```
1  // Fig. 8.1: Time1.java
2  // Time1 class definition
3  import java.text.DecimalFormat;  // used for number formatting
4
5  // This class maintains the time in 24-hour format
6  public class Time1 extends Object {
7     private int hour;     // 0 - 23
8     private int minute;   // 0 - 59
9     private int second;   // 0 - 59
10
11    // Time1 constructor initializes each instance variable
12    // to zero. Ensures that each Time1 object starts in a
13    // consistent state.
14    public Time1()
15    {
16       setTime( 0, 0, 0 );
17    }
18
19    // Set a new time value using universal time. Perform
20    // validity checks on the data. Set invalid values to zero.
21    public void setTime( int h, int m, int s )
22    {
23       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
24       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
25       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
26    }
27
28    // Convert to String in universal-time format
29    public String toUniversalString()
30    {
31       DecimalFormat twoDigits = new DecimalFormat( "00" );
32
33       return twoDigits.format( hour ) + ":" +
34              twoDigits.format( minute ) + ":" +
35              twoDigits.format( second );
36    }
37
38    // Convert to String in standard-time format
39    public String toString()
40    {
41       DecimalFormat twoDigits = new DecimalFormat( "00" );
42
43       return ( (hour == 12 || hour == 0) ? 12 : hour % 12 ) +
44              ":" + twoDigits.format( minute ) +
45              ":" + twoDigits.format( second ) +
46              ( hour < 12 ? " AM" : " PM" );
47    }
48 }
```

Fig. 8.1 Abstract data type **`Time1`** implementation as a class (part 1 of 2).

```
// Fig. 8.1: TimeTest.java
// Class TimeTest to exercise class Time1
import javax.swing.JOptionPane;

public class TimeTest {
   public static void main( String args[] )
   {
      Time1 t = new Time1();  // calls Time1 constructor
      String output;

      output = "The initial universal time is: " +
               t.toUniversalString() +
               "\nThe initial standard time is: " +
               t.toString() +
               "\nImplicit toString() call: " + t;

      t.setTime( 13, 27, 6 );
      output += "\n\nUniversal time after setTime is: " +
                t.toUniversalString() +
                "\nStandard time after setTime is: " +
                t.toString();

      t.setTime( 99, 99, 99 );  // all invalid values
      output += "\n\nAfter attempting invalid settings: " +
                "\nUniversal time: " + t.toUniversalString() +
                "\nStandard time: " + t.toString();

      JOptionPane.showMessageDialog( null, output,
         "Testing Class Time1",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

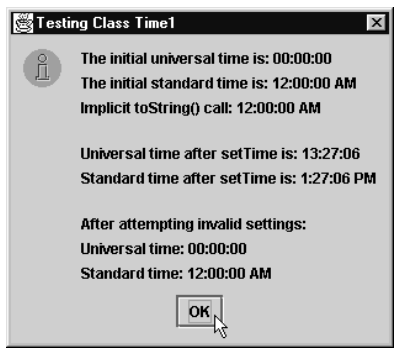


Fig. 8.1 Abstract data type **`Time1`** implementation as a class (part 2 of 2).

```
1  // Fig. 8.2: TimeTest.java
2  // Demonstrate errors resulting from attempts
3  // to access private class members.
4  public class TimeTest {
5     public static void main( String args[] )
6     {
7        Time1 t = new Time1();
8
9        t.hour = 7;
10    }
11 }
```

```
TimeTest.java:9: Variable hour in class Time1 not
                 accessible from class TimeTest.
      t.hour = 7;
       ^
1 error
```

Fig. 8.2 Erroneous attempt to access private members of a class.

