2.1 What isclass?

A class is a group of objects that share the same properties and methods. It is a **template** or blueprint from which objects are created. The objects are the **instances** of class. Because an object is an instance of a class. A class is used to define new type of the data. Once defined, this new type can be used to create objects of its type. The class is the logical entity and the object is the logical and physicalentity.

The general form of the class

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

```
class classname
{
        type instance-variable1;
        type instance-variable2;
                ……..
                …….
        type instance-variableN;

        type method1(parameterlist)
        {
                //body of the method1
        }
        type method2(parameterlist)
        {
                //body of the method2
        }
                ……..
                …….
        type methodN(parameterlist)
        {
                //body of the methodN
        }
}
```

- The data or variables, defined within the class are called, ***instancevariables***.
- The methods also contain thecode.
- The methods and instance variables collectively called as***members***.
- Variable declared within the methods are called ***localvariables***.

A Simple Class

Let's begin our study of the class with a simple example. Here is a class called **Student** that defines three instance variables: **sid**, **sname**, and **branch**. Currently, **Student** does not contain any methods.

```
class Student
{ //instance variables
        int sid;
        String sname; String branch;
}
```

As stated, a class defines new data type. The new data type in this example is, Box. This defines the template, but does not actually create object.

Note: By convention the First letter of every word of the class name starts with Capital letter, but notcompul sory. For example student is written as"Student"The First letter ofthe word of the method name begins with small and remaining first letter of every word starts with Capital letter, butnotcompul sory. For exampleboxVolume( )‖

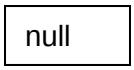
2.2 Creating theObject

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*' key word is used to create the object.
Initialization: The '*new*' keyword is followed by a call to a constructor. This call initializes the new object.

Step 1:
Student s;

Effect: s

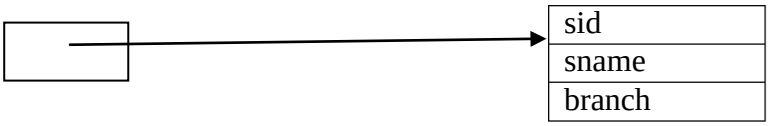


Declares the class variable. Here the class variable contains the value **null**. An attempt to access the object at this point will lead to Compile-Time error.

Step 2:
Student s=new Student();

Here *new* is the keyword used to create the object. The object name is *s*. The **new** operator allocates the memory for the object, that means for all instance variables inside the object, memory isallocated.

Effect:s



Student Object

Step 3:
There are many ways to initialize the object. The object contains the instance variable.
The variable can be assigned values with reference of the object.

s.sid=1234;
s.sname="Anand";
s.branch="IT";

Here is a complete program that uses the **Student** class: (**StudentDemo.java)**

```
class Student
{
        int sid;
        String sname;
        String branch;
```

```
}

// This class declares an object of type Student.
class StudentDemo
{
        public static void main(Stringargs[])
        {
                //declaring the object (Step 1) and instantiating (Step 2)object
                Student std = newStudent();

                // assign values to std's instance variables (Step 3)
                s.sid=1234;
                s.sname="Anand";
                s.branch="IT";

                // display student details
                System.out.println(std.sid" "+std.sname+""+std.branch);
        }
}
```

When you compile this program, you will find that two **.class** files have been created, one for **Student** and one for **StudentDemo**. The Javacompiler automatically puts each class into its own **.class** file. It is not necessary for both the **Student** and the **StudentDemo** class to actually be in the same sourcefile.
To run this program, you must execute **StudentDemo.class**. When you do, you will see the following output:

```
1234            Anand           IT
```

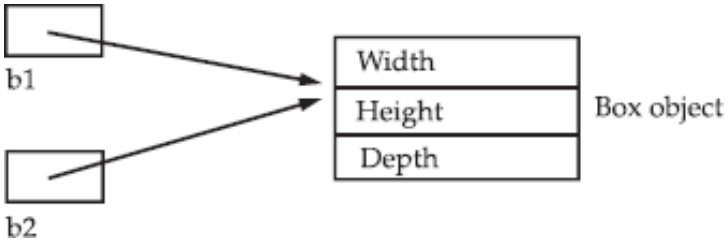
As stated earlier, each object has its own copies of the instance variables. This means that if you have two **Student** objects, each has its own copy of **sid**, **sname**, and **branch**.

2.2.1 Assigning Object Reference Variables

Object reference variables act differently than you might expect when an assignment takes place. For example, what do you think the following fragment does?

```
Box b1 = new Box();
Box b2 = b1;
```

You might think that **b2** is being assigned a reference to a copy of the object referred to by**b1**. That is, you might think that **b1** and **b2** refer to separate and distinct objects. However,this would be wrong. Instead, after this fragment executes, **b1** and **b2** will both refer to the*same* object. The assignment of **b1** to **b2** did not allocate any memory or copy any part of theoriginal object. It simply makes **b2** refer to the same object as does **b1**. Thus, any changesmade to the object through **b2** will affect the object to which **b1** is referring, since they are thesameobject.
This situation is depicted here:



2.3 Methods

Classes usually consist of two things:***instance variables and methods***. This is the general form of a method:

type name(*parameter-list*)
{
// body of method
}

Here,

- *type* specifies the type of data returned by the method. This can be any valid data type, including class types that youcreate.
- If the method does not return a value, its return type must be**void**.
- The name of the method is specified by *name.* This can be any legal identifier other than those already used by other items within the currentscope.
- The *parameter-list* is a sequence of type and identifier pairs separated bycommas.
- Parameters are essentially variables that receive the value of the ***arguments*** passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.
- Methods that have a return type other than **void** return a value to the calling routine using the following form of the **return**statement:

  return *value*;

Here, *value* is the value returned.

Adding a method to the Student class **(Student.java)**

```
class Student
{
        intsid;
        String  sname;
        Stringbranch;
        show()
        {
                System.out.println("Student ID"+sid);
                System.out.println("Student Name: "+sname);
                System.out.println("Branch:"+branch);
        }
}
```

Here the method name is "show( )". This methods contains some code fragment for displaying the

student details. This method can be accessed using the object as in the following code:

StudentDemo.java

```
class StudentDemo
{
	public static void main(String args[])
	{
		Student std = new Student();
		s.sid=1234;
		s.sname=”Anand”;
		s.branch=”IT”;
		std.show();		// display studentdetails

	}
}
```

Output

Student ID: 1234
Student Name: Anand
Branch:IT

Returning a Value

A method can also return the value of specified type. The method after computing the task returns the value to the **caller** of the method.

```
class Student
{
	int sid;
	String sname;
	String branch;
	int sub1, sub2, sub3;

	public float percentage()
	{
		return (sub1+ sub2+sub3)/3;
	}
	public void show()
	{
		System.out.println("Student ID"+sid);
		System.out.println("Student Name: "+sname);
		System.out.println("Branch:"+branch);
	}
}

class StudentDemo
{
	public static void main(String args[])
	{
		Student std = new Student();
		std.sid=1234;
		std.sname=”Anand”;
```

```
                std.branch=”IT”;
                std.sub1=89;
                std.sub2=90;
                std.sub3=91;
                float per=std.percentage();          //calling the method
                std.show();
                System.out.println("Percentage:"+per);
        }
}
```

Output

Student ID: 1234
Student Name: Anand
Branch: IT
Percentage: 90

2.3.1 Adding a method that takes theparameters

We can also pass arguments to the method through the object. The parameters separated with comma operator. The values of the actual parameters are copied to the formal parameters in the method. The computation is carried with formal arguments, the result is returned to the caller of the method, if the type ismentioned.

```
float percentage(int s1, int s2, ints3)
{
        sub1 = s1;
        sub2 = s2;
        sub3 = s3;
        return(sub1+sub2+sub3)/3;
}
```

2.3.2 MethodOverloading

In Java it is possible to define **two or more methods** within the same class that share the same name, as long as their parameter declarations are different. In this case, the methods are said to be *overloaded*, and the process is referred to as ***method overloading***. Method overloading is one of the ways that Java supports **polymorphism**.

- When an overloaded method is invoked, Java looks for a match between arguments of the methods to determine which **version** of the overloaded method to actuallycall.
- While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of amethod.
- When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in thecall.
- When an overloaded method is invoked, Method overloading supports **polymorphism** becauseit isone way that Java implementsthe –one interface, multiple methods|paradigm.

Here is a simple example that illustrates method overloading:

```
class Overload
{
        void test()
        {
```

```
                System.out.println("No parameters");
        }
        // Overload test for one integer parameter.
        void test(int a)
        {
                System.out.println("a: " + a);
        }
}
class OverloadDemo
{
        public static void main(String args[])
        {
                Overload ol = new Overload;
                double result;
                // call all versions of test()
                ob.test();
                ob.test(10);
        }
}
```

Output

No parameters
a: 10

The *test()* method is overloaded two times, first version takes no arguments, second version takes oneargument.

2.4 Constructors

A constructor is a special member function used for automatic initialization of an object. Whenever an object is created, the constructor is called automatically. Constructors can be overloaded.

Characteristics

- Constructors have the same name as that of the class they belongsto.
- They automatically execute whenever an object iscreated.
- Constructors will not have any return type evenvoid.
- Constructors will not return anyvalues.
- The main function of constructor is to initialize objects and allocation of memory tothe objects.
- Constructors can beoverloaded.
- A constructor without any arguments is called as defaultconstructor.

Example

```
class Student
{
        int sid;
        String sname;
        String branch;
        int sub1, sub2, sub3;

        Student()               // This is the constructor forStudent
```

```
	{
		sid=1234;
		sname=”Anand”;
		branch=”IT”;
		sub1=89;
		sub2=90;
		sub3=91;
	}
	public float percentage()
	{
		return (sub1+ sub2+sub3)/3;
	}
	public void show()
	{
		System.out.println("Student ID"+sid);
		System.out.println("Student Name: "+sname);
		System.out.println("Branch:"+branch);
	}
}
class StudentDemo
{
	public static void main(Stringargs[])
	{
		// declare, allocate, and initialize Student object
		Student std = newStudent();
		float per=std.percentage();		//calling the method
		std.show();
		System.out.println("Percentage:"+per);
	}
}
```

Output

Student ID: 1234
Student Name: Anand
Branch: IT
Percentage: 90

Parameterized Constructors

It may be necessary to initialize the various data members of different objects with different values when they are created. This is achieved by passing arguments to the constructor function when the objects are created. The constructors that can take arguments are called ***parameterized constructors***.

```
/* Here, Student uses a parameterized constructor to initialize the details ofastudent.	*/
class Student
{
	int sid;
	String sname;
	String branch;
	int sub1, sub2, sub3;
	// This is a parameterized constructor forStudent
	Student(int id, String name, String b, int s1 ,int s2, int s3 )
```

```
	{
		sid=id;
		sname=name;
		branch=br;
		sub1=s1;
		sub2=s2;
		sub3=s3;
	}
	public float percentage()
	{
		return (sub1+ sub2+sub3)/3;
	}
	public void show()
	{
		System.out.println("Student ID"+sid);
		System.out.println("Student Name: "+sname);
		System.out.println("Branch:"+branch);
	}
}
class StudentDemo
{
	public static void main(String args[])
	{
		// declare, allocate, and initialize Student object
		Student std = new Student(1234, "Anand","IT",89,90,91);
		float per=std.percentage();			//getting percentage
		std.show();
		System.out.println("Percentage:"+per);
	}
}
```

Output

Student ID: 1234
Student Name: Anand
Branch: IT
Percentage: 90

2.4.1 Constructor Overloading

In Java, it is possible to define two or more class constructors that share the same name, as long as their parameter declarations are different. This is called constructor overloading.

When an overloaded constructor is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded constructor to actually call. Thus, overloaded constructors must differ in the type and/or number of their parameters.

Example: OverloadCons.java

```
class Student
{
	int sid;
	String sname;
	String branch;
```

```
        int sub1, sub2, sub3;

        Student()               // This is the defaultconstructor
        {
                sid=501;
                sname= "Akhil";
                branch= "CSE";
                sub1=91;
                sub2=92;
                sub3=93;
        }
        // This is a parameterized constructor forStudent
        Student(int id, String name, String b, int s1 ,int s2, int s3 )
        {
                sid=id;
                sname=name;
                branch=br;
                sub1=s1;
                sub2=s2;
                sub3=s3;
        }
        public float percentage()
        {
                return (sub1+ sub2+sub3)/3;
        }
        public void show()
        {
                System.out.println("Student ID"+sid);
                System.out.println("Student Name: "+sname);
                System.out.println("Branch:"+branch);
        }
}
class OverloadCons
{
        public static void main(String args[])
        {
                // create objects using the various constructors
                Student std1 = new Student();
                Student std2 = new Student(1234, "Anand","IT",89,90,91);

                float per1=std1.percentage();           //Calculating CSE student percentage
                std1.show();
                System.out.println("Percentage:"+per1);

                float per2=std2.percentage();           //Calculating IT student percentage
                std2.show();
                System.out.println("Percentage:"+per2);

        }
}
```

Output
Student ID: 501
Student Name: Akhil
Branch: CSE
Percentage:92

Student ID: 1234
Student Name: Anand
Branch: IT
Percentage:90

2.5 GarbageCollection

Since objects are dynamically allocated by using the **new** operator, such objects are destroyed and their memory released for later reallocation. Java handles de-allocation for you automatically. The technique that accomplishes this is called ***garbage collection***. Java has its own set of algorithms to do this as follow. There are Two Techniques:

1. **Reference Counter**
2. **Mark andSweep**.

In the Reference Counter technique, when an object is created along with it a reference counter is maintained in the memory. When the object is referenced, the reference counter is incremented by one. If the control flow is moved from that object to some other object, then the counter value is decremented by one. When the counter reaches to zero (0), then it's memory is reclaimed.

In the Mark and Sweep technique, all the objects that are in use are ***marked*** and are called live objects and are moved to one end of the memory. This process we call it as compaction. The memory occupied by remaining objects is reclaimed. After these objects are deleted from the memory, the live objects are placed in side by side location in the memory. This is called copying.

It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. Garbage collection only occurs during the execution of your program. *The main job of this is to release memory for the purpose ofreallocation*.

The finalize( ) Method
Sometimes an object will need to perform some action when it is ***destroyed***. ***For example***, if an object is holding some non-Java resource such as a file handle or character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called ***finalization***. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

To add a *finalizer* to a class, you simply define the ***finalize( )*** method. The Java run time calls that method whenever it is about to recycle an object of that class. Inside the **finalize( )** method, you will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referencedobjects
The **finalize( )** method has this generalform:

```
protected void finalize( )
{
```

```
        // finalization code here
}
```

Here, the keyword **protected** is a specifier that prevents access to **finalize( )** by code defined outside its class.

2.6 statickeyword

- There will be times when you will want to define a class member that will be used independently of any object of that class. To create such a member, precede its declaration with the keyword**static**.
- The *static* keyword in java is used for memory managementmainly.
- We can apply java static keyword with variables, methods, blocks and nestedclass.
- The static keyword belongs to the class than instance of theclass.
- The static canbe:
    - variables (also known as classvariables)
    - methods (also known as classmethods)
    - block
    - nestedclass

The most common example of a **static** member is ***main()***. **main( )** is declared as **static** because it must be called before any objects exist.

Java static variable

- If you declare any variable as static, it is known staticvariable.
- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of studentsetc.
- The static variable gets memory only once in class area at the time of classloading.

Advantage of static variable

- It makes your program memory efficient (i.e it savesmemory).

Example:

```
class Student
{
        int rollno;
        String name;
        String college="VVIT";
}
```

Suppose there are 3000 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once. Java static property is shared to all objects.

```
class Student
{
    int rollno;
    String name;
    static String college ="VVIT";
```

```
    Student(int r, String n)
    {
        rollno = r;
        name = n;
    }

    void show( )
    {
        System.out.println(rollno+" "+name+" "+college);
    }
    public static void main(String args[])
    {
        Student s1 = new Student(1201,"Ajith");
        Student s2 = new Student8(1202,"Alekhya");
        s1.display();
        s2.display();
     }
}
```

Output:

```
1201    Ajith           VVIT
1202    Alekhya         VVIT
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
class Counter
{
        staticint count=0;      //will get memory only once and retain its value
        Counter()
        {
                count++;
                System.out.println(count);
        }
        public static void main(String args[])
        {
                Counter c1=new Counter();
                Counter c2=new Counter();
                Counter c3=new Counter();
        }
}
```

Output:1 2 3

Java static method

- If you apply *static* keyword with any method, it is known as *staticmethod*.
- A static method belongs to the class rather than object of aclass.
- A static method can be invoked without the need for creating an instance of aclass.
- Static method can access static data member and can change the value ofit.

There are three main restrictions for the static method. They are:

1. They cannot call **non-static**methods.
2. They cannot access **non-static**data.
3. They cannot refer to **this** or **super** in anyway.

Example of static method that calculates cube of a given number.

```
class Calculate
{
        static int cube(int x)
        {
             return x*x*x;
        }
        public static void main(String args[])
        {
                int result = Calculate.cube(5);
                System.out.println(result);
        }
}
```

Output: 125

Another example of static method that access static data member and can change the value of it

```
class Student
{
        int rollno;
        String name;
        static String college= "BBCCIT";      //static variable
        staticvoidchange()                    //static method
        {
                college = "VVIT";
        }

        Student(int r, String n)
        {
                rollno = r;
                name = n;
        }
        void display ()
        {
                System.out.println(rollno+" "+name+" "+college);
        }
        public static void main(String args[])
        {
                Student.change();
                Student s1 = new Student (1211,"Akshara");
                Student s2 = new Student (1222,"Chaitanya");
                Student s3 = new Student (1233,"Krishna");
                s1.display();
                s2.display();
```

```
            s3.display();
        }
}
```

Output:

```
1211   Akshara      VVIT
1222   Chaitanya    VVIT
1233   Krishna      VVIT
```

Java static block

- It is used to initialize the static datamember.
- It is executed before *main* method at the time ofclassloading.

