



Class

- Classes have been used since the beginning of this book.
- Java is purely object oriented, so we can't think Java program without any class.
- The classes created in the preceding two chapters primarily exist simply to encapsulate the `main()` method, which has been used to demonstrate the basics of the Java syntax.
- Class is an implementation of the Encapsulation concept of OOP concept. Class is nothing but one kind of structure of data bind. Class is user-defined data type. A class defines the properties and behaviors for objects.
- Usually we create data members (fields) with private access specifier and data functions with public access specifier. But it's not compulsory.
- Thus, a class is a template for an object, and an object is an instance of a class. Because an object is an instance of a class, you will often see the two words object and instance used interchangeably.
- The General Form of a Class When you define a class, you declare its exact form and nature. You do this by specifying the data that it contains, and the code that operates on that data.
- A class is declared by use of the class keyword.

Syntax#

```
[access modifiers/non-access modifiers] class ClassName {  
    data members  
    data functions  
}
```

We can also add modifiers before the class declaration. Modifiers fall into two categories:

1. Access modifiers: public, protected, private.
2. Non-access modifiers: final, abstract, etc.

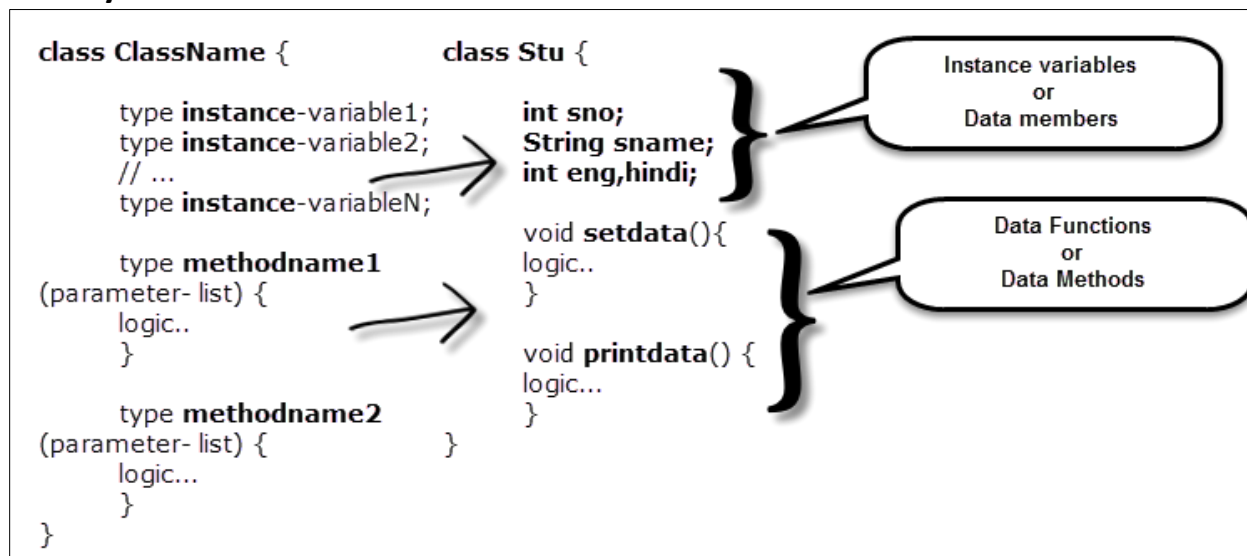
Example#

```
public class Stu{}
```

```
final class Emp{}
```

- A class is a collection of data members (instance variables) or data methods. In another way we can say class have two kinds of members - data members and data functions.





- 📄 The data, or say variables, defined within a class are called instance variables.
- 📄 The logic contained within methods. The logic contained inside any method is usually related with various operations on data members.
- 📄 To use classes in main() method we need to create an object.
- 📄 The object is the variable or say instance of the class. We can create as many instances of your class as we want.
- 📄 The size of the object depends on the total memory allocated by data members of the class. The class has never occupied memory class occupies memory.
- 📄 Usually with the help of class we divide big projects in small parts, Classes make modifications and improvements of large and complex programs possible.
- 📄 The concept of Objects & Classes helps you hide your code implementation from the user of your class.

Example# Basic example of a Class

Output View

```
E:\DemoJava>javac cls1.java  
  
E:\DemoJava>java cls1  
S1's sno = 5 sname = Ram
```

Coding View

```
class stu  
{  
    int sno;  
    String sname;
```



```
};

class cls1
{
    public static void main(String args[])
    {
        stu s1=new stu();

        s1.sno=5;
        s1.sname="Ram";

        System.out.println("S1's sno = " + s1.sno + " sname = " + s1.sname);
    }
}
```

In above example , we just create one class with two datamembers sno and sname. Unlike C++, we can direct access sno and sname data members in the main using object because the default access specifiers for class in java is public and that of C++ is private.

But according to the OOPS concept we need to create data functions to operate operation on data members and hide data of the class.

Example# Basic example of a Class with data members and data functions

Output View

```
E:\DemoJava>javac cls1.java

E:\DemoJava>java cls1
Enter sno =>1
Enter sname =>Ram
Enter sno =>2
Enter sname =>Vedika
No = 1 Sname = Ram
No = 2 Sname = Vedika
```

Coding View

```
import java.util.*;

class stu
{
    int sno;
    String sname;

    void setdata()
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter sno =>");
        sno=sc.nextInt();
    }
}
```



```
System.out.print("Enter sname =>");
sname=sc.next();
}

void printdata()
{
System.out.println("No = "+ sno + " Sname = "+ sname);
}

};







class cls1
{
public static void main(String args[])
{
stu s1=new stu();
stu s2=new stu();

s1.setdata();
s2.setdata();

s1.printdata();
s2.printdata();
}
}
```

Data members / Data Fields / Instance variable

- Data members are variable declared inside any class.
- Data members are also known as fields or instance variables or properties or attributes of the class.
- For example,

 Bank Class	 Student Class	 Traingle Class
 Acno CustomerName TypeofAccount Balance	 RollNo StuName Std Gender	 Height Width



- 📄 Data members are access by Data functions usually. Even we can call using object Name like if that data member declared as a public, objectName.DatamemberName=value
- 📄 The size of the object is decided by total number of data members that present in the class.
- 📄 In Java, We can initialize the data members directly at the time of declaration.
- 📄 We can also add modifiers before the data members declaration. Modifiers fall into two categories:
 1. Access modifiers: public, protected, private.
 2. Non-access modifiers: final,etc.

Syntax#

class className

```
{  
[access modifiers/non-access modifiers] datatype data member name;  
}
```

Example:

```
class student  
{  
    private int no;  
    String name="Ram";  
    final int x=5;  
}
```



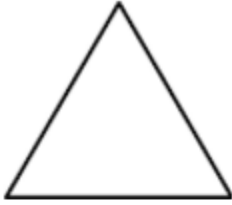
Data function / Member function / Method / Instance method

- 📄 Data function means function which is defined in the class.
- 📄 Usually we define Data functions as per requirement of operation defined in the class.

