Nested Classes

It is possible to define a class as a member of another class. Such a class is called **nested class**:

```
class EnclosingClass {
      . . .
    class NestedClass {
          . . .
    }
}
```

A nested class has special privilege: It has unlimited access to its enclosing class's members, even if they are declared private.

A class should be defined within another class when the nested class makes sense only in the context of its enclosing class or when it relies on the enclosing class for its function.

Nested Classes (cont.)

Like other members, a nested class can be declared static.

A static nested class is called just **static nested class.**

A non-static nested class is called an **inner class.**

```
class EnclosingClass {
      . . .
    static class StaticNestedClass {
          . . .
    }

    class InnerClass {
          . . .
    }
}
```

Nested Classes (cont.)

A **static nested class** cannot refer directly to instance variables or methods defined in its enclosing class.

An **inner class** is associated with an instance of its enclosing class and has direct access to that object's instance variables and methods. It cannot define any (non-final) static members itself.

Like other classes, nested classes can be declared **abstract** or **final**.

Also, the access specifiers – **private**, **protected** and **public** – may be used to restrict access to nested classes.

A nested class can be also declared in **any** block of code.

A nested class declared within a method or other smaller block of code has access to any final local variables in scope.

Inner Classes – Example

```
public class Container1 {
   private Object[] items;
     . . .
   public Iterator iterator() {
      return new ContainerIterator();
   }

   class ContainerIterator implements Iterator {
      int index = 0;
      public boolean hasNext() {
         return index < items.length;
      }
      public Object next() {
         if (!hasNext())
            throw new NoSuchElementException();
         return items[index++];
      }
        . . .
   }
}
```

Anonymous Inner Classes

An inner class can be declared without naming it. However, anonymous classes can make code **difficult to read**.

```
public class Container2 {
   private Object[] items;
     . . .
   public Iterator iterator() {
      return new Iterator() {
         int index = 0;
         public boolean hasNext() {
            return index < items.length;
         }
         public Object next() {
            if (!hasNext())
               throw new NoSuchElementException();
            return items[index++];
         }
           . . .
      };
   }
}
```

Anonymous Inner Classes (cont.)

An anonymous class is never **abstract**.

An anonymous class is always an **inner class**, it is never **static**.

An anonymous class is always implicitly **final**.

An anonymous class cannot have an explicitly declared constructor. Instead, the compiler provides an **anonymous constructor**.