Nested Classes

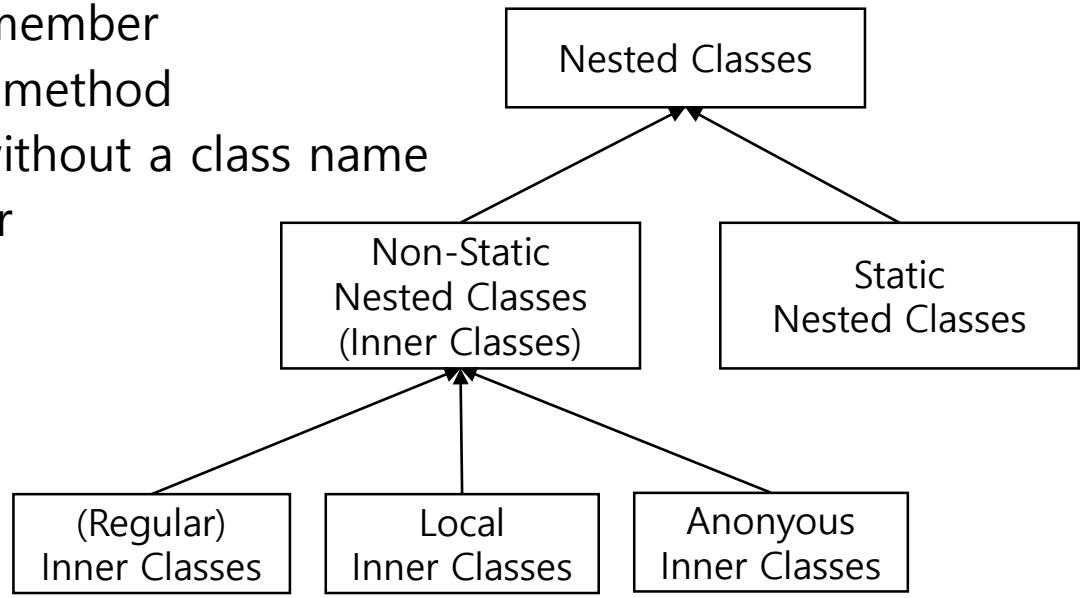
Chapter 6, Core Java Volume I

Contents

- Nested Classes
- Inner Classes
- Special Rules for Inner Classes
- Local Inner Classes
- Anonymous Inner Classes
- Static Nested Classes

Nested Classes

- An nested class is a class that is defined inside another class.
- Four Kinds of Nested Classes
  - (Regular) Inner class – declared as a non-static member
  - Local inner class – declared inside an non-static method
  - Anonymous inner class – like a local class, but without a class name
  - Static nested class – declared as a static member



- Advantages
  - (Organization) Nested classes can be hidden from other classes in the same package.
  - (Convenience) It can lead to more readable and maintainable code.
  - (Access) Inner class methods can access the data from the scope in which they are defined.

Inner Classes

- Example

```
public class TalkingClock  // outer class
{
   private int interval;  // interval between two announcements
   private boolean beep; // flag to turn beeps on or off
   public TalkingClock(int interval, boolean beep) { . . . } // constructor of outer class
   public void start() { . . . }   //
   public class TimePrinter implements ActionListener {  //  inner class inside outer class
      public void actionPerformed(ActionEvent event)
     {
        System.out.println("At the tone, the time is " + new Date());
        if (beep) Toolkit.getDefaultToolkit().beep();
     }
   }
} // end of class
```

Inner Classes

- Creating Objects of Inner Classes
  - A method of the outer class creates and uses instance objects of the inner class.
  - You can construct an inner class object providing an outer class object explicitly:

