**Classes in JAVA**:

A class is a blueprint from which individual objects are created. The class is at the core of Java. It is the logical construct upon which entire Java language is built because it defines the shape and nature of an object. Any concept you wish to implement in a Java program must be encapsulated within a class.

Class defines a new data type. Once defined, this new type can be used to create objects of that type. Thus class is template for an object, and object is an instance of a class.

A class is declared by use of the class keyword. A simplified general form of a class definition is shown here…

Following is a sample of a class.

class Classname

{

type instance variable1; //Instance variables

type instance variable2; //Instance variables

…..

…..

type instance variableN;//Instance variables

type method1(parameter list)

{

//Body of method1;

}

type method2(parameter list)

{

//Body of method2;

}

//…..

type methodN(parameter list)

{

//Body of this method;

}

}

**Example**:

//program to show the simple class

/\* A program that uses the Box class.

Call this file BoxDemo.java

\*/

class Box

{

double width;

double height;

double depth;

}

// This class declares an object of type Box.

class BoxDemo

{

public static void main(String args[])

{

Box mybox = new Box();

double vol;

// assign values to mybox's instance variables

mybox.width = 10;

mybox.height = 20;

mybox.depth = 15;

// compute volume of box

vol = mybox.width \* mybox.height \* mybox.depth;

System.out.println("Volume is " + vol);

}

}

-> A class can contain any of the following variable types.

**Local variables**: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

**Instance variables**: Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

**Class variables**: Class variables are variables declared within a class, outside any method, with the static keyword.

* A class can have any number of methods to access the value of various kinds of methods
* Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

**Encapsulation:**

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

**Constructors**:

Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

There are two types of constructors 1. Default constructor 2. Parameterized constructor

Following is an example of a constructor −

**Example**:

// Here, Box uses a constructor to initialize the dimensions of a box.

class Box

{

double width;

double height;

double depth;

// This is the default constructor for Box.

Box()

{

System.out.println("Constructing Box");

width = 10;

height = 10;

depth = 10;

}

// compute and return volume

double volume()

{

return width \* height \* depth;

}

}

class BoxDemo

{

public static void main(String args[])

{

// declare, allocate, and initialize Box objects

Box mybox1 = new Box();

Box mybox2 = new Box();

double vol;

// get volume of first box

vol = mybox1.volume();

System.out.println("Volume is " + vol);

// get volume of second box

vol = mybox2.volume();

System.out.println("Volume is " + vol);

}

}

* Java also supports Singleton Classes where you would be able to create only one instance of a class.

**Example:**

// Here, Box uses a parameterized constructor to initialize the dimensions of a box.

class Box

{

double width;

double height;

double depth;

// This is the constructor for Box.

Box(double w, double h, double d) {

width = w;

height = h;

depth = d;

}

// compute and return volume

double volume() {

return width \* height \* depth;

}

}

class BoxDemo7

{

public static void main(String args[])

{

// declare, allocate, and initialize Box objects

Box mybox1 = new Box(10, 20, 15);

Box mybox2 = new Box(3, 6, 9);

double vol;

// get volume of first box

vol = mybox1.volume();

System.out.println("Volume is " + vol);

// get volume of second box

vol = mybox2.volume();

System.out.println("Volume is " + vol);

}

}

**Creating an Object:**

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class –

* **Declaration**: − A variable declaration with a variable name with an object type.
* **Instantiation**: − The 'new' keyword is used to create the object.
* **Initialization**: − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

**Declaring and definition of an object:**

Obtaining objects of a class is a two-step process.

First, you must declare a variable of the class type. This variable does not define an object.

Instead, it is simply a variable that can refer to an object. Second, you must acquire an actual,

physical copy of the object and assign it to that variable. You can do this using the new

operator. The new operator dynamically allocates (that is, allocates at run time) memory

for an object and returns a reference to it. This reference is, more or less, the address in

memory of the object allocated by new.

Box mybox = new Box();

This statement combines the two steps just described. It can be rewritten like this to show

each step more clearly:

Box mybox; // declare reference to object

mybox = new Box(); // allocate a Box object

The first line declares mybox as a reference to an object of type Box. At this point, mybox

does not yet refer to an actual object. The next line allocates an object and assigns a

reference to it to mybox. After the second line executes, you can use mybox as if it were a

Box object. But in reality, mybox simply holds, in essence, the memory address of the actual

Box object.

**new** **operator:**

You must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the ***new*** operator. The ***new*** operator instantiates a class by dynamically allocating (i.e. allocation at run time) memory for a new object and returning a reference to that memory. This reference is then stored in the variable. Thus, in Java, **all class objects must be dynamically allocated.**

The ***new*** operator is also followed by a call to a class constructor, which initializes the new object. A [constructor](http://www.geeksforgeeks.org/constructors-in-java/) defines what occurs when an object of a class is created. Constructors are an important part of all classes and have many significant attributes. In below example we will use the [default constructor](http://www.geeksforgeeks.org/g-fact-50/). Below is general syntax of instantiation and initialization with an example:

Box yellow=new Box();