```
1  // Fig. 8.3: Time1.java
2  // Time1 class definition
3  package com.deitel.jhtp3.ch08;
4  import java.text.DecimalFormat;  // used for number formatting
5
6  // This class maintains the time in 24-hour format
7  public class Time1 extends Object {
8     private int hour;     // 0 - 23
9     private int minute;   // 0 - 59
10    private int second;   // 0 - 59
11
12    // Time1 constructor initializes each instance variable
13    // to zero. Ensures that each Time1 object starts in a
14    // consistent state.
15    public Time1()
16    {
17       setTime( 0, 0, 0 );
18    }
19
20    // Set a new time value using military time. Perform
21    // validity checks on the data. Set invalid values
22    // to zero.
23    public void setTime( int h, int m, int s )
24    {
25       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
26       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
27       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
28    }
29
30    // Convert to String in universal-time format
31    public String toUniversalString()
32    {
33       DecimalFormat twoDigits = new DecimalFormat( "00" );
34
35       return twoDigits.format( hour ) + ":" +
36              twoDigits.format( minute ) + ":" +
37              twoDigits.format( second );
38    }
39
40    // Convert to String in standard-time format
41    public String toString()
42    {
43       DecimalFormat twoDigits = new DecimalFormat( "00" );
44
45       return ( (hour == 12 || hour == 0) ? 12 : hour % 12 ) +
46              ":" + twoDigits.format( minute ) +
47              ":" + twoDigits.format( second ) +
48              ( hour < 12 ? " AM" : " PM" );
49    }
50 }
```

Fig. 8.3 Creating a package for software reuse (part 1 of 2).

```
// Fig. 8.3: TimeTest.java
// Class TimeTest to use imported class Time1
import javax.swing.JOptionPane;
import com.deitel.jhtp3.ch08.Time1;  // import Time1 class

public class TimeTest {
   public static void main( String args[] )
   {
      Time1 t = new Time1();

      t.setTime( 13, 27, 06 );
      String output =
         "Universal time is: " + t.toUniversalString() +
         "\nStandard time is: " + t.toString();

      JOptionPane.showMessageDialog( null, output,
         "Packaging Class Time1 for Reuse",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

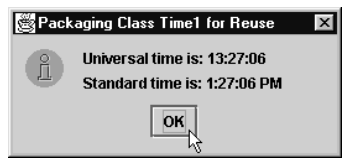


Fig. 8.3 Creating a package for software reuse (part 2 of 2).

```
1  // Fig. 8.4: Time2.java
2  // Time2 class definition
3  package com.deitel.jhtp3.ch08;   // place Time2 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  // This class maintains the time in 24-hour format
7  public class Time2 extends Object {
8     private int hour;     // 0 - 23
9     private int minute;   // 0 - 59
10    private int second;   // 0 - 59
11
12    // Time2 constructor initializes each instance variable
13    // to zero. Ensures that Time object starts in a
14    // consistent state.
15    public Time2() { setTime( 0, 0, 0 ); }
16
17    // Time2 constructor: hour supplied, minute and second
18    // defaulted to 0.
19    public Time2( int h ) { setTime( h, 0, 0 ); }
20
21    // Time2 constructor: hour and minute supplied, second
22    // defaulted to 0.
23    public Time2( int h, int m ) { setTime( h, m, 0 ); }
24
25    // Time2 constructor: hour, minute and second supplied.
26    public Time2( int h, int m, int s ) { setTime( h, m, s ); }
27
28    // Time2 constructor: another Time2 object supplied.
29    public Time2( Time2 time )
30    {
31       setTime( time.hour, time.minute, time.second );
32    }
33
34    // Set a new time value using universal time. Perform
35    // validity checks on the data. Set invalid values to zero.
36    public void setTime( int h, int m, int s )
37    {
38       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
39       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
40       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
41    }
42
43    // Convert to String in universal-time format
44    public String toUniversalString()
45    {
46       DecimalFormat twoDigits = new DecimalFormat( "00" );
47
```

Fig. 8.4 Using overloaded constructors (part 1 of 4).

```
48          return twoDigits.format( hour ) + ":" +
49                 twoDigits.format( minute ) + ":" +
50                 twoDigits.format( second );
51       }
52
53       // Convert to String in standard-time format
54       public String toString()
55       {
56          DecimalFormat twoDigits = new DecimalFormat( "00" );
57
58          return ( (hour == 12 || hour == 0) ? 12 : hour % 12 ) +
59                 ":" + twoDigits.format( minute ) +
60                 ":" + twoDigits.format( second ) +
61                 ( hour < 12 ? " AM" : " PM" );
62       }
63    }
```

