Inner Classes

- What are Inner Classes?
- Categories of Inner Classes
- Using Inner Classes

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What Are Inner Classes?

- Inner classes are classes that are defined within the block of another class

```
public class EnclosingClass
{
   public class MyInnerClass
   {
      . . .
   }
   . . .
}
```

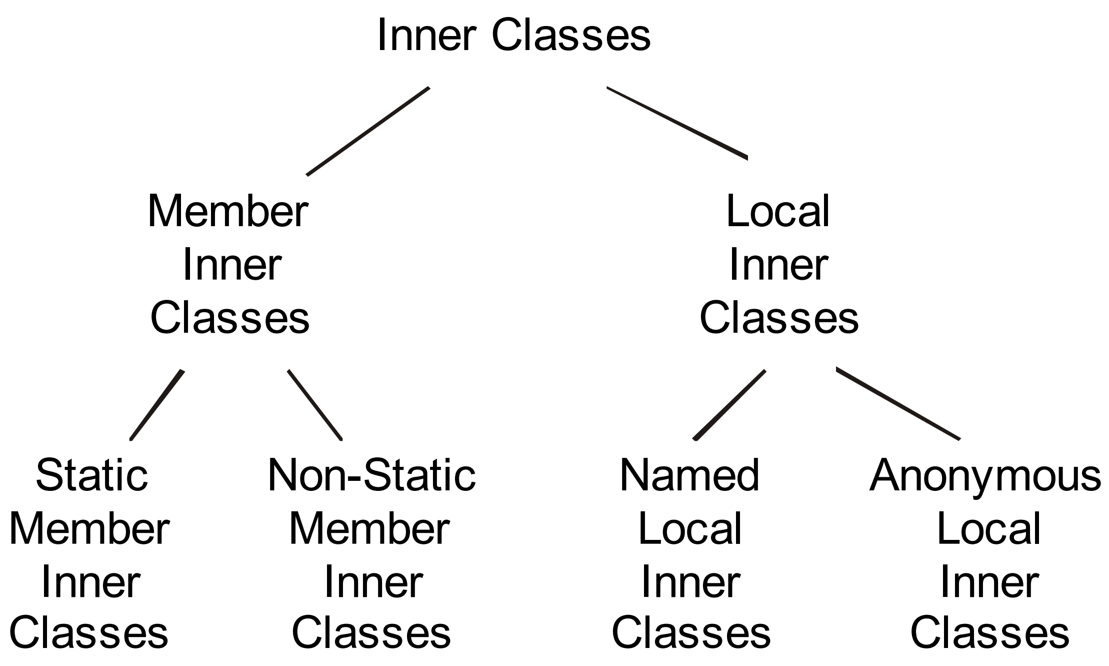
Inner classes let you scope a class definition to within another class.

Inner classes were introduced to Java in Version 1.1.

When you compile a class with inner classes, the compiler generates separate class files for the enclosing and inner classes. The naming convention for the inner-class file names is EnclosingClass$MyInnerClass.class (there is a special case for file names of anonymous inner classes).

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inner Class Categories


Inner Classes
Member Inner Classes
Local Inner Classes
Static Member Inner Classes
Non-Static Member Inner Classes
Named Local Inner Classes
Anonymous Local Inner Classes

These category names are widely used and descriptive, but are not actually defined in the Java documentation.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use Inner Classes?

- If a given class is used only by a single class, you can simplify the design by using an inner class
- Inner classes let you explicitly show that a class depends on another
- Anonymous local inner classes are very handy to hook up listeners to JavaBean events
- Downside: Inner class syntax is a bit obscure

Used properly, inner classes can simplify designs and make code more readable and maintainable. The syntax can be a bit intimidating until you get used to it, however.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Static Member Inner Classes

- Static member inner classes are defined as static within the enclosing class

```
public class MyEnclosingClass
{
. . .
    static public class InnerClass3
    {
        private int y = 14;

        public void myMethod()
        {
            System.out.println(y);
        }
    }
. . .
}
```

Here we show defining a static member inner class that itself defines a private field and a public method. Within the inner class, you are free to use any of the class definition techniques available to a non-inner class.