For example,





 Bank Class	 Student Class	 Traingle Class
Acno CustomerName TypeofAccount Balance	RollNo StuName Std Gender	Height Width
✓ OpenAccount() PrintAccount() Deposit() Withdrawn()	✓ SetStu() PrintStu() PrintResult()	✓ SetData() PrintData() PrintArea()

- The variables define in the data function known as local variable.
- Data function may or may not have return type and parameters . if no specific return type use return type as void.
- For returning the data from the data function the return keyword is used.

Syntax#

```
[access modifiers/non-access modifiers] returnType methodName([parameters])
{
  Logic....
}
```

- The difference between Java and C++ function here we declare each function with different access modifier.
- The rule of calling data function is the same as other programming, objectname.functionName().

Example:

```
class Bank
{
  int acno;
  String cname;
  void openAc()
  {
    Logic...
  }
}
```














```
void printAc()
{
    Logic..
}
void deposit()
{
    Logic..
}
void withdrawn()
{
    Logic...
}
}
```

Declaring Object

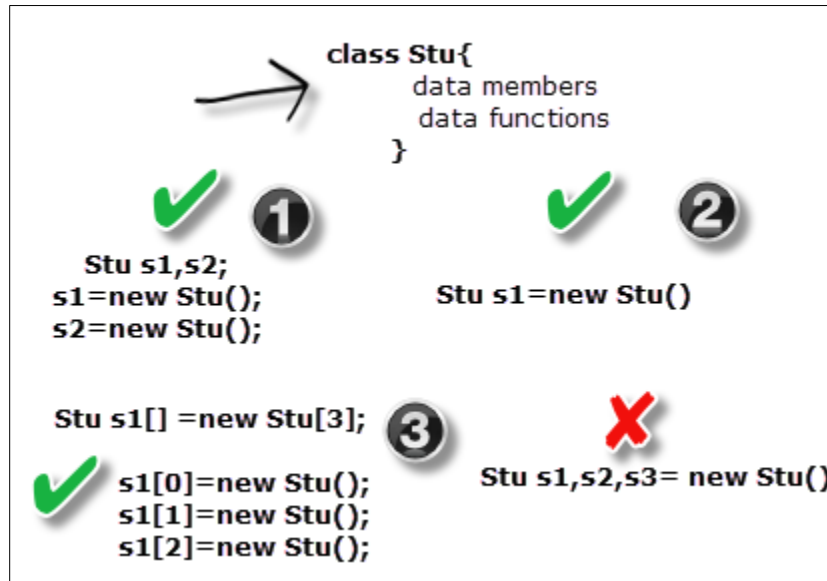
According to Object Oriented concepts,

-  An object represents an entity in the real world that can be distinctly identified.
-  Every object have unique identity, type, state and behavior.
-  For example, the relationship between classes and objects is analogous to that between a Bank and HDFC.
-  All the objects contained different values.

How to declare object of the class?

-  An object is an instance of a class. In other word we can say object is a variable of the class.
-  We can create many instances of a class. We can also declare object as an array. Object can be passed in the function's argument. Object can be used as return type of the function.
-  Creating an instance is referred to as instantiation.
-  The operator new is used to create an object of the class.
-  There are three ways to declare object.





Syntax#1

Stu s1; //declare reference to object

s1=new Stu(); // s1 gets instantiated for class Stu

The first line declare s1 as a reference to an object of type Stu. After this line executes, s1 contains the value null, which indicates that it does not yet to an actual object. We cannot access any properties of s1's object or cannot call any data function using s1 object because we are not allocating memory of the Stu class. Any attempt to use s1 at this point will result in a compile-time error.

```

stu s1;
s1.sno=5;

```

Without "new" and Compile time compiler generates an error like

variable s1 might not have been initialized

After the second line executes, we can use s1 as if it were a Stu object. The second line allocates an actual object and assigns a reference to it to s1.

Syntax#2

It is easy to declare objects individually ,

```

Stu s1=new Stu();
Stu s2=new Stu();

```

Here we declare and instantiated in single line only.



**Syntax#3**

```
Stu s1[] =new Stu[3];
```

In the above statement object declared as an array, it means which can hold references to three Stu objects. It doesn't initiate all the three Stu objects in the single step. For that we need to create three Stu objects individually and assign their reference to the array elements.

```
s1[0]=new Stu();  
s1[1]=new Stu();  
s1[2]=new Stu();
```

We can also use for-loop instead of assigning individual as given below,

```
for(i=0;i<3;i++)  
{  
s1[i]=new Stu();  
}
```



New Operator

The new operator in Java is responsible for the creation of new object.

The new operator dynamically allocates memory for an object and returns a reference to it. The dynamically allocation is just means that the memory is allocated at the run time of the program.

Syntax#

1) ClassName objName=new ClassName();

**2) ClassName objName;
objName=new ClassName();**

ClassName objName - This statement only creates a reference variable of a class. The reference object is pointed to the null.

objName = new ClassName() - The new operator than the actual or physical creation of the object is occurred. New Operator Firstly creates physical object in the heap with the variable pointing to the object from the stack and then it calls the constructor of the Class.

Object Reference

Each object has unique object reference. Each object have different memory allocation.

Example# Basic example of a Object Reference

Output View

```
E:\DemoJava>javac ObjRef.java

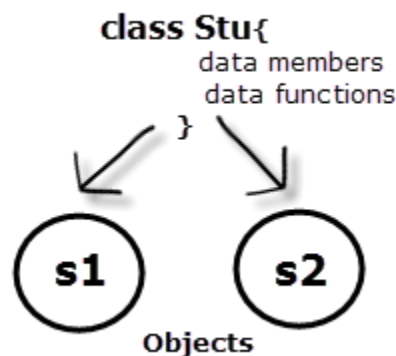
E:\DemoJava>java ObjRef
s1's Sno = 5 Name = Vedika
s2's Sno = 7 Name = Bhavin
```

Coding View

```
class Stu
{
int sno;
String sname;
}

class ObjRef {
public static void main(String args[])
{
Stu s1=new Stu();
s1.sno=5;
s1.sname="Vedika";

Stu s2=new Stu();
s2.sno=7;
```



```
s2.sname="Bhavin";

System.out.println("s1's Sno = " + s1.sno + " Name = " + s1.sname);
System.out.println("s2's Sno = " + s2.sno + " Name = " + s2.sname);

}
}
```

- When we use the concept of Object reference that time new object points to existing object.

