1. Objects and Classes II :: Nested Classes

A **nested class** is a class declared within another class:

```
public class Outer {
  // Outer instance variable declarations ...
  // Outer instance method declarations ...
  private class Nested {
    // Nested instance variable declarations ...
    // Nested instance method declarations ...
  }
}
```

Why use nested classes? The primary use is when a reasonably small class (the nested class) is useful to only one other class (the outer class). This increases encapsulation and can lead to more readable and maintainable code.

1. Objects and Classes II :: Nested Classes :: Inner Classes

Nested classes are divided into two categories: **static** nested classes and **nonstatic** nested classes (referred to as **inner classes**). Static nested classes are rare so we will not discuss them.

An **inner class** is declared within an outer class and is associated with an instance (object) of the outer class, i.e., an inner class object exists as an instance variable encapsulated within an outer class object:

```
public class Outer {
  private int mX;
  private Inner mInner;                   // An instance of the inner class Inner
  public Outer(int pX, int pY) {
    mX = pX;
    mInner = new Inner(pY);               // Instantiate an Inner object
  }
  private class Inner {
    int mY;                               // An inner class may declare instance
    public Inner(int pY) { mY = pY; }     // variables and instance methods
  }
}
public void someMethod() {
  Outer outer = new Outer(10, 20);
  ...
}
```

1. Objects and Classes II :: Nested Classes :: Inner Classes (continued)

Within the methods of an inner class, all of the outer class instance variables and methods are accessible, regardless of the accessibility modifiers associated with them. An inner class may not declare static (class) variables or methods. An inner class may also be public:

```
public class Outer {
  ...
  public class PublicInnerClass {
    // Instance variable and method declarations ...
  }
}
```

PublicInnerClass can be accessed from outside *Outer* but a *PublicInnerClass* object cannot stand on its own:

```
Outer.PublicInnerClass innerObj = new Outer.PublicInnerClass(); // Syntax error!
```

but rather, can only be created within the an *Outer* object:

```
// Create an Outer object
Outer outerObj = new Outer();
// Create a PublicInnerClass object inside of outerObj
Outer.PublicInnerClass innerObj = outerObj.new PublicInnerClass();
```