Fig. 8.4 Using overloaded constructors (part 2 of 4).

```
64 // Fig. 8.4: TimeTest.java
65 // Using overloaded constructors
66 import javax.swing.*;
67 import com.deitel.jhtp3.ch08.Time2;
68
69 public class TimeTest {
70    public static void main( String args[] )
71    {
72       Time2 t1, t2, t3, t4, t5, t6;
73       String output;
74
75       t1 = new Time2();
76       t2 = new Time2( 2 );
77       t3 = new Time2( 21, 34 );
78       t4 = new Time2( 12, 25, 42 );
79       t5 = new Time2( 27, 74, 99 );
80       t6 = new Time2( t4 );   // use t4 as initial value
81
82       output = "Constructed with: " +
83                "\nt1: all arguments defaulted" +
84                "\n      " + t1.toUniversalString() +
85                "\n      " + t1.toString();
86
87       output += "\nt2: hour specified; minute and " +
88                 "second defaulted" +
89                 "\n      " + t2.toUniversalString() +
90                 "\n      " + t2.toString();
91
92       output += "\nt3: hour and minute specified; " +
93                 "second defaulted" +
94                 "\n      " + t3.toUniversalString() +
95                 "\n      " + t3.toString();
96
```

Fig. 8.4 Using overloaded constructors (part 3 of 4).

```
97          output += "\nt4: hour, minute, and second specified" +
98                    "\n       " + t4.toUniversalString() +
99                    "\n       " + t4.toString();
100
101         output += "\nt5: all invalid values specified" +
102                   "\n       " + t5.toUniversalString() +
103                   "\n       " + t5.toString();
104
105         output += "\nt6: Time2 object t4 specified" +
106                   "\n       " + t6.toUniversalString() +
107                   "\n       " + t6.toString();
108
109         JOptionPane.showMessageDialog( null, output,
110            "Demonstrating Overloaded Constructors",
111            JOptionPane.INFORMATION_MESSAGE );
112
113         System.exit( 0 );
114      }
115   }
```

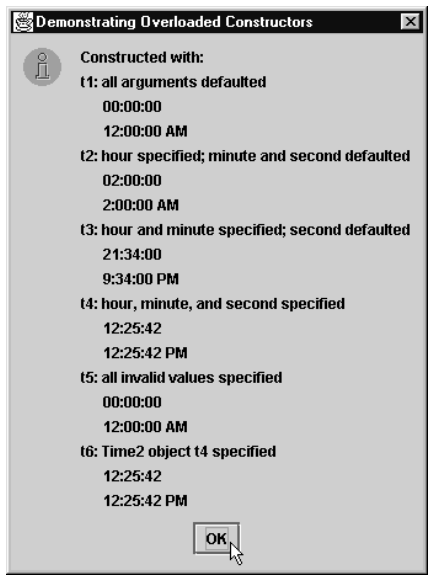


Fig. 8.4 Using overloaded constructors (part 4 of 4).

```
1  // Fig. 8.5: Time3.java
2  // Time3 class definition
3  package com.deitel.jhtp3.ch08;   // place Time3 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  // This class maintains the time in 24-hour format
7  public class Time3 extends Object {
8     private int hour;     // 0 - 23
9     private int minute;   // 0 - 59
10    private int second;   // 0 - 59
11
12    // Time3 constructor initializes each instance variable
13    // to zero. Ensures that Time object starts in a
14    // consistent state.
15    public Time3() { setTime( 0, 0, 0 ); }
16
17    // Time3 constructor: hour supplied, minute and second
18    // defaulted to 0.
19    public Time3( int h ) { setTime( h, 0, 0 ); }
20
21    // Time3 constructor: hour and minute supplied, second
22    // defaulted to 0.
23    public Time3( int h, int m ) { setTime( h, m, 0 ); }
24
25    // Time3 constructor: hour, minute and second supplied.
26    public Time3( int h, int m, int s ) { setTime( h, m, s ); }
27
28    // Time3 constructor: another Time3 object supplied.
29    public Time3( Time3 time )
30    {
31       setTime( time.getHour(),
32                time.getMinute(),
33                time.getSecond() );
34    }
35
36    // Set Methods
37    // Set a new time value using universal time. Perform
38    // validity checks on the data. Set invalid values to zero.
39    public void setTime( int h, int m, int s )
40    {
41       setHour( h );    // set the hour
42       setMinute( m );  // set the minute
43       setSecond( s );  // set the second
44    }
45
46    // set the hour
47    public void setHour( int h )
48       { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
```