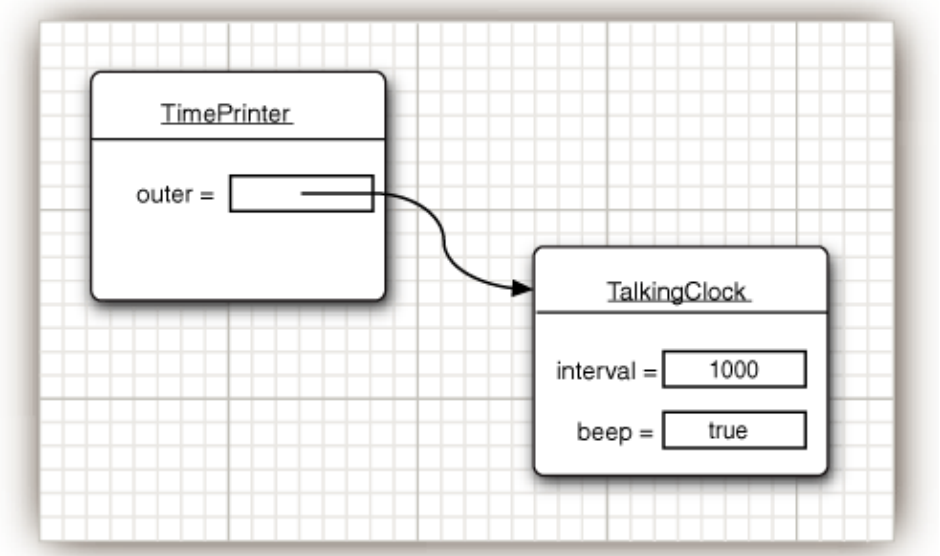
```
public class TalkingClock
{
   ...
   public viod start()
  {
      var listener = new TimePrinter();
      var timer = new Timer(interval, listener);
      timer.start();
   }
   ..
}
```

```
TalkingClock jabberer = . . .;
ActionListener listener = jabberer.new TimePrinter();
```

```
this.new TimePrinter();
```

Inner Classes

- Every inner class object is always associated with an outer class object.
- An inner class object can directly access to the methods and fields of the associated outer object.
  - `TimePrinter` object can access the `beep` field of the `TalkingClock` object that created it.
- The inner class object has a reference to the outer class object that created it.



Listing 6.7: InnerClassTest

```
package innerClass;  // this program demonstrates the use of inner class
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.Timer;
public class InnerClassTest{
   public static void main(String[] args) {
      TalkingClock clock = new TalkingClock(1000, true);
      clock.start();
      JOptionPane.showMessageDialog(null, "Quit program?");
      System.exit(0);
   } // end of main()
} // end of innerclassTest
class TalkingClock {              // outer class
   private int interval;          // interval between messages in milliseconds
   private boolean beep;          // it is true if the clock should beeps
   public TalkingClock(int interval, boolean beep) {
      this.interval = interval;
      this.beep = beep;
   } // end of constructor ;  Continue next page
```

Listing 6.7: InnerClassTest

```
/* Starts the clock.*/
  public void start()
  {
    ActionListener listener = new TimePrinter();   // this.new TimePrinter()
    Timer t = new Timer(interval, listener);
    t.start();
  }

  public class TimePrinter implements ActionListener  // inner class
  {
    public void actionPerformed(ActionEvent event)
    {
      System.out.println("At the tone, the time is " + new Date());
      if (beep) Toolkit.getDefaultToolkit().beep();
    }
  } // end of TimerPrinter(inner class)
}//end of TalkingClock( outer class)
```

Special Rules for Inner Classes

- If a declaration of an inner class has the same name as another declaration in the outer class, then the declaration of the inner class *shadows* the declaration of the outer class.
- To refer to the name of the outer class, use *OuterClass*.this:

```
if (TalkingClock.this.beep) . . .
```

- To use the name of a (non-private) inner class outside the outer class, use *OuterClass.InnerClass*.

```
TalkingClock jabberer = . . .;
TalkingClock.TimePrinter listener = jabberer.new TimePrinter();
```

- Inner classes cannot have static fields and methods.
- In a static method of outer classes, the object of inner classes can not be instantiated without providing an outer class object.

Local Inner Classes

- Local inner class = class defined inside a method:
  - Example: we only needed the TimePrinter class in a single method.

```
public void start()  // this method belongs to  outer class
{
   class TimePrinter implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         System.out.println("At the tone, the time is " + new Date());
         if (beep) Toolkit.getDefaultToolkit().beep();
      }
   }
   ActionListener listener = new TimePrinter();
   Timer t = new Timer(interval, listener);
   t.start();
}
```

- Local classes do not have access modifiers such as public, private, or protected. (*why?*)
- They are completely hidden from the outside world, even other methods in the outer classes

Local Inner Classes

- Local inner class can access fields of its outer class and local variables of its enclosing method

```
public void start(int interval, final boolean beep)  // 1
{  // interval and beep  moved from TalkingClock() constructor  to start() method
      class TimePrinter implements ActionListener {
                public void actionPerformed(ActionEvent event) {  // 6
                  Date now = new Date();
                   System.out.println("At the tone, the time is " + now);
                   if (beep) Toolkit.getDefaultToolkit().beep();
            }
     }
   ActionListener listener = new TimePrinter();  // 2
   Timer t = new Timer(interval, listener);  // 3
    t.start();  // 4
} // 5
```