Syntax#

```
class obj1;
class objref1;
objref1=obj1;
```

- It is very common to assign value of reference object to another object variable.
- Reference object points to existing variable.
- Reference object does not allocate extra memory.
- We can say it's an alternate name of existing object.
- Changes are reflected to each other.
- The value of a reference variable can be read in the same way as the value of an ordinary variable is read.
- A reference variable declared a final cannot ever be reassigned to refer to another object. The data within the object can be modified, but the reference variable cannot be changed.
- Let see one practical demo,

Example# Basic example of an Object Reference

Output View

```
E:\DemoJava>javac ObjRef2.java

E:\DemoJava>java ObjRef2
s1's Sno = 5 Name = Vedika
s1's Sno = 7 Name = Bhavin
s2's Sno = 7 Name = Bhavin
s1's Sno = 101 Name = Chirag
s2's Sno = 101 Name = Chirag
```

Coding View

```
class stu
{
int sno;
String sname;
}

class ObjRef2 {
```

```
public static void main(String args[])
{
    stu s1=new stu();
    stu s2;

    s1.sno=5;
    s1.sname="Vedika";

    System.out.println("s1's Sno = " + s1.sno + " Name = " + s1.sname);

    s2=s1;
    s2.sno=7;
    s2.sname="Bhavin";

    System.out.println("s1's Sno = " + s1.sno + " Name = " + s1.sname);
    System.out.println("s2's Sno = " + s2.sno + " Name = " + s2.sname);

    s2.sno=101;
    s2.sname="Chirag";

    System.out.println("s1's Sno = " + s1.sno + " Name = " + s1.sname);
    System.out.println("s2's Sno = " + s2.sno + " Name = " + s2.sname);
}
}
```

In above example, if there's a change in s1 then s2 change and there's a change in s2 then value of s1 will change, vice versa.

Accessing Class Members

- Class have two types of members, data functions (methods) and data members (fields).
- Class members are access by , Object Name . Member name
- The dot operator (.) is also known as a member access operator.

Example# Basic example of a Dot operator to access data members and data functions

Output View

```
E:\DemoJava>java MathUse
A = 22 B = 5
22 is max
Addition = 27
```

Coding View



```
class math
{
    int a,b;
    void max()
    {
        if(a>b)
        {
            System.out.println(a + " is max");
        }
        else
        {
            System.out.println(b + " is max");
        }
    }
    void add()
    {
        System.out.println("Addition = " + (a+b));
    }
}

class MathUse {
    public static void main(String args[])
    {
        math m1=new math();

        m1.a=22;
        m1.b=5;

        System.out.println("A = " + m1.a + " B = " + m1.b);

        m1.max();
        m1.add();
    }
}
```



Constructor

- 📄 Constructor is special member function of the class.
- 📄 The name of a constructor must be the name of the class.
- 📄 Constructor is used to initialize the values to the data member of the class.
- 📄 Constructor have no return type, even we cannot write even void before constructor.
- 📄 Constructor is called when object is created. Constructor can't be called explicitly.
- 📄 Constructor can be private, public or protected. Constructors cannot be abstract, final, native, static, or synchronized.
- 📄 Just like member functions, constructors can also be overloaded in a class.
- 📄 Constructor can be invoked called by its derived class.

Syntax#

[accessmodifier] constructorName ([parameters])

```
{  
//constructor logic  
}
```

ClassName ObjectName = new ClassName(); //call constructor

There are three types of it



Without Argument Constructor

- 📄 If we don't want to pass any value at the time of creation of the object and want to initialize something that time we can create without argument constructor.
- 📄 Constructor without parameter or say argument is known as without argument constructor.
- 📄 **Objects which call without argument constructor have same values in its field.**

Example# Demo of without argument constructor

Output View

```
E:\DemoJava>javac Con1.java
```

```
E:\DemoJava>java Con1
```

```
Sno = 5 Sname = Ram English = 22 Hindi = 33  
Sno = 5 Sname = Ram English = 22 Hindi = 33
```

Coding View

```
class stu
```

```
{
```

```
int sno;
```

```
String sname;
```

```
int eng,hindi;
```

```
stu() //without argument constructor
```

```
{
```

```
sno=5;
```

```
sname="Ram";
```

```
eng=22;
```

```
hindi=33;
```

```
}
```

```
void printData()
```

```
{
```

```
System.out.println("Sno = " + sno + " Sname = " + sname + " English = " + eng + " Hindi  
= " + hindi);
```

```
}
```

```
}
```

```
class Con1 {
```

```
public static void main(String args[])
```

```
{
```

```
stu s1=new stu(); //call
```

```
stu s2=new stu(); //call
```

```
s1.printData();
```

```
s2.printData();
```

```
}
```

```
}
```

With Argument Constructor

- With Argument Constructor also known as parameterized constructor.
- Parameterized constructors are required to pass parameters on creation of objects.
- Here all the objects have different unique distinct copies. Because they passed value at the time of calling parameterized constructor.



Example# Demo of with argument constructor

Output View

```
E:\DemoJava>javac Con2.java

E:\DemoJava>java Con2
Sno = 22 Sname = Bhavin English = 30 Hindi = 40
Sno = 33 Sname = Vedika English = 15 Hindi = 25
```

Coding View


```
class stu
{
    int sno;
    String sname;
    int eng,hindi;

    stu(int x,String y,int a,int b) //with argument
    {
        sno=x;
        sname=y;
        eng=a;
        hindi=b;
    }

    void printData()
    {
        System.out.println("Sno = " + sno + " Sname = " + sname + " English = " + eng + " Hindi
        = " + hindi);
    }
}

class Con2 {
    public static void main(String args[])
    {
        stu s1=new stu(22,"Bhavin",30,40); //call
        stu s2=new stu(33,"Vedika",15,25); //call
        s1.printData();
        s2.printData();
    }
}
```

Default Constructor

 If we don't define a constructor, then the compiler creates a default constructor. This generated constructor is called a default constructor.





- 📄 It's not visible in our code, but it's there just the same.
- 📄 Remember Default constructors are created only if there are no constructors defined by us. If we do write a constructor for our class, Java does not generate a default constructor.
- 📄 Default constructors initialize 0 to all the numeric and NULL to all the objects.

Example# Demo of Default (System generated) constructor**Output View**

```
E:\DemoJava>javac Con3.java

E:\DemoJava>java Con3
Sno = 0 Sname = null English = 0 Hindi = 0
Sno = 0 Sname = null English = 0 Hindi = 0
```

Coding View

```
class stu
{
int sno;
String sname;
int eng,hindi;

void printData()
{
System.out.println("Sno = " + sno + " Sname = "+ sname + " English = " + eng + " Hindi
= " + hindi);
}
}

class Con3 {
    public static void main(String args[])
    {
        stu s1=new stu(); //calls default constructor which is declared by Java itself
        stu s2=new stu(); //calls default constructor which is declared by Java itself
        s1.printData();
        s2.printData();
    }
}
```

Constructor overloading

Constructor is nothing but a special member function so it also supports the concept of constructor overloading.





When we declare more than one constructor and number of arguments or sequence of arguments or data type can be different that time we say it's an example of constructor overloading.

Compiler will decide which constructor is going to execute for the particular object.

