**Nested Classes**

A nested class is one that is declared entirely in the body of another class or interface. Interface is a type of class that defines only abstract methods

The class, which is nested, exists only as long as the enveloping class exists. Therefore, the scope of inner class is limited to the scope of enveloping class.

There are four types of nested class.

Nested static class is like any other static member of the enveloping class

Nested non-static classes are also called inner classes, which may be

(a) member inner class,

(b) local class, and

(c) anonymous class.

This inner class is a member of the enveloping class just like any other member of the enveloping class, and therefore, it has access to all the members of the enveloping class including the private members. However, the enveloping class does not have direct access to members of the nested (inner) class.

The following points regarding non-static nested classes should be noted.

1. The inner class has access to all the members of the enveloping class including the members declared public, protected, or private.

2. The enveloping class does not have direct access to nested class members. It is only through the reference of inner class.

3. An instance of inner class outside the enveloping class can only be created with an instance of outer class.

4. The object of inner (nested) class may be declared only in the scope of outer (enveloping) class.

5. A method of inner class cannot be directly accessed by the object of outer class.

The access is through the object of inner class by a fully qualified name for accessing, which is given in the following example. Here, objectOuter and objectInner are objects of outer and inner classes, respectively.

objectOuter.objectInner.methodName();

**1.**Example for **Nested static**

class Outer1

{ // Enveloping class

static private double length;// declaration of variables

static private double width;

Outer1( double x, double y)// constructor

{length = x; width = y; }

double area (){ return length\* width; }

static class Inner2// nested class

{

private double rate;

Inner2(double r ){ rate = r;}

void displayInner()

{

System.out.println ("The cost of fencing = " + rate\*2\* (length + width));

}

}// End of class Inner2

}// End of class Outer1

// Below is class with main method public

class NestedStatic

{

// class with main method

public static void main (String args [])

{

Outer1.Inner2 inn1 = new Outer1.Inner2(10);// object

Outer1 Obj = new Outer1 (40,30);

System.out.println ("Obj\_Area = " + Obj.area());

inn1.displayInner();

}

}

**2.**Example for **Member inner class**

class Outer

{

private double length;// declaration of variables

private double width;

Outer (double x, double y)// Constructor method

{length = x; width = y;}

double area(){ return length\* width;}// definition of method

void displayOuter()

{

Inner iner = new Inner();// iner is object of inner class

iner.displayInner();

}

class Inner

{

double rate = 10.0;// declaration of variables

double perimeter = 2\*( length + width);

double fencingCost = rate \* perimeter;

void displayInner()

{// definition of another method

System.out.println ("The cost of fencing = " + fencingCost);

}

}

}

class NEST

{

public static void main (String args [])

{

Outer obj = new Outer (40.0, 20.0);

obj.displayOuter();// accessing the method

System.out.println ("Area of Obj = " + obj.area());

}

}

**3.**Example for **local class**

public class LocalDemo

{

public static void main(String[] args)

{

class Local

{// local class defined

private int x;// declaring instance variable

Local(int a) {x = a;}// constructor of local class

public void display()

{

System.out.println("Cube of "+x+" = " + Math.pow(x,3));

}

}// End of local class

Local local = new Local(20);// creating an object

local.display();

}

}

**4.**Example for **Anonymous classes**

As the name indicates, anonymous classes are inner classes without a name. It is defined inside another class. Because class has no name it cannot have a constructor method and its objects cannot be declared outside the class. Therefore, an anonymous class must be defined and initialized in a single expression. An anonymous class may be used where the class has to be used only once.

new class-name ([argument list]){class body}

Illustration of anonymous class implementing an interface

interface InterFace// interface definition

{

public void display();// abstract method

}

public class AnonymousDemo1

{

public static void main (String args[])

{//inner anonymous class definition

InterFace f = new InterFace ()

{// creating reference of interface

public void display()

{

System.out.println( "Anonymous class implements interface.");

}

};// The anonymous implementor of Interface closes

f.display(); //The reference f calls the method display()

}

}

Illustration of an anonymous class extending an abstract class

abstract class Person

{// abstract class declared

abstract void display();// abstract method declared

}

class AnonymousDemo2

{

public static void main (String args[])

{

Person person = new Person() {// creating an object

void display()

{

System.out.println("My name is John.");

}

};// anonymous class closes

person.display();

} //The main class closes

}// class AnonymousDemo2 closes