Java Notes 2012

An Introduction into the concept of nested classes in Java
[Introduction to nested classes in Java | JavaWorld's Daily Brew]
Overview
This article is going to shed some lights on the concepts of the implementation of the nested classes in Java in a simple and easy to comprehend manner. The various types of nested classes are explained with some examples that explain their usage and advantage over normal top-level-classes.
Concepts
One of the nice and highly advantageous features of Java is the ability to define nested classes, i.e. classes within classes. In java classes are either top-level-classes (classed defined directly within a package) or nested classes (classes defined within top-level-classes). One might wonder in regard to why all this complication is even helpful, and whether or not such classes bring any advantage to the java programmers at all.

Well, during the time I've spent exploring the what's ad why's of Java's nested classes, I've realized two important, but distinct, advantages to that, and these are:

1. Having the ability to define nested classes will help grouping the related functionality together, imagine the case in which a certain class is being used only once from within another class, then there is no obvious need to add a clutter to the enclosing package by defining a top-level-class that will merely be used by only one another class. Thus, having that functionality contained within a a nested class increase readability and reduce clutter.
2. Another very important advantage of having nested classes is the fact that nested classes will be able to access all the class members of their enclosing top-level-class (including the private member methods and variables), this is a more efficient way than the traditional way of accessing the non-private members using an object instance and a noticeable advantage of this feature can be seen when developing event-based applications where action and event listeners can be defined as nested classes rather than having them as separate individual top-level-classes
3. A difference or may be an advantage nested classes posses over subclasses (as one might think, why not simply define subclasses instead of nested classes) is that subclasses can't access private members (unless they are declared as protected) while nested classes can.

Nested classes types and basic examples
Nested classes in java can be classified into the following types, each exhibit a slightly different behaviors than each other, which might limit their use to a specific particular case:

- non-static nested classes: Non static nested classes are defined within the top-level-classes and they are bound by the scope of their enclosing class, non-static nested classes can have direct access into all of the members (event the private ones) of their enclosing class, on the other hand, the enclosing class doesn't have any access to the members of the enclosed nested class. It is also worth to mention that nested classes (in general) can be defined from within any block inside a top-level-class, that's could be a method block or even a control statement target block (such as if, for or while). There are two flavors of non-static nested classes, and these are:
    - Inner Classes: These are the most widely used type of nested classes, the following code excerpt shows how to define a an inner class within a normal top-level-class, how easily the private members of the enclosing class can be accessed by the nested class and how to obtain

an instance of the inner class from within the enclosing class:

```
public class OuterClass
{
        private int private_member_variable = 100;
        public class InnerClass
        {
        public void printPrivateVariable()
        {
        System.out.println(private_member_variable);
        }
        }
        public void callInnerClassMethod()
        {
        InnerClass innerClass = new InnerClass();
        innerClass.printPrivateVariable();
        }
        public static void main(String args[])
        {
        OuterClass outerClass = new OuterClass();
        outerClass.callInnerClassMethod();
        }
}
```

- It is very important to state that the instance of the Inner class doesn't exist in isolation, rather it is bounded by and associated with an instance of their enclosing top-level-class, and this impose a slight restriction on the inner nested classes which is that they can't have any static members that normal top-level and static nested classes can. the following code excerpt depicts that concept and how an instance of the Inner class can be obtained from code outside their enclosing class:

```
public class TestClass
{
        public static void main(String args[])
        {
        OuterClass outerClass = new OuterClass();
        OuterClass.InnerClass innerClass = outerClass.new InnerClass();
        innerClass.printPrivateVariable();
        }
}
```

- Anonymous Classes: Java supports the definition of un-named inner classes, these classes differ from the inner classes in only one thing, which is that they are un-named, hence came the name "anonymous". Anonymous classes can have direct access to all of the class members of the top-level-class in which they are defined.
- The following code excerpt shows how to define anonymous classes from within top-level-classes:

    - public class Test
    - {
    - public static void main(String args[])
    - {
    - Runnable runnable = new Runnable()
    - {
    - public void run()
    - {
    -
    - }
    - };
    - }
    - }
    -
    - Anonymous inner classes come in handy when an object instance are required on the fly that extends a certain class or implement a certain interface without the need to write a complete top-level-class for that. The situations where anonymous inner classes are beneficial can be envisaged when the need arise for a Runnable class to be passed to a Thread (as the above code excerpt shows) or when an event listener implementation need to be passed to register for certain event occurrence in event-driven development environments. The syntax for defining anonymous inner classes can be broken down into the following:
    -
    - new Interface to implement | abstract/concrete class to extend () {body of the anonymous class to be defined};
    -
- static nested classes: Static nested classes exist in isolation from within the top-level-class that they are defined within. On the contrary to the non-static nested classes, static nested classes can't directly access the members of the class that they are defined in (simply because they are static members, and in static class members can only access other static members directly), but they can do so using an object instance of the enclosing top-level-class. 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:
-
- public class OuterClass
- {
- private int private_member_variable = 100;
-
- public static class StaticInnerClass
- {
- public void printPrivateVariables()
- {
- OuterClass outerClass = new OuterClass();
- System.out.println(outerClass.private_member_variable);
- }
- }
-
- public static void main(String args[])
- {
- StaticInnerClass staticInnerClass = new StaticInnerClass();

- staticInnerClass.printPrivateVariables();
- }
- }
-
- To obtain an instance of a static nested class from within code defined outside the enclosing class, one needs to follow the normal instance creation mechanism in java as the static class exist in isolation from their enclosing class and they are not associated with any instance of their enclosing class:
-
- public class Test
- {
- public static void main(String args[])
- {
- OuterClass.StaticInnerClass staticInnerClass = new OuterClass.StaticInnerClass();
- staticInnerClass.printPrivateVariables();
- }
- }
-