Fig. 8.5 Using ***set*** and ***get*** methods (part 1 of 6).

```
49
50      // set the minute
51      public void setMinute( int m )
52         { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
53
54      // set the second
55      public void setSecond( int s )
56         { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
57
58      // Get Methods
59      // get the hour
60      public int getHour() { return hour; }
61
62      // get the minute
63      public int getMinute() { return minute; }
64
65      // get the second
66      public int getSecond() { return second; }
67
68      // Convert to String in universal-time format
69      public String toUniversalString()
70      {
71         DecimalFormat twoDigits = new DecimalFormat( "00" );
72
73         return twoDigits.format( getHour() ) + ":" +
74                twoDigits.format( getMinute() ) + ":" +
75                twoDigits.format( getSecond() );
76      }
77
78      // Convert to String in standard-time format
79      public String toString()
80      {
81         DecimalFormat twoDigits = new DecimalFormat( "00" );
82
83         return ( ( getHour() == 12 || getHour() == 0 ) ?
84                   12 : getHour() % 12 ) + ":" +
85                twoDigits.format( getMinute() ) + ":" +
86                twoDigits.format( getSecond() ) +
87                ( getHour() < 12 ? " AM" : " PM" );
88      }
89   }
```

Fig. 8.5 Using ***set*** and ***get*** methods (part 2 of 6).

```
90  // Fig. 8.5: TimeTest.java
91  // Demonstrating the Time3 class set and get methods
92  import java.awt.*;
93  import java.awt.event.*;
94  import javax.swing.*;
95  import com.deitel.jhtp3.ch08.Time3;
96
97  public class TimeTest extends JApplet
98                       implements ActionListener {
99     private Time3 t;
100    private JLabel hourLabel, minuteLabel, secondLabel;
101    private JTextField hourField, minuteField,
102                       secondField, display;
103    private JButton tickButton;
104
105    public void init()
106    {
107       t = new Time3();
108
109       Container c = getContentPane();
110
111       c.setLayout( new FlowLayout() );
112       hourLabel = new JLabel( "Set Hour" );
113       hourField = new JTextField( 10 );
114       hourField.addActionListener( this );
115       c.add( hourLabel );
116       c.add( hourField );
117
118       minuteLabel = new JLabel( "Set minute" );
119       minuteField = new JTextField( 10 );
120       minuteField.addActionListener( this );
121       c.add( minuteLabel );
122       c.add( minuteField );
123
124       secondLabel = new JLabel( "Set Second" );
125       secondField = new JTextField( 10 );
126       secondField.addActionListener( this );
127       c.add( secondLabel );
128       c.add( secondField );
129
130       display = new JTextField( 30 );
131       display.setEditable( false );
132       c.add( display );
133
134       tickButton = new JButton( "Add 1 to Second" );
135       tickButton.addActionListener( this );
136       c.add( tickButton );
137
138       updateDisplay();
139    }
140
```

