**Basics of Nested Classes**

The Java programming language allows you to define a class within another class. Such a class is called a nested class and is illustrated here:

Syntex :

public class OuterClass {

...

class NestedClass {

...

}

}

Nested classes are divided into two categories:  **static** and **non-static**.

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

Non-static nested classes are called inner classes.

Syntex :

public class OuterClass {

...

static class StaticNestedClass {

...

}

class InnerClass {

...

}

}

A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.

Static nested classes do not have access to other members of the enclosing class. As a member of the OuterClass, a nested class can be declared private, public, protected, or package private. (Recall that outer classes can only be declared public or package private.)

**Why Use Nested Classes?**

Compelling reasons for using nested classes include the following:

* It is a way of logically grouping classes that are only used in one place: If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.
* It increases encapsulation: Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
* It can lead to more readable and maintainable code: Nesting small classes within top-level classes places the code closer to where it is used.

**Static Nested Classes**

As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

Static nested classes are accessed using the enclosing class name:

OuterClass.StaticNestedClass

For example, to create an object for the static nested class, use this syntax:

OuterClass.StaticNestedClass nestedObject =

new OuterClass.StaticNestedClass();

The complete example :

public class StaticNestedClasses {

public static void main(String[] args) {

final OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

nestedObject.innerClassMethod();

}

}

class OuterClass{

public void outerClassMethod(){

System.out.println( " outerClassMethod ");

}

static class StaticNestedClass{

public void innerClassMethod(){

System.out.println( "innerClassMethod ");

}

}

}

**Inner Classes or Non-Static Classes**

As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

Objects that are instances of an inner class exist within an instance of the outer class. Consider the following classes:

class OuterClass {

...

class InnerClass {

...

}

}

An instance of InnerClass can exist only within an instance of OuterClass and has direct access to the methods and fields of its enclosing instance.

To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:

OuterClass.InnerClass innerObject = outerObject.new InnerClass(); Inner Class Example :

public class InnerClassExampleOracle {

public static void main(String s[]) {

// Fill the array with integer values and print out only

// values of even indices

final DataStructure ds = new DataStructure();

ds.printEven();

}

}

class DataStructure {

// Create an array

private final static int SIZE = 15;

private final int[] arrayOfInts = new int[SIZE];

public DataStructure() {

// fill the array with ascending integer values

for (int i = 0; i < SIZE; i++) {

arrayOfInts[i] = i;

}

}

public void printEven() {

// Print out values of even indices of the array

final DataStructureIterator iterator = this.new EvenIterator();

while (iterator.hasNext()) {

System.out.print(iterator.next() + " ");

}

System.out.println();

}

interface DataStructureIterator extends java.util.Iterator<Integer> { }

// Inner class implements the DataStructureIterator interface,

// which extends the Iterator<Integer> interface

private class EvenIterator implements DataStructureIterator {

// Start stepping through the array from the beginning

private int nextIndex = 0;

public boolean hasNext() {

// Check if the current element is the last in the array

return (nextIndex <= SIZE - 1);

}

public Integer next() {

// Record a value of an even index of the array

final Integer retValue = Integer.valueOf(arrayOfInts[nextIndex]);

// Get the next even element

nextIndex += 2;

return retValue;

}

public void remove() {

// TODO Auto-generated method stub

}

public void forEachRemaining(Consumer<? super Integer> action) {

// TODO Auto-generated method stub

}

}

}

There are two special kinds of inner classes:

* Local Inner Classes
* Anonymous Classes

**Local Inner Classes**

Local classes are classes that are defined in a block, which is a group of zero or more statements between balanced braces. You typically find local classes defined in the body of a method.

**Declaring Local Classes**

You can define a local class inside any block (see Expressions, Statements, and Blocks for more information). For example, you can define a local class in a method body, a for loop, or an if clause.

The following example, LocalClassExample, validates two phone numbers. It defines the local class PhoneNumber in the methodvalidatePhoneNumber:

public class LocalClasseExample {

static String regularExpression = "[^0-9]";

public static void validatePhoneNumber(

String phoneNumber1, String phoneNumber2) {

final int numberLength = 10;

// Valid in JDK 8 and later:

// int numberLength = 10;

class PhoneNumber {

String formattedPhoneNumber = null;

PhoneNumber(String phoneNumber){

// numberLength = 7;

final String currentNumber = phoneNumber.replaceAll(

regularExpression, "");

if (currentNumber.length() == numberLength)

formattedPhoneNumber = currentNumber;

else

formattedPhoneNumber = null;

}

public String getNumber() {

return formattedPhoneNumber;

}

// Valid in JDK 8 and later:

// public void printOriginalNumbers() {

// System.out.println("Original numbers are " + phoneNumber1 +

// " and " + phoneNumber2);

// }

}

final PhoneNumber myNumber1 = new PhoneNumber(phoneNumber1);

final PhoneNumber myNumber2 = new PhoneNumber(phoneNumber2);

// Valid in JDK 8 and later:

// myNumber1.printOriginalNumbers();

if (myNumber1.getNumber() == null)

System.out.println("First number is invalid");

else

System.out.println("First number is " + myNumber1.getNumber());

if (myNumber2.getNumber() == null)

System.out.println("Second number is invalid");

else

System.out.println("Second number is " + myNumber2.getNumber());

}

public static void main(String... args) {

validatePhoneNumber("123-456-7890", "456-7890");

}

}

The example validates a phone number by first removing all characters from the phone number except the digits 0 through 9. After, it checks whether the phone number contains exactly ten digits (the length of a phone number in North America). This example prints the following:

*`First number is 1234567890*

*`Second number is invalid*

**Local Classes Are Similar To Inner Classes**

Local classes are similar to inner classes because they cannot define or declare any static members. Local classes are non-static because they have access to instance members of the enclosing block. Consequently, they cannot contain most kinds of static declarations.

You cannot declare an interface inside a block; interfaces are inherently static. For example, the following code excerpt does not compile because the interface HelloThere is defined inside the body of the method greetInEnglish:

public class LocalClasseExample {

static String regularExpression = "[^0-9]";

public void greetInEnglish() {

interface HelloThere {

public void greet();

}

class EnglishHelloThere implements HelloThere {

public void greet() {

System.out.println("Hello " + name);

}

}

HelloThere myGreeting = new EnglishHelloThere();

myGreeting.greet();

}

}

**Note : This example gives compile time error:** The member interface **HelloThere**can only be defined inside a top-level class or interface or in a static context.