Example# Demo of Constructor overloading**Output View**

```
E:\DemoJava>javac Con4.java

E:\DemoJava>java Con4
Sno = 5 Sname = Ram English = 22 Hindi = 33
Sno = 22 Sname = Mayur English = 50 Hindi = 70
Sno = 23 Sname = Rehan English = 20 Hindi = 30
```

Coding View

```
class stu
{
int sno;
String sname;
int eng,hindi;
stu() //without argument constructor
{
sno=5;
sname="Ram";
eng=22;
hindi=33;
}

stu(int x,String y,int a,int b) //with 4 argument
{
sno=x;
sname=y;
eng=a;
hindi=b;
}

stu(int x,String y) //with 2 argument
{
sno=x;
sname=y;
eng=20;
hindi=30;
}

void printData()
{
```





```

System.out.println("Sno = " + sno + " Sname = " + sname + " English = " + eng + " Hindi
= " + hindi);
}
}

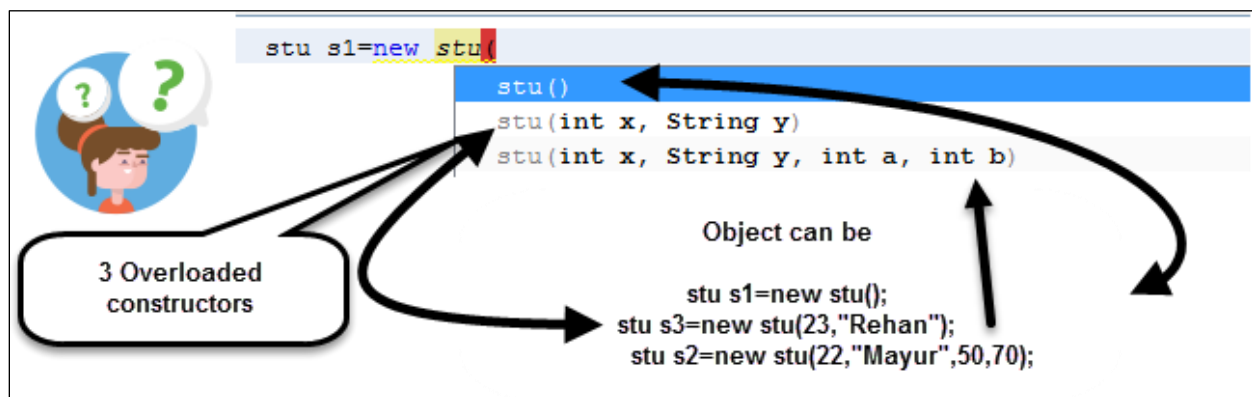
```

```

class Con4 {
    public static void main(String args[])
    {
        stu s1=new stu(); //without arguments
        stu s2=new stu(22,"Mayur",50,70); //with 4 arguments
        stu s3=new stu(23,"Rehan"); //with 2 arguments
        s1.printData();
        s2.printData();
        s3.printData();
    }
}

```

If we use netBeans Editor or Eclipse Editor to write Java Code, then we get options like given below while creating an object,



Difference between method and constructor

Constructors	Methods
Cannot be abstract, final, native, static, or synchronized.	Can be abstract, final, native, static, or synchronized.
No return type, not even void	void or a valid return type
Same name as the class	Any name except the class. Usually the name of an action.
Constructors are not inherited	Methods are inherited
If the class has no constructor, a default constructor is automatically supplied	Does not apply
The main use of constructor is used to finalize data members of the class.	Methods have own its logic.
Constructor calls automatically when object is created.	Methods need to call explicitly by object.



Example#

```
Class stu
{
Data members
Stu()
{
Logic...
}
Other Data functions()
}
```

Example#

```
Class stu
{
Data members
Constructor()
void setdata(){ Logic..}
void printdata(){ Logic..}
}
```

Method overloading

When we declare more than one function with same name but no. of arguments can be different, sequence of the arguments can be different or return type can be different.



Method overloading is an implementation of Polymorphism concept of OOP.

Method overloading is the practical implementation of compile time polymorphism. Static binding happens at compile time. Method overloading is also an example of static binding.

Overloaded methods are fast because they are bonded during compile time and no check or binding is required during runtime.

Method overloading increases the readability of the program.

Method overloading can appear in the same class. Overloaded method can have different access modifiers also.

When we call overloaded method that time it gives list as given below image:



Best example of method overloading is `System.out.println()` method which takes `String`, `int`, `float`, `double`, `char` or even object in output.

Example# Demo of Method overloading number of arguments can be different

Output View

```
E:\DemoJava>javac fun1.java  
  
E:\DemoJava>java fun1  
Sum of 2 = 55  
Sum of 3 = 15
```

Coding View

```
class math  
{  
    void add(int a,int b)  
    {  
        System.out.println("Sum of 2 = " + (a+b));  
    }  
    void add(int a,int b,int c)  
    {  
        System.out.println("Sum of 3 = " + (a+b+c));  
    }  
}  
  
class fun1 {  
    public static void main(String args[])  
    {  
        math m1=new math();  
        m1.add(22,33);  
        m1.add(4,5,6);  
    }  
}
```

Example# Demo of Method overloading Sequence of the arguments can be different

Output View

```
E:\DemoJava>javac fun2.java  
  
E:\DemoJava>java fun2  
Int and String A = 22 B = Ram  
String and Int A = Ritu B = 33
```

Coding View

```
class Demo  
{
```



```
void print(int a,String b)
{
System.out.println("Int and String A = " + a + " B = " + b);
}
void print(String a,int b)
{
System.out.println("String and Int A = " + a + " B = " + b);
}

}

class fun2 {
    public static void main(String args[])
    {
        Demo d1=new Demo();
        d1.print(22,"Ram");
        d1.print("Ritu",33);
    }
}
```

Example# Demo of Method overloading data type can be different

Output View

```
E:\DemoJava>javac fun3.java

E:\DemoJava>java fun3
Sum of numbers = 11
Concat = ABCDEF
```

Coding View

```
class Demo
{
    void add(int a,int b)
    {
        System.out.println("Sum of numbers = " + (a+b));
    }
    void add(String a,String b)
    {
        System.out.println("Concat = " + (a+b));
    }
}

class fun3 {
    public static void main(String args[])
    {
        Demo d1=new Demo();
        d1.add(5,6);
        d1.add("ABC","DEF");
    }
}
```



```
}
}
```

Issue of return type in Method overloading

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity.

The return type of method is not part of the method signature in Java. So return type of method doesn't matter while overloading a method.

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity as given below:

```
class math
{
    int add(int a,int b)
    {
        return a+b;
    }
    double add(int a,int b)
    {
        return a+b;
    }
}
```

Wrong example of function overloading

```
E:\DemoJava>javac fun4.java
fun4.java:7: error: method add(int,int) is already defined in class math
double add(int a,int b)
      ^
1 error
```

Overload main method

Yes we can overload main method also

Example# Demo of Method overloading data type can be different

Output View

```
E:\DemoJava>javac fun5.java

E:\DemoJava>java fun5
Default main() method invoked
main() method with 1 argument : 20
main() method with 2 arguments : Pal 21
```

Coding View

```
class fun5{

    public static void main(String a,int b)
    {
```





Shyamsir

JAVA

78 74 39 11 91

```
System.out.println("main() method With 2 arguments : " + a + " " + b);  
}
```

```
public static void main(int x){  
    System.out.println("main() method With 1 argument : " + x);  
}
```

```
public static void main(String args[]){  
    System.out.println("Default main() method invoked");  
    main(20);  
    main("Pal",21);  
}
```

```
}
```



Shyamsir

JAVA

78 74 39 11 91



Passing object in the Function / Object as Argument

When we want to give number of effects to instance variable of the object or add two objects or copy values of one object's value to another one, etc.