Fig. 8.5 Using ***set*** and ***get*** methods (part 3 of 6).

```
141    public void actionPerformed( ActionEvent e )
142    {
143       if ( e.getSource() == tickButton )
144          tick();
145       else if ( e.getSource() == hourField ) {
146          t.setHour(
147             Integer.parseInt( e.getActionCommand() ) );
148          hourField.setText( "" );
149       }
150       else if ( e.getSource() == minuteField ) {
151          t.setMinute(
152             Integer.parseInt( e.getActionCommand() ) );
153          minuteField.setText( "" );
154       }
155       else if ( e.getSource() == secondField ) {
156          t.setSecond(
157             Integer.parseInt( e.getActionCommand() ) );
158          secondField.setText( "" );
159       }
160
161       updateDisplay();
162    }
163
164    public void updateDisplay()
165    {
166       display.setText( "Hour: " + t.getHour() +
167          "; Minute: " + t.getMinute() +
168          "; Second: " + t.getSecond() );
169       showStatus( "Standard time is: " + t.toString() +
170          "; Universal time is: " + t.toUniversalString() );
171    }
172
173    public void tick()
174    {
175       t.setSecond( ( t.getSecond() + 1 ) % 60 );
176
177       if ( t.getSecond() == 0 ) {
178          t.setMinute( ( t.getMinute() + 1 ) % 60 );
179
180          if ( t.getMinute() == 0 )
181             t.setHour( ( t.getHour() + 1 ) % 24 );
182       }
183    }
184 }
```

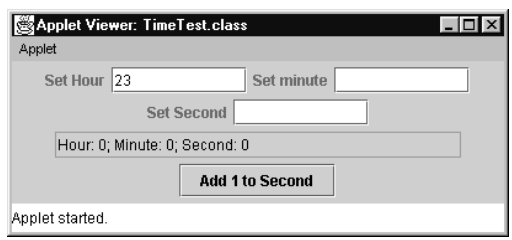
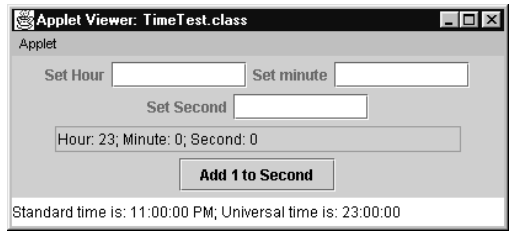
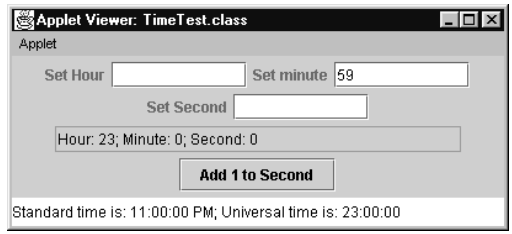
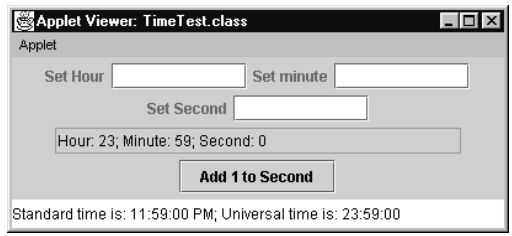
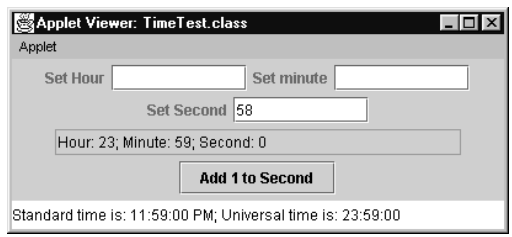


Fig. 8.5 Using ***set*** and ***get*** methods (part 4 of 6).









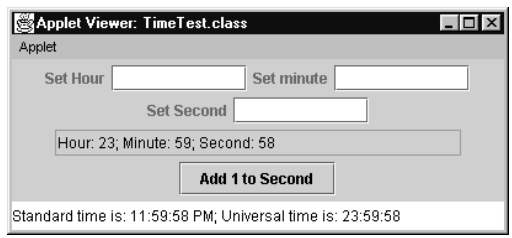


Fig. 8.5 Using ***set*** and ***get*** methods (part 5 of 6).

