------------------------//
```



```
s = new Integer(i).toString();
System.out.println ("int i = " + s);
 s = new Float(f).toString();
System.out.println ("float f = " + s);
s = new Double(d).toString();
 System.out.println ("double d = " + s);
s = new Long(l).toString();
System.out.println ("long l = " + s);    }}
```

The java.lang is one of the most important packages in java. It provides the fundamental for java programming, its most important classes are:

- Object
- Class
- Math
- String
- StringBuffer
- System
- Thread, etc

The Object class: All the classes are subclass of the object class and inherit it methods. Object, which is the root of the java class hierarchy. The toString() method creates a string representation of the value of an object of the object class.

wait() and notify() are methods of the object class and used for controlling threads.

etc.

The System class:

System.in.read(); //input

System.out.println("helo"); //output

System.err.println("Error"); //error

The Class Class: This is used to find runtime state of an object or interface. Objects of type class are created automatically when classes are loaded.

getName()=returns the complete name of the class or interface of the invoking object.

getSuperClass()=returns the super class of the invoking object.

Java.lang has also some other usefull class. The Runtime and Process classes are used to execute other java programs.


```
class A
{
int x;
int y;
}
class B extends A
{
int x;
}
class f
{
```



```
public static void main(String s[])
{
A a1=new A();
B b1=new B();
System.out.println("a1 is object of
   type"+a1.getClass());
System.out.println("b1 is object of
   type"+b1.getClass());
}
}
```

e.g. If you want to launch notepad do the following:

exec("program name")=execute the program name
exitValue()= Return to the execution code by the process. This is 0 for no problem occurs.


```
class n
{
public static void main(String f[])
{Runtime r=Runtime.getRuntime();
//it returns the current object
Process p1=null;
try{
     p1=rt.exec("Notepad");
     p1.waitFor();//waits
 }
catch(Exception e)
{
  System.out.println("error");
  System.out.println("Notepad closed"+p.exitValue());

 }
 }
```


Math Class: The java.lang.Math class contains all the mathematical functions that are used for geometry and trigonometry, as well as statistics methods.

Math.method-name();//math class methods can be used
abs(x)- Returns absolute value of the x
ceil(x)
floor(x)
sin(x)
log(x)-returns the natural logarithm
pow(x,y)-returns the x raise to power y
max(x,y)
Sqrt(x)- Returns the square root of x
etc.