Local Inner Classes

- A captured variable is one that has been copied so it can be used in a nested class.
  - The reason it has to be copied is that the object may out live the current context.
  - Example: After the **start** method creates the **TimePrinter** object and then exits, the **ActionPerformed** method in the **TimePrinter** object executes **if(beef)… Timer t = new Timer(interval, listener)** … **i.e. access the local variables of the enclosing method.**
  - The **beep** and **interval** parameters are captured variables in the **TimePrinter** object.
- The captured variables must be *effectively final*. (final before Java 8)

Anonymous Inner Classes

- If a local class is only *instantiated once*, it can be anonymous:

```
public void start(int interval, boolean beep)
{
   ActionListener listener = new ActionListener()   // here, do not insert semicolon.
   {
      public void actionPerformed(ActionEvent event)
      {
         System.out.println("At the tone, the time is " + new Date());
         if (beep) Toolkit.getDefaultToolkit().beep();
      }
   };  // do not forget semi-colon.
   Timer t = new Timer(interval, listener);
   t.start();
}
```

use a lambda expression if it has only one method

```
Timer t = new Timer(interval, event-> { .... } );
```

Anonymous Inner Classes

- General syntax:

  new SuperType(construction parameters)
  {
  inner class methods and fields
  }

- For Class Types:
  - the anonymous inner class extends that class
  - construction parameters are given to superclass constructor
- Examples:

```
Person p1 = new Person(" Kim");          // a Person object is created.
Person P2 = new Person("park") {……};    // an object of an inner class extending Person
```

- For Interface Types:
  - the anonymous inner class implements that interfere.
  - anonymous inner class has no construction parameters.

Listing 6.8. AnonymousInnerClassTest

```
package anonymousInnerClass;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.Timer;
public class AnonymousInnerClassTest {  // this program demonstrates anonymous Inner class
   public static void main(String[] args) {
      TalkingClock clock = new TalkingClock();
      clock.start(1000, true);
      JOptionPane.showMessageDialog(null, "Quit program?"); // keep running until user selects "Ok"
      System.exit(0);
   } // end of main
} // end of Classs
class TalkingClock {
   public void start(int interval, boolean beep) {
      ActionListener listener = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
               System.out.println("At the tone, the time is " + new Date());
               if (beep) Toolkit.getDefaultToolkit().beep();
            }
        }; // do not forget semi-colon
      Timer t = new Timer(interval, listener);
      t.start();
   } // end of start()
}//end of TalkingClock
```

Static Nested Classes

- Static nested class = a nested class without reference to a creating object.
  - cannot refer directly to instance variables or methods defined in its enclosing class.
- Useful for a private or scoped class that doesn't need to know the creating object.
- Example:

```
class ArrayAlg
{
  public static class Pair
  {
    public double first;
    public double second;
  }
  . . .
  public static Pair minmax(double[] values)
  {
    . . .
    return new Pair(min, max); // no outer class object
  }
}
```

- Called as:

```
ArrayAlg.Pair p = ArrayAlg.minmax(data);
```

Listing 6.9. StaticInnerClassTest

```
package staticInnerClass;
public class StaticInnerClassTest{  // this program demonstrates the use of static inner class
   public static void main(String[] args)
   {
      double[] d = new double[20];
      for (int i = 0; i < d.length; i++)
         d[i] = 100 * Math.random();
      ArrayAlg.Pair p = ArrayAlg.minmax(d);
      System.out.println("min = " + p.getFirst());
      System.out.println("max = " + p.getSecond());
   } // end of main
} // end of class
class ArrayAlg {
     public static class Pair { //A pair of floating-point number
      private double first;
      private double second;
      public Pair(double f, double s) { //Constructs a pair from two floating-point numbers
         first = f;
         second = s;
      } // end constructor
      public double getFirst(){
         return first;
      }
      public double getSecond(){
         return second;
      }
   } //end of class Pair( inner class)
```

Listing 6.9. StaticInnerClassTest

```
   // Computes both the minimum and the maximum of an array
   public static Pair minmax(double[] values)
   {
      double min = Double.POSITIVE_INFINITY;
      double max = Double.NEGATIVE_INFINITY;
      for (double v : values)
      {
         if (min > v) min = v;
         if (max < v) max = v;
      }
      return new Pair(min, max);
   }
} // end of ArrayAlg( outer class)
```