Since the inner class is defined as public, it is accessible from outside of the enclosing class. If we had defined it as private, then it would only be visible within the enclosing class.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Enclosing Class Members

- Static member inner classes can access any static field or method defined in the enclosing class

```
public class MyEnclosingClass
{
    private int x = 12;
    static private int y = 20;
    static private int z = 21;

    static public class InnerClass3
    {
        private int y = 14;

        public void myMethod()
        {
          System.out.println(y);
          System.out.println(z);
          System.out.println(MyEnclosingClass.y);
        }
    }
. . .
}
```

The inner class can access private members of the enclosing class. That makes sense if you consider that the inner class is itself considered a member of the enclosing class.

Note the odd syntax that's required if the inner class defines a field with the same name as a field in the enclosing class.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Instantiating Static Member Inner Class Objects

- From within a static method in the enclosing class, you can use normal **new** operator syntax
- From outside the enclosing class, you use **new** and specify both the enclosing and static member inner class name

Inside enclosing class

```
InnerClass3 m3 = new InnerClass3();
```

Outside of the enclosing class

```
MyEnclosingClass.InnerClass3 m3 =
    new MyEnclosingClass.InnerClass3();
```

The syntax to instantiate a static member inner class object from outside of the class is a bit weird, but is consistent with the syntax for accessing any kind of static member.

Note that in either case, since the inner class is static, you don't have to first create an instance of the enclosing class. We will see that with non-static member classes, you will need to first create an enclosing class object.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Non-Static Member Inner Classes

- Non-static member inner classes are defined within the boundary of the enclosing class

```
public class MyEnclosingClass
{
. . .
    public class InnerClass2
    {
        private int x = 14;

        public void myMethod()
        {
            System.out.println(x);
        }
    }
. . .
}
```

Non-static member inner classes are syntactically similar to static -- just omit the static keyword. You instantiate objects and access them a bit differently, however.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Enclosing Class Members

- Non-static member inner classes can access any of the enclosing class fields or methods

```
public class MyEnclosingClass
{
    private int x = 12;

    public class InnerClass3
    {
        private int x = 14;
        private static int y = 17;

        public void myMethod()
        {
          System.out.println(x);
          System.out.println(y);
          System.out.println(MyEnclosingClass.this.x);
        }
    }
. . .
}
```

Non-static member inner classes can access any of the enclosing class's members, be they private or not, static or not.

Note the odd syntax required if an inner class defines a field with the same name as field from the enclosing class. The implication is that within an inner class, there are actually two "this" references -- the "this" for the inner class itself and the "this" for the enclosing class.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Instantiating Non-Static Member Inner Class Objects

- Before you can instantiate a non-static member inner class object, you must first instantiate an enclosing class object
- From within a non-static method in the enclosing class, you can use normal **new** operator syntax
- From outside, you use **new** along with the reference to the enclosing class object

Inside enclosing class non-static method

```
InnerClass2 m2 = new InnerClass2();
```

Outside of an enclosing class non-static method

```
MyEnclosingClass m = new MyEnclosingClass();
MyEnclosingClass.InnerClass2 m2 =
    m.new InnerClass2();
```

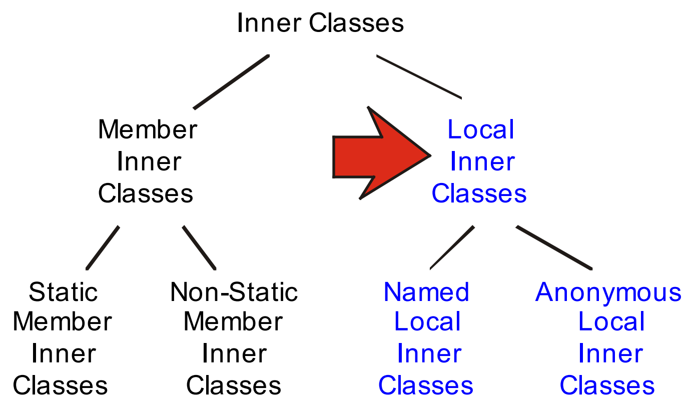
Non-static member inner classes are considered to part of an enclosing class INSTANCE, not of the enclosing class itself (static inner classes ARE considered part of the enclosing class). Therefore, the enclosing class instance must first exist before you can create an instance of the inner class.