Note: Static blocks are executed first, even than the static methods.

Example of static block

```
class A
{
        static int i;
        static
        {
                System.out.println("Static block is invoked");
                i=10;
        }
        public static void main(String args[])
        {
                System.out.println("Now in main");
                System.out.println("i:ow"+A.i);
        }
}
```

Output

```
static block is invoked
Now in main
I:10
```

2.7 *this*keyword

- Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines *this*keyword.
- **this** can be used inside any method to refer to the ***current object***. i.e. **this** is always a reference to the object on which the method was invoked.

Usage of java this keyword

1. *this* can be used to refer current class instancevariable.
2. *this* can be used to invoke current classmethod.
3. *this* can be used to invoke current classconstructor.

this: To refer current class instance variable

- The *this* keyword can be used to refer current class instancevariable.
- If there is ambiguity between the instance variables and parameters, *this* keyword resolves the problem ofambiguity.

Let's understand the problem with example given below:

```
class Student
{
	int sid;
	String sname;
	float fee;
	Student(int sid, String sname, float fee)
	{
		this.sid=sid;
		this.sname=sname;
		this.fee=fee;
	}
	void show()
	{
		System.out.println(sid+" "+sname+" "+fee);
	}
}
class Test
{
	public static void main(String args[])
	{
		Student std=new Student(1201,"Ankit",88000f);
		std.show();

	}
}
```

Output

```
1201	Ankit88000.0
```

this: To invoke current class method

- You may invoke the method of the current class by using the *this*keyword.
- If you don't use the *this* keyword, compiler automatically adds *this* keyword while invoking themethod.

Let's see the example

```
class A
{
	void m( )
	{
		System.out.println("hello m");
	}
	void n( )
	{
		System.out.println("hello n");
		this.m();
	}
}
class Test
{
	public static void main(String args[])
	{
```

```
                A a=new A();
                a.n();
        }
}
```

Output

hello n
hello m

this: To invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
class A {
        A(){

          }

System.out.println("hello a");


        A(int x) {
                this();
                System.out.println(x);
        }
}

class Test{
        public static void main(String args[]) {
                A a=new A(10);
        }
}
```

Output

hello a
10

2.9 Nestedclasses

- A class defined within another class; such classes are known as *nestedclasses.*
- The scope of the nested class (***inner class***) is bounded by the scope of it's enclosing class (***outerclass***).
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable andmaintainable.
- Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

```
class Java_Outer_class
{
        //code
```

```
        class Java_Inner_class
        {
                //code
        }
}
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1. Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class includingprivate.
2. Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one placeonly.
3. Code Optimization: It requires less code to write.

Types of Nested classes

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes.

1. Non-static nested class (innerclass)
    a. Member inner class
    b. Anonymous innerclass
    c. Local inner class
2. Static nestedclass

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.

Static inner class

- A static inner class is a nested class which is a static member of the outerclass.
- It can be accessed without instantiating the outer class, using other static members.Just like staticmembers.
- A static nested class does not have access to the instance variables and methods of the outerclass.
- The syntax of static nested class is as follows −

```
classMyOuter
{
        static class Mynested
        {       }
}
```

Example: Outer.java

```
public class Outer
{
        //inner static class
        static class Nested_Demo
```

```
        {
                public void my_method()
                {
                        System.out.println("This is my nested class");
                }
        }
        //body of outer class
        public static void main(String args[])
        {
                //without creating the object of outer class
                Outer.Nested_Demo nested = new Outer.Nested_Demo();
                nested.my_method();
        }
}
```

Note:

1. **static inner classes are rarely used inprograms.**
2. **If the static inner class is static method, we can directly call that method in the main() function without creating the object. As follow:Outer.Nested_Demo.my_method();**

Non-static inner class

- Creating an inner class is quite simple. You just need to write a class within aclass.
- An inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside theclass.

Following is the program to create an inner class and access it. In the given example, we make the inner class private and access the class through a method.

Example

```
public classOuter
{
        int outer_x=100;
        voidtest()
        {
                //creating the object of Inner class
                Inner inn=new Inner();
                inn.disp();
        }
        //inner non-static class
        private class Inner
        {
                public void disp()
                {
                        System.out.println("The value of x is:"+outer_x);
                }
        }
        public static void main(String args[])
        {
                //creating the out class object
                Outer2 out=new Outer2();
                out.test();
        }
}
```