**Inner Classes:**

**A Regular Inner Class:**

class MyOuter

{

class MyInner

{

}

}

Now, if you compile it,

**Two class file will be generated.**

MyOuter.class

MyOuter$MyInner.class

Now, when you try to run it jvm will obviously follow the necessary steps. (loading, linking and initialization). But, youn can not pass the inner class’s name as argument to the java command (which invokes the jvm). Because, it is a regular inner class and does not contain main.

**Instantiating Inner Class:**

**Instantiating an Inner Class within the outer class:**

class MyOuter

{

private int x=7;  
 private void makeInner()

{

MyInner in=new MyInner();

In.seeOuter();

}

class MyInner

{

public void seeOuter()

{

System.out.println(“Outer x is: “+x);

}

}

}

**Creating An inner Class Object From Outside The Outer Class Instance Code:**

You have to create the outer class’s instance first.

**Inner Class: Using It’s Own Reference and Outer Class’s reference:**

class myInner

{

public void seeOuter()

{

System.out.println(“Outer class is:”+x);

System.out.println(“Inner class ref is: “+this);

System.out.println(“Outer class reference is:+MyOuter.this);

}

}

**Member Modifiers Applied To Inner Classes:**

* final
* abstract
* public
* private
* Protected
* static (then, it will be static nested class, not an inner class)
* strictfp

**Method Local Inner Class:**

class MyOuter

{

private String x="Outer";

void dostuff()

{

class MyInner

{

public void seeOuter()

{

System.out.println("Outer x is: "+x):

}

//close inner class method

//close inner class definition

}

MyInner mi=new MyInner();

//this line must come after the class

mi.seeOuter();

}

//class outer class method dostuff()

//close outer class

}

**What a Method local Inner Object Can and Cannot do:**

* A method local inner class can be instantiated only within the method where the inner class is defined. In other words, no other code running in any other method-inside or outside the outer class -can never instantiate the method local inner class.
* A inner class object cannot use it’s local variable of the methods the inner class is in. Because, those local variables of the methods are in the stack. Now, inner class reference has an object initialized to it which is allocated from heap. When the program ends, the reference alongside with the method local variables are gone. We can prevent the reference by passing the reference to some other function and there initializing it to some other reference. But, the variables will be lost. **Until those variables are made final.**

**Anonymous Inner Classes:**

**First Example:**

class Popcorn

{

public void pop()

{

System.out.println("popcorn");

}

class Food()

{

Popcorn p=new Popcorn()

{

public void pop()

{

System.out.println("Anonymous popcorn");

}

};

}

}

Now, most important node about this, is the reference variable p does not refer to an instance of Popcorn, but to an instance of anonymous (unnamed subclass of popcorn)

Now, in is simple.

**An complex Example:**

interface Cookable

{

public void cook();

}

class Food

{

Cookable c=new Cookable()

{

public void cook()

{

System.out.println("anonymous cookable implementation");

}

};

}

**Static Nested Class (Not Static Inner Class)**

**We can not declare top-level class with a static modifier, but can declare nested classes as static. Such type of classes are called Nested static classes.**

**What are the differences between static and non-static nested classes?**

Following are major differences between static nested class and non-static nested class. Non-static nested class is also called Inner Class.

1) Nested static class doesn’t need reference of Outer class, but Non-static nested class or Inner class requires Outer class reference.

/\* Java program to demonstrate how to implement static and non-static

classes in a java program. \*/

class OuterClass{

private static String msg = "GeeksForGeeks";

// Static nested class

public static class NestedStaticClass{

// Only static members of Outer class is directly accessible in nested

// static class

public void printMessage() {

// Try making 'message' a non-static variable, there will be

// compiler error

System.out.println("Message from nested static class: " + msg);

}

}

// non-static nested class - also called Inner class

public class InnerClass{

// Both static and non-static members of Outer class are accessible in

// this Inner class

public void display(){

System.out.println("Message from non-static nested class: "+ msg);

}

}

}

class Main

{

// How to create instance of static and non static nested class?

public static void main(String args[]){

// create instance of nested Static class

OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();

// call non static method of nested static class

printer.printMessage();

// In order to create instance of Inner class we need an Outer class

// instance. Let us create Outer class instance for creating

// non-static nested class

OuterClass outer = new OuterClass();

OuterClass.InnerClass inner = outer.new InnerClass();

// calling non-static method of Inner class

inner.display();

// we can also combine above steps in one step to create instance of

// Inner class

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

// similarly we can now call Inner class method

innerObject.display();

}

}

**Now, look at this:**

OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();

printer.printMessage();

1. For instantiating static inner class, we need not a reference of Outerclass which has to be instantiated with OuterClass’s object.

2) Inner class(or non-static nested class) can access both static and non-static members of Outer class. A static class cannot access non-static members of the Outer class. It can access only static members of Outer class.