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:

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

Nested classes are divided into two categories:

1. Static
2. Non-static.

Nested classes that are declared static are called *static nested classes*.
Non-static nested classes are called *inner classes*.

```
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*.

Why Use Nested Classes?

Compelling reasons for using nested classes include the following:

1. 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.
2. 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.
3. 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.
Note: A static nested class interacts with the instance members of its outer class (and other classes) just like any other top-level class. In effect, a static nested class is behaviourally a top-level class that has been nested in another top-level class for packaging convenience.

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 following code excerpt shows how static nested classes are defined, how they access the the private members of their enclosing class (only via the use of an object instance) and how to obtain an instance a static nested class:

```
class StaticInner{
static class Inner{
 void inn(){
System.out.println("inner");
   }
}
public static void main(String args[]){
Inner  ob = new Inner();
ob.inn();
   }
}
```

Non-Static Nested Class

Inner 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 {
      ...
   }
```

}

1. 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.
2. To instantiate an inner class, you must first instantiate the outer class. Then, create the inner object within the outer object with this syntax:
3. OuterClass.InnerClass innerObject = outerObject.new InnerClass();
4. There are three kinds of inner classes: Member inner class, local inner classes and anonymous inner classes.

A. Member inner class

Member inner classes are treated as a member of an outer class.

Member Inner class can be accessed only through live instance of outer class.

Outer class can create instance of the inner class in the same way as normal class member.

Member inner class will be treated like member of the outer class so it can have several Modifiers as opposed to Class.

- final
- abstract
- public
- private
- protected

Created As Below:

```
class OuterClass {
      ...
     class InnerClass {
      ...
   }
}
```

Example:

```
class Outer{
    void out(){
                System.out.println("outer");
               Inner  ob = new Inner();
               ob.inn();
              }
   class Inner{
       void inn(){
                   System.out.println("inner");
                    }
                }
}
class MemberInner{
public static void main(String args[]){
    Outer ob= new Outer();
   ob.out();
   }
}
```

B. Local inner class

1. 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.
2. You can define a local class inside any block. For example, you can define a local class in a method body, a for loop, or an ifclause.
3. A local class has access to the members of its enclosing class.
4. A local class has access to local variables. However, a local class can only access local variables that are declared final. Local classes in static methods, such as the class , which is defined in the static method can only refer to static members of the enclosing class

Example:

```
class Outer{

   void out(){

    class Inner

     {

       void inn(){

         System.out.println("inner");

         }

       }

    System.out.println("outer");

    Inner  ob = new Inner();

    ob.inn();

  }

}

class LocalInner
```

```
{
public static void main(String args[])
 {
  Outer ob= new Outer();
  ob.out();
 }
}
```

C. Anonymous inner class

1. Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name. Use them if you need to use a local class only once.
2. While local classes are class declarations, anonymous classes are expressions, which means that you define the class in another expression
3. Anonymous classes have the same access to local variables of the enclosing scope as local classes:

    a. An anonymous class has access to the members of its enclosing class.
    b. An anonymous class cannot access local variables in its enclosing scope that are not declared as `final` or effectively final.

4. Anonymous classes also have the same restrictions as local classes with respect to their members:

    a. You cannot declare static initializers or member interfaces in an anonymous class.
    b. An anonymous class can have static members provided that they are constant variables.

Example:

```
abstract class Outeranon{
abstract void display();
}
class AnonymousInner{
 public static void main(String args[]){
Outeranon ob =new Outeranon(){
void display()
  {
    System.out.println("inner");
  }
};
ob.display();
   }
}
```

REFRENCES

[1]The Complete Refrence JAVA by Herbert Schildt, TATA McGRAW-HILL publication.

[2] http://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html