Again, the syntax is different depending on where you are trying to instantiate the inner class object. From within an enclosing-class, non-static method, there's an implicit "this" for the enclosing class, so you can use the normal "new" syntax. From anywhere else, you need to use the rather odd "new" syntax that uses the dot operator following the reference to the enclosing class instance.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Local Inner Classes

- Local inner classes are defined within the boundary of a method
- The local class is only visible within the method that defines it
- Local class methods can access enclosing class fields *and* method local variables defined as **final**
- There are two categories of local inner classes: **named** and **anonymous**


Inner Classes
Member Inner Classes
Local Inner Classes
Static Member Inner Classes
Non-Static Member Inner Classes
Named Local Inner Classes
Anonymous Local Inner Classes

The key difference between local inner classes and the two categories we've already covered is that local inner classes reside within a method, rather than being a member of the enclosing class itself.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Named Local Inner Classes

- Named local inner classes are defined within the boundary of a method and are only visible within that method

```java
public class MyEnclosingClass
{
    public void anotherMethod()
    {
        class Inner4
        {
            private int x = 17;

            public void myMethod()
            {
                System.out.println(x);
            }
        }
    }
. . .
}
```

Named local inner classes reside within a method and are considered part of the method. They are useful if you need a one-off class just for use by a single method.

Note that you cannot use "public" or "private" modifiers on the inner class definition, since such modifiers are only allowed on members defined at class scope.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Enclosing Class Members

- Named local inner classes can access any enclosing class field or method *and* method local variables defined as **final**

```
public class MyEnclosingClass
{
. . .
    public void anotherMethod()
    {
        int z = 44;
        final int q = 32;
        class Inner4
        {
            private int x = 17;
            public void myMethod()
            {
              System.out.println(x);
              System.out.println(q);
              System.out.println(MyEnclosingClass.this.x);
            }
        }
. . .
}
```

Named local inner classes have basically the same access to fields as do non-static member inner classes, but there's a twist: local inner classes can also access local variables defined in the method in which they reside. Caveat: The local variables must use the final modifier, which means they are constant, and greatly reduces the usefulness of this technique.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Instantiating Named Local Inner Class Objects

- Since named local inner classes are only visible within the method that defined them, you can use normal **new** operator syntax

```
public class MyEnclosingClass
{
. . .
    public void anotherMethod()
    {
        class Inner4
        {
            public void myMethod()
            {
               . . .
            }
        }

        Inner4 m5 = new Inner4();
        m5.myMethod();
    }
}
```

Since local inner classes are only visible from within the method that encloses them, you instantiate and use objects typed to the inner class within the enclosing method.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Intro to Anonymous Local Inner Classes

- Anonymous local inner classes are defined within the boundary of an enclosing class method
- The definition of an anonymous inner class is a Java expression, not a complete statement, so it can reside within an expression (e.g. a call to a method)
- Anonymous inner classes can reference a superclass or a super-interface -- if you specify a super-interface, Object is the implicit superclass

Anonymous inner classes might seem a bit weird at first -- how can you define a class without naming it? But they are quite useful, especially in GUI applications. You do have to get used to the syntax, however.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing and Instantiating an Anonymous Local Inner Class

- An anonymous class can reference a superclass -- Java implicitly defines a new class with the specified fields and methods

```
public class MyEnclosingClass
{
    public void anotherMethod2()
    {
        System.out.println (new Object()
        {
            private int x = 17;
            public String toString()
            {
                return "Inner x: " + x +
            " Enclosing x: " + MyEnclosingClass.this.x;
            }
        } );
    }
. . .
}
```

For anonymous inner classes, you define the class and instantiate an object all in one step.

Note that we defined the inner class within a call to System.out.println -- it's a common scenario to define anonymous classes within method calls, especially registration calls for JavaBean events.

In this case, we subclassed Object and override toString() -- remember that System.out.println() automatically calls toString() on any object references passed.

Anonymous inner classes have the same access to enclosing class members and final local variables as do named local inner classes.

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Anonymous Local Inner Class and Interfaces

- Instead of referencing a superclass, you can specify a super-interface -- Java implicitly defines a new class that subclasses Object and implements the interface
- You will need to include all interface methods in the anonymous inner class!

```
public class MyEnclosingClass
{
    public void anotherMethod3()
    {
        System.out.println (new java.io.Serializable()
        {
            public String toString()
            {
                return "Hello";
            }
        } );
    }
. . .
}
```

In this case, we defined an anonymous inner class that subclasses Object and implements the java.io.Serializable interface (which is a tagging interface with no methods).

It does look odd to use the "new" operator on an interface!

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Review Questions

- What are the four categories of inner classes?
- How is a static member inner class different from the other categories?

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the four categories of inner classes
- How to define inner classes and instantiate inner class-typed objects

Copyright © Descriptor Systems, 2001-2010. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum