INNER CLASSES IN JAVA

Inner class is a class within another class, method or block. An inner class can be one of the following four types: Anonymous, Local, Member and Nested top-level.

An **anonymous inner class** is an inner class without a name and is created as part of a statement.
A inner class declared within a method is called **method-local inner class**.
A **member inner class** is a **regular inner class** declared within a class, but outside all methods.
Static inner classes or nested top-level inner classes are actually nested classes that are inner classes marked with the static modifier.

A regular or member inner class

A "regular" inner class inside the class, but outside all methods or blocks, is a normal member of the enclosing (outer) class, and can be marked with any access modifier and/or, an abstract or a final modifier. Note that you can't use both abstract and final together for an inner class or even a top-level class.

A regular non-static member inner class instance has access to all of the outer class's members, including those marked private.

To instantiate an inner class from outside the enclosing class (outer class), you must have a reference to an instance of the outer class.

MyOuter mo = new MyOuter(); //instance of outer class

MyOuter.MyInner inner = mo.new MyInner();

To access an inner class, you have to use the outer class name with dot operator before it as: OuterClass.InnerClass.

From code within the enclosing class (outer class), you can instantiate the inner class using only the name of the inner class, just like a regular class.

From code within the inner class, the keyword 'this' holds a reference to the inner class instance.

To reference the enclosing class (outer class) 'this' from an inner class, precede the keyword 'this' with the outer class name as:

MyOuter.this;

Method-Local Inner Classes

A method-local inner class is defined within a method of the enclosing class.

You must instantiate this inner class within the same method, but after the class definition code.

A method-local inner class can only use final variables within the method.

The only modifiers you can apply to a method-local inner class are abstract or final, but never both at the same time.

We can declare a method-local inner class with the same name as a regular inner class within the same class and when accessed from within the local block (the method), local inner class will take precedence, just like a local variable take precedence over an instance variable.

Anonymous Inner Classes

Anonymous inner classes have no name, and they either subclass a class or implement an interface.

An anonymous inner class is always created as part of a statement and hence might have to end with a semicolon when applicable.

An **argument-defined inner class** is declared, defined, and automatically instantiated as part of a method invocation and should not end with a semicolon.
We can define an anonymous inner class for an interface as **new InterfaceName(){ //implementation code }**and use it in places where an object of type InterfaceName is required.

```
interface Foo { int bar(); }

public class Sprite {

public int fubar( Foo foo ) { return foo.bar(); }

public void testFoo() {

fubar( new Foo() { public int bar() { return 1; }});
}

}
```

We are not instantiating an Interface here. We are actually creating an object of a new, anonymous inner class that implements interface Foo using a special syntax.

Because of polymorphism, the only methods you can call on an anonymous inner class reference are those defined in the reference variable class (or interface).

Unlike non-anonymous classes (inner or otherwise), an anonymous inner class cannot both extend a class and implement an interface, nor can it implement more than one interface.

Static Nested Classes

Static nested classes are inner classes marked with the static modifier.

A static nested class is not like a regular inner class, but it's a top-level nested class.

Because the nested class is static, it does not share any special relationship with an instance of the outer class. In fact, you don't need an instance of the outer class to instantiate a static nested class.

Instantiating a static nested class requires using both the outer and nested class names as follows:

OuterClass.StaticInner n = new OuterClass.StaticInner();

Source : http://javajee.com/inner-classes-in-java