**Nested Class:-**

1) **It is a way of logically grouping classes and interfaces that are only used in one place**.

**2**) **It increases encapsulation**.

Nested classes represent a special type of **relationship that is it can access all the members (data members and methods) of outer class including private**.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

**Basic :**

1.If the Inner Class is static Inner Class then ‘no need to create object of Outer Class. Syntax is:

**Outer.Inner obj=new Outer.Inner();**

2. If the Inner Class is Instance Inner Class then ‘ need to create object of Outer Class. Syntax is:

**Outer out=new Outer();**

**Outer.Inner obj= out.new inner();//new must be out.new**

**We can combine**

**OuterClass.InnerClass innerObject = new OuterClass().new InnerClass();**

**Rules of the Inner classes in Java:-**

1. In Java it is possible to define a class inside another class such classes are called nested classes or inner class.
2. There are 3 types of the Inner classes 1.instance Inner class, 2.static inner class and 3.anonymous inner class
3. If the Inner class is declared as instance inner class then it can access all of the members of the outer enclosing class including **private** members
4. If the Inner class is declared as static then it can only access the **static** members of the outer class (including the **private** **static** members). But it can NOT access the instance members
5. Consider that there is a variable x is defined in both the outer class and the instance inner classes then general form for accessing the variable from the inner class is this.x for the inner x and OuterClassname.this.x for the outer x.
6. You can also define an inner class **inside any method** or any **other** **block**.
7. The general form for instantiating the inner class from outside the outer class is Outer.Inner ob = new Outer.new Inner();
8. The general form for instantiating the inner class from outside the outer class is (if the inner class is declared as static) Outer.Inner ob = new Outer.Inner();
9. The Inner classes can be declared with any of the access modifier keywords.
10. **We can declare an inner class as private but we can not declare an outer class as private.**If the Inner class is declared as private then it can NOT be instantiated from outside the outer class. Also in this case you can NOT access the members of the Inner class from outside the outer class even you have an object reference and even if the members of the private inner class are declared as public. **Private Class cannot be access outside the class even though not in outer class.** Once you declare an inner class private, it cannot be accessed from an object outside the class.
11. If the Inner class is declared as instance inner class then it can also access the superclass members of the outer class through the general statement Outer.super.variable; Outer.super.method(params);

**Program:-1-A**

**class NestedClass**

{

int x=20;

void print(int x)

{

x=x;

System.out.println("Value is "+this.x); //it refer class variable

}

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

{

NestedClass obj=new NestedClass();

obj.print(10);

}

}

**Output:-** Value is 20

**Program:-1-B**

class NestedClass

{

int x=20;

void print(int x)

{

x=x;

System.out.println("Value is "+x);// It refer as ‘ parameter varibe’.

}

public static void main (String []args)

{

NestedClass obj=new NestedClass();

obj.print(10);

}

}

**Output:-** Value is 10

Inner classes are of **three** types depending on how and where you define them. They are −

* 1.1 Inner Class(**require object of outer class**)

1.2 Method-local Inner Class(**must initiate within method**)

A method-local inner class can be instantiated only within the method where the inner class is defined.

* 2. Anonymous class: It should be used if you have to override method of class(can be **Abstract** or **Concreate**) or **interface.**

In case of anonymous inner classes, we declare and instantiate them at the same time.

* 3. Static Inner Class(only access static member)

### **Inner Class**

class Outer\_Demo {

private int num = 175;

public class Inner\_Demo

{

public int getNum()

{

System.out.println("This is the getnum method of the inner class");

return num;

}

}

}

public class My\_class2 {

public static void main(String args[]) {

// Instantiating the outer class

**Outer\_Demo outer = new Outer\_Demo();**

// Instantiating the inner class

**Outer\_Demo.Inner\_Demo inner = outer.new Inner\_Demo();**

System.out.println(inner.getNum());

}

}

If you compile and execute the above program, you will get the following result −

**Output**

This is the getnum method of the inner class: 175

## **Method-local Inner Class**

A method-local inner class can be instantiated only within the method where the inner class is defined.

public class Outerclass

{

void my\_Method()

{

int num = 23;

class MethodInner\_Demo

{

public void print()

{

System.out.println("This is method inner class"+num);

}

}

MethodInner\_Demo inner = new MethodInner\_Demo();

inner.print();

}

public static void main(String args[])

{

Outerclass outer = new Outerclass();

outer.my\_Method();

}

}

If you compile and execute the above program, you will get the following result −

**Output**

This is method inner class 23

## **Anonymous Inner Class (can be abstract)**

**In case of anonymous inner classes, we declare and instantiate them at the same time.**

Generally, they are used whenever you need to override the method of a class or an interface.

In the following way, you can override the methods of the concrete class as well as the interface using an anonymous inner class.

[Live Demo](http://tpcg.io/R6PaWE)

**abstract** class AnonymousInner {

public abstract void mymethod();

}

public class Outer\_class {

public static void main(String args[]) {

**AnonymousInner inner = new AnonymousInner**() {

public void mymethod() {

System.out.println("This is an example of anonymous inner class");

}

};

inner.mymethod();

}

}

If you compile and execute the above program, you will get the following result −

**Output**

This is an example of anonymous inner class

**Anonymous Class(Interface)**

**interface** Eatable{

**void** eat();

}

**class** TestAnnonymousInner1{

**public** **static** **void** main(String args[]){

 Eatable e=**new** Eatable()

{

**public** **void** eat(){System.out.println("nice fruits");}

 };

 e.eat();

 }

}

Output: nice fruits

**Static Inner Class**

**class** TestOuter1{

**static** **int** data=30;

**static** **class** Inner{

**void** msg(){System.out.println("data is "+data);}

  }

**public** **static** **void** main(String args[]){

  TestOuter1.Inner obj=**new** TestOuter1.Inner();

  obj.msg();

  }

}