Passing an object as argument is as simple as passing variable to any function argument.

Here the data type of the argument is a className.

Syntax#

Returntype Functionname (ClassName objectName)

```
{  
Logic...  
}
```

Example#

```
1 void print(Stu s1)  
  {  
    Logic....  
  }  
2 void copy(Stu s1,Stu s2)  
  {  
    Logic..  
  }  
3 void findmax(Math m1,Math m2,Math m3)  
  {  
    Logic...  
  }
```

Let see practical demo

Example# Demo of Object as Argument, Addition of Two object's a and b values in third one object.

Output View

```
E:\DemoJava>javac objAs.java  
  
E:\DemoJava>java ObjAs  
Enter a =>11  
Enter b =>2  
Enter a =>20  
Enter b =>10  
A = 11 B = 2  
A = 20 B = 10  
A = 31 B = 12
```





Coding View

```
import java.util.*;

class Math {
    int a,b;
    void setdata()
    {
        Scanner s1=new Scanner(System.in);
        System.out.print("Enter a =>");
        a=s1.nextInt();
        System.out.print("Enter b =>");
        b=s1.nextInt();
    }
    void printdata()
    {
        System.out.println("A = " + a + " B = " + b);
    }

    void add(Math m1,Math m2)
    {
        a=m1.a+m2.a;
        b=m1.b+m2.b;
    }
}

class ObjAs
{
    public static void main(String[] args) {
        Math m1=new Math();
        Math m2=new Math();
        Math m3=new Math();

        m1.setdata();
        m2.setdata();

        m1.printdata();
        m2.printdata();

        m3.add(m1,m2);

        m3.printdata();
    }
}
```

The method **add()** have M1 and M2 object as arguments , and there is M3 object , using which the method add() is invoked. So M3 is implicitly passed. **So data member a and b**





Returning Object from Method

We can return an object from a method like normal value.

Usually when we want to return number of effects to particular object that time we can use concept of returning object.

Syntax#

ClassName Functionname (ArgumentList)

```
{  
Logic...  
}
```

Example#

```
1  Stu print()  
   {  
   Stu s1;  
   Logic...  
   return s1;  
   }  
2  Math findMax(Math m2)  
   {  
   if ...  
   return ....  
   else  
   return m2;  
   }
```

Example# Multiplication between two Object

Output View

```
E:\DemoJava>javac objAsReturn.java  
  
E:\DemoJava>java ObjAsReturn  
Enter a =>5  
Enter b =>10  
Enter a =>2  
Enter b =>3  
m1's A = 5 B = 10  
m2's A = 2 B = 3  
m3's A = 10 B = 30
```

Coding View

```
import java.util.*;
```





```
class Math {
int a,b;
void setdata()
{
Scanner s1=new Scanner(System.in);
System.out.print("Enter a =>");
a=s1.nextInt();
System.out.print("Enter b =>");
b=s1.nextInt();
}
void printdata(String objName)
{
System.out.println(objName + "'s A = " + a + " B = " + b);
}
}
```

Math mul(Math m2)

```
{
Math m3=new Math();
m3.a=a*m2.a;
m3.b=b*m2.b;
return m3;
}

}
class ObjAsReturn
{
    public static void main(String[] args) {
        Math m1=new Math();
        Math m2=new Math();
        Math m3=new Math();

        m1.setdata();
        m2.setdata();

        m1.printdata("m1");
        m2.printdata("m2");
        m3=m1.mul(m2);
        m3.printdata("m3");

    }
}
```

The method **mul()** has object M2 as argument , and there is M1 object which is implicitly passed because M1 is invoked mul() function. **So data member a and b is by default .**



This keyword

- Sometimes a method will need to refer to the object that invoked it.
- "this" keyword can be used inside any method to refer the current object.
- We can use the "this" keyword to refer to the object's instance members.

Syntax#

This

```
System.out.println(this);
```

Prints

classname = full qualified name

hashcode = hexadecimal format - hash code is an unique id number allocated to an object by JVM.

this . (this dot)

this.Data member name

```
System.out.println(this.a);
```

Use of this keyword

- 1) Using 'this' to get current class Names like clsMath

```
void getNameofTheClass() {  
    System.out.println("Class : " + this.getClass());  
}
```

- 2) Using 'this' keyword to pass current object as parameter.

```
btnAns.addActionListener(this);
```

- 3) Using 'this' keyword to return current Object.

This can be used to return current object in a method.

- 4) When argument name and data member name are same



This

Example# Basic Demo of this

Output View

```
E:\DemoJava>javac This1.java

E:\DemoJava>java This1
This = Math@2a139a55
This = Math@15db9742
This = Math@6d06d69c
M1's This = Math@2a139a55 A = 11 B = 2
M2's This = Math@15db9742 A = 12 B = 3
M3's This = Math@6d06d69c A = 13 B = 4
```

Coding View

```
class Math {
int a,b;
Math(int x,int y)
{
System.out.println("This = " + this);
a=x;
b=y;
}
void printdata(String objName)
{
System.out.println(objName + "'s This = " + this + " A = " + this.a + " B = " + b);
}
}
class This1
{
    public static void main(String[] args) {

        Math m1=new Math(11,2);
        Math m2=new Math(12,3);
        Math m3=new Math(13,4);

        m1.printdata("M1");
        m2.printdata("M2");
        m3.printdata("M3");

    }
}
```

In above program when M1 object is call constructor that time hashcode of M1 will be print and when it call printdata() function that time also hascode of M1 will be print. Same way M2 and M3. We can also refer data member with "this" as we used "this.a".

Example# Basic Demo of Max between two objects and Transfers to third object

Output View



```
E:\DemoJava>javac This2.java

E:\DemoJava>java This2
M1's  A = 11 B = 2
M2's  A = 2 B = 32
M3's  A = 2 B = 32
```

Coding View

```
class Math {
int a,b;

Math(int x,int y)
{
a=x;
b=y;
}

void printdata(String objName)
{
System.out.println(objName + "'s  A = " + a + " B = " + b);
}

Math max(Math m2)
{
    if(a+b<m2.a+m2.b)
    {
        return m2;
    }
    else
    {
        return this;
    }
}
}
class This2
{
    public static void main(String[] args) {
        Math m1=new Math(11,2);
        Math m2=new Math(2,32);
        Math m3=new Math(0,0);

        m1.printdata("M1");
        m2.printdata("M2");
        m3=m1.max(m2);
        m3.printdata("M3");
    }
}
```



Instance variable hiding using This

We know that declaring two local variables with the same name inside the same or enclosing scopes will generate a compile time error.

There can be situations where the instance variable names are same as the local variable name in a method.

In such cases, the local variables of the method overlap instance variable thereby hiding them.

While it is usually easier to simply use different names of the argument and instance variables.

When we want name of the parameters of the method and instance variable have same name we can differentiate data members name by this.name=name, means using "this" keyword else local variables overrides the value of instance variables.