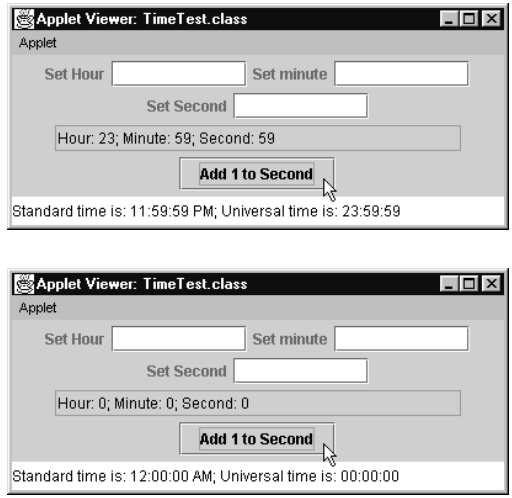


Fig. 8.5 Using ***set*** and ***get*** methods (part 6 of 6).

```
1  // Fig. 8.6: Increment.java
2  // Initializing a final variable
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Increment extends JApplet
8                implements ActionListener {
9     private int count = 0, total = 0;
10    private final int INCREMENT = 5; // constant variable
11
12    private JButton incr;
```

Fig. 8.6 Initializing a **`final`** variable (part 1 of 2).

```
13
14    public void init()
15    {
16       Container c = getContentPane();
17
18       incr = new JButton( "Click to increment" );
19       incr.addActionListener( this );
20       c.add( incr );
21    }
22
23    public void actionPerformed( ActionEvent e )
24    {
25       total += INCREMENT;
26       count++;
27       showStatus( "After increment " + count +
28                   ": total = " + total );
29    }
30 }
```

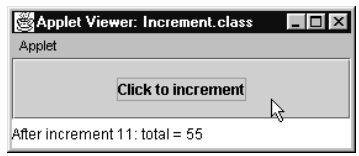


Fig. 8.6 Initializing a **`final`** variable (part 2 of 2).

```
Increment.java:10: Blank final variable 'increment' may
not have been initialized. It must be assigned a value in
an initializer, or in every constructor.
   private final int increment; // constant variable
                             ^

1 error
```

Fig. 8.7 Compiler error message as a result of not initializing **`increment`**.

```
1  // Fig. 8.8: Date.java
2  // Declaration of the Date class.
3  package com.deitel.jhtp3.ch08;
4
5  public class Date extends Object {
6     private int month;  // 1-12
7     private int day;    // 1-31 based on month
8     private int year;   // any year
9
10    // Constructor: Confirm proper value for month;
11    // call method checkDay to confirm proper
12    // value for day.
13    public Date( int mn, int dy, int yr )
14    {
15       if ( mn > 0 && mn <= 12 )       // validate the month
16          month = mn;
17       else {
18          month = 1;
19          System.out.println( "Month " + mn +
20                              " invalid. Set to month 1." );
21       }
22
23       year = yr;                      // could also check
24       day = checkDay( dy );           // validate the day
25
26       System.out.println(
27          "Date object constructor for date " + toString() );
28    }
29
30    // Utility method to confirm proper day value
31    // based on month and year.
32    private int checkDay( int testDay )
33    {
34       int daysPerMonth[] = { 0, 31, 28, 31, 30,
35                              31, 30, 31, 31, 30,
36                              31, 30, 31 };
37
38       if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
39          return testDay;
40
```

Fig. 8.8 Demonstrating an object with a member object reference (part 1 of 4).

```
41          if ( month == 2 &&   // February: Check for leap year
42               testDay == 29 &&
43               ( year % 400 == 0 ||
44                 ( year % 4 == 0 && year % 100 != 0 ) ) )
45             return testDay;
46
47          System.out.println( "Day " + testDay +
48                              " invalid. Set to day 1." );
49
50          return 1;  // leave object in consistent state
51       }
52
53       // Create a String of the form month/day/year
54       public String toString()
55          { return month + "/" + day + "/" + year; }
56    }
```

