5.7: Nested and inner classes

IT1406 - Introduction to Programming

Level I - Semester 1

5.7. Nested and inner classes

- It is possible to define a class within another class; such classes are known as *nested classes*.
- The scope of a nested class is bounded by the scope of its enclosing class.
- There are two types of nested classes:
  - *static*
  - *non-static*
- A static nested class is one that has the static modifier applied. Because it is static, it must access the non-static members of its enclosing class through an object. That is, it cannot refer to non-static members of its enclosing class directly. Because of this restriction, static nested classes are seldom used.

5.7. Nested and inner classes

- The most important type of nested class is the *inner* class. An inner class is a non-static nested class. It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.
- The following program illustrates how to define and use an inner class. The class named **Outer** has one instance variable named **outer_x**, one instance method named **test( )**, and defines one inner class called **Inner**.

5.7. Nested and inner classes

```
// Demonstrate an inner class.
class Outer {
        int outer_x = 100;
        void test() {
                Inner inner = new Inner();
                inner.display();
        }
        // this is an inner class
        class Inner {
                void display() {
                        System.out.println("display: outer_x = " + outer_x);
                }
        }
}
class InnerClassDemo {
        public static void main(String args[]) {
                Outer outer = new Outer();
                outer.test();
        }
}
```

5.7. Nested and inner classes

- Output from this application is shown here:

  display: outer_x = 100

- In the program, an inner class named **Inner** is defined within the scope of class **Outer**. Therefore, any code in class **Inner** can directly access the variable **outer_x**. An instance method named **display( )** is defined inside **Inner**. This method displays **outer_x** on the standard output stream. The **main( )** method of **InnerClassDemo** creates an instance of class **Outer** and invokes its **test( )** method. That method creates an instance of class **Inner** and the **display( )** method is called.
- It is important to realize that an instance of **Inner** can be created only in the context of class **Outer**.
- Members of the inner class are known only within the scope of the inner class and may not be used by the outer class.
- It is possible to define inner classes within any block scope. For example, you can define a nested class within the block defined by a method or even within the body of a **for** loop, as this next program shows:

5.7. Nested and inner classes

```
// Define an inner class within a for loop.
class Outer {
        int outer_x = 100;
        void test() {
                for(int i=0; i<10; i++) {
                        class Inner {
                                void display() {
                                        System.out.println("display: outer_x = " + outer_x);
                                }
                        }
                Inner inner = new Inner();
                inner.display();
                }
        }
}
class InnerClassDemo {
        public static void main(String args[]) {
                Outer outer = new Outer();
                outer.test();
        }
}
```

5.7. Nested and inner classes

- The output from this version of the program is shown here:

```
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
display: outer_x = 100
```