Example# Wrong program without This ... and add "this" keyword then again run the program

Output View

```
E:\DemoJava>javac This3.java
E:\DemoJava>java This3
A = 0 B = 0
```

Coding View

```
class Math {
int a,b;

    Math(int a,int b)
    {
        a=a;
        b=b;
    }

void printdata()
{
    System.out.println("A = " + a
+ " B = " + b);
}
}
```

```
class This3
{
    public static void main(String[] args) {
```

```
Math(int a,int b)
{
    this.a=a;
    this.b=b;
}
```

Rewrite the program
with "this" keyword
in constructor
and check the output

```
E:\DemoJava>javac This3.java
E:\DemoJava>java This3
A = 11 B = 2
```




Shyamsir

JAVA

78 74 39 11 91

```
Math m1=new Math(11,2);  
m1.printdata();  
}  
}
```



Shyamsir

JAVA

78 74 39 11 91

Static Keyword

Java is using Static keyword with



Static variable / Class variable

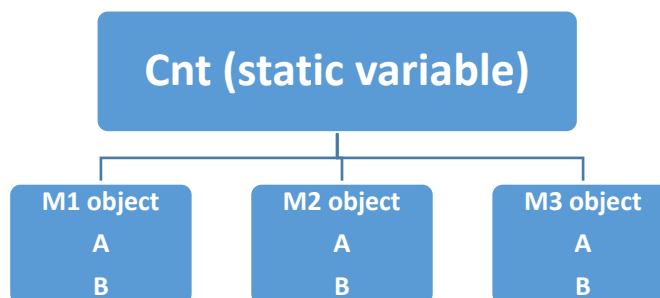
- A data member which declared with static is known as Static variable.
- Classes can contain static data member and static data functions.
- Static variables are called class variables.
- Static variables are normally used to maintain values common or global to the entire class.
- When we declare static, we are **telling the compiler that only one copy of that variable** will exist and that all objects of the class will share that variable. The static variable gets memory only once in class area at the time of class loading.

Syntax#

```
class ClassName
{
static datatype variableName;
}
```

Example#

```
class Stu
{
static int cnt;
}
```



- Only one copy of that member is created for the entire class and is shared by all objects of that class, no matter how many objects are created.

When to use static variable?

- Count the number of objects currently present
- Common data accessed by all objects

Static variable	Instance variable
Values are common for all the object	Values are different all the objects



Static data members are those whose memory space is created only once, whenever the class is loaded in the main memory.	Instance data members are those whose memory space is created each and every time whenever an object is created.
Static Data variables are also known as a class variable because the value of static variable is common for entire class.	Normal Data members are known as a Instance variable because the value of instance variables are vary from object to object.
We can use Static variable in non-static and static method.	We can use Instance variable only in non-static method.
Static data member access with its class name like Stu.CollegeName="Nirma"	Instance data member access with its object name like Stu1.Sname="Kahan"
Syntax# static data type variablename	Syntax# data type variablename

Example# Static variable

Output View

```

E:\DemoJava>javac Static1.java
E:\DemoJava>java Static1
Enter sno =>11
Enter Name =>Bhavin
Sno = 11 Sname = Bhavin Cnt = 1
Sno = 0 Sname = null Cnt = 1
Enter sno =>22
Enter Name =>Mayur
Sno = 11 Sname = Bhavin Cnt = 2
Sno = 22 Sname = Mayur Cnt = 2

```

Value of Instance variables are unique

Value of Static variable is common for all the object

Coding View

```
import java.util.*;
```

```

class Stu {
int sno;
String sname;
static int cnt;

void setdata()
{
Scanner sc=new Scanner(System.in);

System.out.print("Enter sno =>");
sno=sc.nextInt();

```





```

        System.out.print("Enter Name =>");
        sname=sc.next();

        cnt++;
    }
    void printdata()
    {
        System.out.println("Sno = " + sno + " Sname = " + sname + " Cnt = " + cnt);
    }
}
class Static1
{
    public static void main(String[] args) {
        Stu s1=new Stu();
        Stu s2=new Stu();

        s1.setdata();
        s1.printdata();
        s2.printdata();

        s2.setdata();
        s1.printdata();
        s2.printdata();
    }
}

```

Static method

- 📄 A static method belongs to the class rather than object of a class.
- 📄 A static method can be invoked without an object.
- 📄 In C++ we cannot call static method only using ClassName.FunctionName() but in the case of Java we can call either with the ClassName.FunctionName() or ObjectName.FunctionName()

Syntax#

```

class ClassName
{
    static returnType FunctionName()
    {
        Logic..
    }
}

```

To call

ClassName.FunctionName()

Example#

```

class Stu
{
    static void printCnt()
    {
        Logic...
    }
}

```

ToCall

Stu.printCnt()



- 📄 The static methods can be used to access a static data member
- 📄 We can access static variable in non-static data function, but we cannot access non-static data member in side any static function. The reason behind this logic is static methods are associated with a class, not an object.
- 📄 A static function can used to access only static members (function or variables) declared in the same class.
- 📄 Java static methods cannot refer this and super.
- 📄 The main method of Java is static, Java virtual Machine can call it without creating any instance of a class which contains the main method. For that reason we cannot call non-static method inside main() function.

Example# Demo of Static Method

Output View

```
E:\DemoJava>javac Static2.java  
  
E:\DemoJava>java Static2  
Add = 7  
Div = 5
```

Coding View

```
class Static2  
{  
  
    public static void main (String args [ ] )  
    {  
        add(5,2);  
        div(100,20);  
    }  
  
    static void add(int a,int b)  
    {  
        System.out.println("Add = " + (a+b));  
    }  
  
    static void div(int a,int b)  
    {  
        System.out.println("Div = " + (a/b));  
    }  
  
}
```

Example# Demo of Static Method which count how many objects call set data

Output View

```
E:\DemoJava>javac Static3.java

E:\DemoJava>java Static3
Total set objects are Cnt = 1
Total set objects are Cnt = 2
sno = 11 Sname = Dipesesh
sno = 12 Sname = Bhagirath
sno = 13 Sname = Saurin
Total set objects are Cnt = 3
```

Coding View

```
class clsStudent
{
    int sno;
    String sname;
    static int cnt;

    void setdata(int x,String y)
    {
        sno=x;
        sname=y;
        cnt++;
    }

    void printdata()
    {
        System.out.println("sno = " + sno + " Sname = " + sname);
    }

    static void printTotalObject()
    {
        System.out.println("Total set objects are Cnt = " + cnt);
    }
}

class Static3
{
    public static void main (String args [ ] )
    {
        clsStudent.printTotalObject();

        clsStudent s1=new clsStudent();
        clsStudent s2=new clsStudent();
        clsStudent s3=new clsStudent();

        s1.setdata(11,"Dipesesh");
        clsStudent.printTotalObject();

        s2.setdata(12,"Bhagirath");
        s3.setdata(13,"Saurin");
```

```
s1.printdata();  
s2.printdata();  
s3.printdata();
```

```
clsStudent.printTotalObject();  
}  
}
```

If we try to write non-static variable inside any static function then,

```
E:\DemoJava>javac Static3.java  
Static3.java:22: error: non-static variable sno cannot be referenced from a  
static context  
System.out.println("Total set objects are Cnt = " + cnt + " Sno = " + sno);  
1 error
```

Static block

- Static blocks are also called Static initialization blocks.
- The static block, is a block of statement inside a Java class that will be executed when a class is first loaded in to the JVM
- Static blocks will be called only once.