Fig. 8.8 Demonstrating an object with a member object reference (part 2 of 4).

```
57 // Fig. 8.8: Employee.java
58 // Declaration of the Employee class.
59 package com.deitel.jhtp3.ch08;
60
61 public class Employee extends Object {
62    private String firstName;
63    private String lastName;
64    private Date birthDate;
65    private Date hireDate;
66
67    public Employee( String fName, String lName,
68                     int bMonth, int bDay, int bYear,
69                     int hMonth, int hDay, int hYear)
70    {
71       firstName = fName;
72       lastName = lName;
73       birthDate = new Date( bMonth, bDay, bYear );
74       hireDate = new Date( hMonth, hDay, hYear );
75    }
76
77    public String toString()
78    {
79       return lastName + ", " + firstName +
80              "  Hired: " + hireDate.toString() +
81              "  Birthday: " + birthDate.toString();
82    }
83 }
```

Fig. 8.8 Demonstrating an object with a member object reference (part 3 of 4).

```
84 // Fig. 8.8: EmployeeTest.java
85 // Demonstrating an object with a member object.
86 import javax.swing.JOptionPane;
87 import com.deitel.jhtp3.ch08.Employee;
88
89 public class EmployeeTest {
90    public static void main( String args[] )
91    {
92       Employee e = new Employee( "Bob", "Jones", 7, 24, 49,
93                                  3, 12, 88 );
94       JOptionPane.showMessageDialog( null, e.toString(),
95          "Testing Class Employee",
96          JOptionPane.INFORMATION_MESSAGE );
97
98       System.exit( 0 );
99    }
100 }
```


```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
```

Fig. 8.8 Demonstrating an object with a member object reference (part 4 of 4).

```
1  // Fig. 8.9: PackageDataTest.java
2  // Classes in the same package (i.e., the same directory)
3  // can use package access data of other classes in the
4  // same package.
5  import javax.swing.JOptionPane;
6
7  public class PackageDataTest {
8     public static void main( String args[] )
9     {
10       PackageData d = new PackageData();
11       String output;
12
13       output = "After instantiation:\n" + d.toString();
14
15       d.x = 77;          // changing package access data
16       d.s = "Good bye";  // changing package access data
17
18       output += "\nAfter changing values:\n" + d.toString();
19       JOptionPane.showMessageDialog( null, output,
20          "Demonstrating Package Access",
21          JOptionPane.INFORMATION_MESSAGE );
22
23       System.exit( 0 );
24    }
25 }
26
27 class PackageData {
28    int x;     // package access instance variable
29    String s;  // package access instance variable
30
31    // constructor
32    public PackageData()
33    {
34       x = 0;
35       s = "Hello";
36    }
37
38    public String toString()
39    {
40       return "x: " + x + "    s: " + s;
41    }
42 }
```

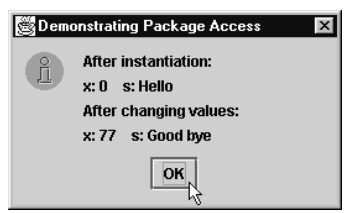


Fig. 8.9 Package access to members of a class.

```
1  // Fig. 8.10: ThisTest.java
2  // Using the this reference to refer to
3  // instance variables and methods.
4  import javax.swing.*;
5  import java.text.DecimalFormat;
6
7  public class ThisTest {
8     public static void main( String args[] )
9     {
10       SimpleTime t = new SimpleTime( 12, 30, 19 );
11
12       JOptionPane.showMessageDialog( null, t.buildString(),
13          "Demonstrating the \"this\" Reference",
14          JOptionPane.INFORMATION_MESSAGE );
15
16       System.exit( 0 );
17    }
18 }
19
20 class SimpleTime {
21    private int hour, minute, second;
22
23    public SimpleTime( int hour, int minute, int second )
24    {
25       this.hour = hour;
26       this.minute = minute;
27       this.second = second;
28    }
29
30    public String buildString()
31    {
32       return "this.toString(): " + this.toString() +
33              "\ntoString(): " + toString() +
34              "\nthis (with implicit toString() call): " +
35              this;
36    }
37
38    public String toString()
39    {
40       DecimalFormat twoDigits = new DecimalFormat( "00" );
41
42       return twoDigits.format( this.hour ) + ":" +
43              twoDigits.format( this.minute ) + ":" +
44              twoDigits.format( this.second );
45    }
46 }
```

Fig. 8.10 Using the **this** reference (part 1 of 2).