Syntax#

```
static {  
    Logic...  
}
```

- A class can have any number of static initialization blocks and they are called in the order that they appeared in the class.
- JVM combines all these blocks into one single static block and then executes.
- Static block is called before the execution of the constructor

When to use?

- When we need to do computation in order to initialize.
- Start some background before object creation.

Example# Demo of Static Block

Output View



```
E:\DemoJava>javac Static4.java
E:\DemoJava>java Static4
Hi in 1
Hi in 2
Hi in 3
Cons1
Cons1
```

 Static Block

Coding View

```
class Emp
{
    int eno;
    String ename;
    static
    {
        System.out.println("Hi in 1");
    }
    Emp(int x,String y)
    {
        System.out.println("Cons1");
        eno=x;
        ename=y;
    }
    static
    {
        System.out.println("Hi in 2");
    }
    void printdata()
    {
        System.out.println("Eno = " + eno + " Ename = "+ ename);
    }
    static
    {
        System.out.println("Hi in 3");
    }
}

class Static4
{
    public static void main (String args [ ] )
    {
        Emp e1=new Emp(1,"Kirtan");
        Emp e2=new Emp(22,"Saumil");
    }
}
```



}

Garbage Collector and Finalize() method

Garbage Collector

- In Java, garbage collector(GC) provides an automatic solution to memory management.
- Garbage means unreferenced objects and the process of removing the object from a running program is known as garbage collection.
- In C language, it is the coder's responsibility to de-allocate memory allocated dynamically using free() function. Same way, In C++ language, it is the coder's responsibility to de-allocate memory allocated dynamically using destructor. But Java does not support destructor or free() function. In Java it is performed automatically by JVM with the help of Garbage Collector(GC).
- The Garbage Collector runs repeatedly, checking for objects that are no longer reference and destroyed it.
- In Java, objects are dynamically allocated by using the new operator.
- An object once created using some memory and the memory remains allocated till there are references for the use of the object.
- Java destruction of objects from memory is done automatically by the JVM.



Garbage Collector(GC)

- When there is no reference to an object, then that object is assumed to be no longer needed and the memory occupied by the object are released. Garbage Collector accomplishes de-allocate memory no reference objects and avoiding memory leaks. In other words, Garbage Collector is a way to destroy the unused objects. Garbage Collection is the process of reclaiming the runtime unused memory automatically.
- Garbage collector frees the memory and this memory can be used by other programs or the same program further in its execution.



Finalize() method

- finalize() method is already defined in java.lang.Object class. It is a protected and non-static method of java.lang.Object class.
- This method will be available in all objects we create in Java.



- 📄 Sometimes we need to perform some actions before destructing an object. We can specify those actions that will execute when an object is just about to be destructed. We can do this things with the help of finalize() method.
- 📄 finalize() method is used to perform some final operations or cleanup operations on an object before it is removed from the memory.
- 📄 The finalize() method is invoked each time before the object is garbage collected.
- 📄 If we have created any object without new, we can use finalize method to perform cleanup processing.
- 📄 We can override the finalize() method to keep those operations we want to perform before an object is destroyed.

Syntax#**protected void finalize()****{}**

It is defined as protected in the super class and that's the reason the overridden method is having the protected access modifier.

It is not guaranteed for that Finalize () method is going execute even we declared in our class.

Example# Demo of finalize()**Output View**

```
E:\DemoJava>javac finalizeMethod.java
Note: finalizeMethod.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

E:\DemoJava>java finalizeMethod
Cons1
Eno = 1 Ename = Kirtan
This is finalize() method
```

Coding View

```
class Emp
{
    int eno;
    String ename;
    Emp(int x,String y)
    {
        System.out.println("Cons1");
        eno=x;
        ename=y;
    }

    void printdata()
    {
```





```
        System.out.println("Eno = " + eno + " Ename = "+ ename);
    }

    protected void finalize()
    {
        System.out.println("This is finalize() method");
    }
}

class finalizeMethod
{
    public static void main (String args [ ] )
    {
        Emp e1=new Emp(1,"Kirtan");
        e1.printdata();
        Runtime.runFinalizersOnExit(true); // this method is required because execution of
        finalize() method is not guaranteed, we are forcing to call it
    }
}
```

Access control and modifiers

Access controls are also known as Visibility Specifiers or Access Specifiers.

Access controls means rights of access. Access control is a way of limiting access data members or data functions or constructors.

When we are talking about Access control it comes with the topic packages, subclasses and same class. We will see these topics in next chapters. Access control act on package boundaries.

Access controls are determine whether a data members or data functions or constructor in a class, can be used or invoked by sub-class or another method in another class like main() method.

We cannot declare class/interface with private or protected access specifiers.

```
protected class stu
{
    modifier protected not allowed here
    ----
    (Alt-Enter shows hints)
```

There are four different modifiers





1	Private	2	Protected
3	Public	4	Default

Access Modifiers	Same Class	Same Package	Subclass	Other packages
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default (no access modifiers)	Y	Y	N	N
private	Y	N	N	N

For better understanding, member level access is formulated as a table given above. It means If we declared public data function in Class A that can be accessed by other method of same class ,same package , sub class (inherited class) , method of the other package.

Private

Private means nowhere.

We can access private data from within the class only in which it is defined. We can't access it from outside of the class.

Example#

```
class Emp {
    private int rno ;
    private void setdata()
    {
        Logic...
    }
}
```

Protected

Protected means somewhere



📄 We can access protected data from within the class only in which it is defined and the derived class. We can't access it from outside of the current package.

📄 Protected access specifier comes into picture when inheritance is implemented.

Example#

Class A

```
{  
protected int x;  
}
```

Class B extends A

```
{  
private int y;  
private void add()  
{  
    System.out.println(x+y);  
}  
}
```

Public

📄 Public means anywhere

📄 We can access public data from any where, whether these classes are in the same package or in another package.

📄 It can also be accessed by the classes of other packages also.

📄 We can have only one public class in a program and the file name should be same as the public class name.

Example#

```
public class Emp {  
    public int eno = 10;  
    public void setdata()  
    {  
        Logic...  
    }  
}
```

Default (no specifier)

📄 Default is not that one which is associated with switch statement.

📄 We cannot write default keyword.

📖 If we don't not specify any keyword before class , data member , data function and constructor that time it consider as a default.

📖 Default data can be access by within the same class or in their sub classes.

Example#

```
class math
{
int a,b;
void setdata(){Logic...}
void printdata(){Logic...}

}
```

Nested Class

We can declare a class within another class; such classes are known as nested classes.

An inner class, or nested class, is a class defined inside other class and act like a member of the enclosing class.

Syntax#

```
class OuterClass {
....
    class NestedClass {
        ....
    }
}
```

Example#

```
class A {
.....
    class B{
        ....
    }
}
```

In above Example, class B is considered as a nested class and class A is considered as an outer class.

Example# Demo of NestedClass Triangle

Output View

```
E:\DemoJava>javac jNested.java

E:\DemoJava>java jNested
Area of Triagnle = 10000.0
```

Coding View

```
class Polygon
{
double h,w;

void findarea()
{
```

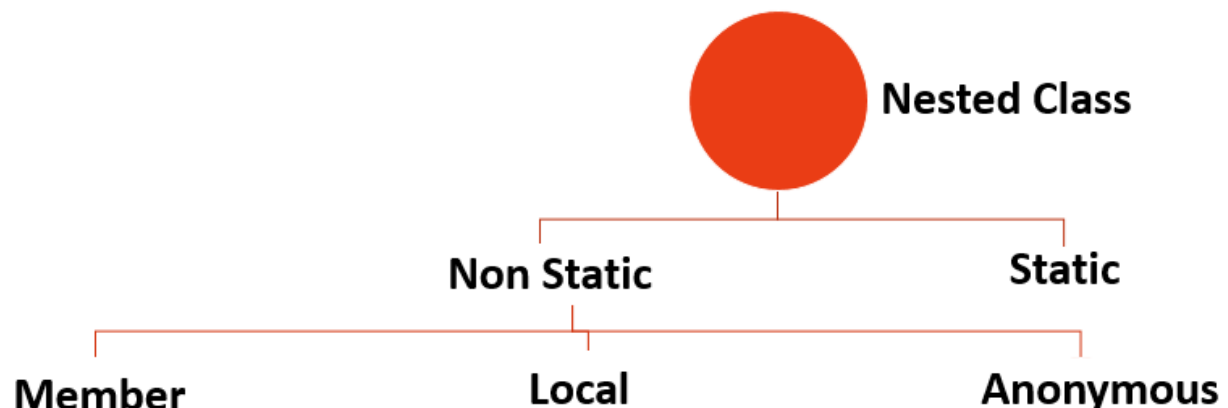
```
Triangle t1=new Triangle();
t1.area();
}

class Triangle //Nested Class
{
    Triangle()
    {
        h=100;
        w=200;
    }
    void area()
    {
        System.out.println("Area of Triagnle = " + (0.5*h*w));
    }
}

public class jNested {
    public static void main(String args[])
    {
        Polygon p1=new Polygon(); //Outer class
        p1.findarea();
    }
}
```

In above program, Polygon is the outer class which contains Triangle as a nested class. The class Polygon has on function findArea() which invokes the constructor of class Triangle and then function. The class Triangle defines a constructor and function area() which invokes the data members of class Polygon. That means inner class can access the members of the outer class.

There are two types of nested classes: static and non-static




Nested classes that are declared static are called static nested classes.



Nested classes that are declared without static are called inner classes. Inner class further divides into Member inner class, Local inner class, and Anonymous inner class.

Some characteristics of Nested Class

Member inner class

 Member Inner class is class created within class. Inner member class has rights to access all data members and data functions methods of Outer class. But outer class cannot directly access members of Inner class.

Syntax#

```
public class Outer {
    ....
    public class Inner {
        ....
    }
}
```

Example#

```
class A {
    .....
    class B{
        ....
    }
}
```

Object Creation

```
Outer ot=new Outer();
Outer.Inner in= ot.new Inner();
```

Please notice how we put new after the reference to the outer class in order to create an instance of the inner class

Example# Demo of MemberInner Class

Output View

```
E:\DemoJava>javac jNested2.java
E:\DemoJava>java jNested2
X : 10
```

Coding View

```
class Outer{

    int x=10;

    class Inner{
        void printdata(){
            System.out.println("X : "+x);
        }
    }

}

class jNested2
```




```
{
public static void main(String args[]){
    Outer o1=new Outer();
    Outer.Inner in=o1.new Inner();
    in.printdata();
}
}
```

Local inner class

- An inner class that is defined within a block like method.
- Local Inner class is class created within the method. Inner member class has rights to access all data members and data functions methods of Outer class. To invoke the methods of local inner class, we must instantiate this class inside the method.

Syntax#

```
public class Outer {
    public void MethodName()
    {
        class Local {
            Logic...
        }
        Local local = new Local();
    }
}
```

Example#

```
public class Outer {
    public void print()
    {
        class Local {
            Logic....
        }
        Local local = new Local();
    }
}
```

Object Creation

```
Outer ot=new Outer();
```

Example# Demo of Local Inner Class

Output View

```
E:\DemoJava>javac jNested3.java
E:\DemoJava>java jNested3
X : 10
```

Coding View

```
class Outer{
    private int x=10;
```

```
void printdata(){
    class Local{
        void localprint(){
            System.out.println("X : "+x);
        }
    }

    Local l1=new Local();
    l1.localprint();
}

class jNested3
{
    public static void main(String args[]){
        Outer obj=new Outer();
        obj.printdata();
    }
}
```

Static inner class

A class which declared as a inner class with static keyword.

Non-static variable and methods of the outer class cannot be referenced from an inner static class.

Only static members of the outer class are accessible to the static inner class.

Syntax#

```
public class Outer {
    public static class Nested
    {
        Logic...
    }
}
```

Example#

```
public class A {
    public static class B
    {
        Logic...
    }
}
```

Object Creation

Outer.Nested instance = new Outer.Nested();

Example# Demo of Static inner Class

Output View

```
E:\DemoJava>javac jNested1.java
E:\DemoJava>java jNested1
X : 10
```

Coding View

class Outer

```
{
static int x=10;

    static class staticInner{
        void printdata(){
            System.out.println("X : "+x);
        }
    }
}

class jNested1 {
    public static void main(String args[])
    {
        Outer.staticInner i1=new Outer.staticInner();
        i1.printdata();
    }
}
```

Anonymous inner class

- Anonymous class is a nested class without a class name.
- It combines the class declaration and the creation of an instance of the class in one step.
- A class without a name and implements exactly only one interface or exactly extends one abstract class.
- Object of an anonymous class can be instantiated from within the same scope in which it is defined.
- It is commonly used in Android application or Applet application or Swing application.

Syntax#

```
public static void main(String args[])
{
    AnonymousInner inner = new
    AnonymousInner() {
        public void methodName(){
            .....
            .....
        }
    }
```

Example#

```
button.addActionListener(new
    ActionListener() {
        public void
        actionPerformed(ActionEvent e){
            }
    });
```



```
};  
}
```

Object Creation

Anyonymousclass.functionName();

Example# Demo of Static inner Class

Output View

```
E:\DemoJava>javac jNested4.java  
  
E:\DemoJava>java jNested4  
Sum = 11
```

Coding View

```
abstract class Math{  
    abstract void add(int x,int y);  
}
```

```
class jNested4  
{  
    public static void main(String args[]){
```

```
        Math m1=new Math(){  
            void add(int a,int b)  
            {  
                System.out.println("Sum = " + (a+b));  
            }  
        };  
        m1.add(5,6);  
    }  
}
```