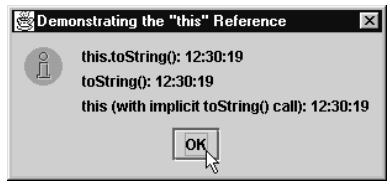


Fig. 8.10 Using the `this` reference (part 2 of 2).

```
1  // Fig. 8.11: Time4.java
2  // Time4 class definition
3  package com.deitel.jhtp3.ch08;   // place Time4 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  // This class maintains the time in 24-hour format
7  public class Time4 extends Object {
8     private int hour;     // 0 - 23
9     private int minute;   // 0 - 59
10    private int second;   // 0 - 59
11
12    // Time4 constructor initializes each instance variable
13    // to zero. Ensures that Time object starts in a
14    // consistent state.
15    public Time4() { this.setTime( 0, 0, 0 ); }
16
17    // Time4 constructor: hour supplied, minute and second
18    // defaulted to 0.
19    public Time4( int h ) { this.setTime( h, 0, 0 ); }
20
21    // Time4 constructor: hour and minute supplied, second
22    // defaulted to 0.
23    public Time4( int h, int m ) { this.setTime( h, m, 0 ); }
24
25    // Time4 constructor: hour, minute and second supplied.
26    public Time4( int h, int m, int s )
27       { this.setTime( h, m, s ); }
28
29    // Time4 constructor: another Time4 object supplied.
30    public Time4( Time4 time )
31    {
32       this.setTime( time.getHour(),
33                     time.getMinute(),
34                     time.getSecond() );
35    }
36
```

Fig. 8.11 Chaining method calls (part 1 of 5).

```
37    // Set Methods
38    // Set a new Time value using military time. Perform
39    // validity checks on the data. Set invalid values to zero.
40    public Time4 setTime( int h, int m, int s )
41    {
42       this.setHour( h );    // set the hour
43       this.setMinute( m );  // set the minute
44       this.setSecond( s );  // set the second
45
46       return this;     // enables chaining
47    }
48
49    // set the hour
50    public Time4 setHour( int h )
51    {
52       this.hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
53
54       return this;     // enables chaining
55    }
56
57    // set the minute
58    public Time4 setMinute( int m )
59    {
60       this.minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
61
62       return this;     // enables chaining
63    }
64
65    // set the second
66    public Time4 setSecond( int s )
67    {
68       this.second = ( ( s >= 0 && s < 60 ) ? s : 0 );
69
70       return this;     // enables chaining
71    }
72
73    // Get Methods
74    // get the hour
75    public int getHour() { return this.hour; }
76
77    // get the minute
78    public int getMinute() { return this.minute; }
79
80    // get the second
81    public int getSecond() { return this.second; }
82
83    // Convert to String in universal-time format
84    public String toUniversalString()
85    {
86       DecimalFormat twoDigits = new DecimalFormat( "00" );
87
```

Fig. 8.11 Chaining method calls (part 2 of 5).

```
88          return twoDigits.format( this.getHour() ) + ":" +
89                 twoDigits.format( this.getMinute() ) + ":" +
90                 twoDigits.format( this.getSecond() );
91       }
92
93       // Convert to String in standard-time format
94       public String toString()
95       {
96          DecimalFormat twoDigits = new DecimalFormat( "00" );
97
98          return ( ( this.getHour() == 12 ||
99                     this.getHour() == 0 ) ?
100                    12 : this.getHour() % 12 ) + ":" +
101                twoDigits.format( this.getMinute() ) + ":" +
102                twoDigits.format( this.getSecond() ) +
103                ( this.getHour() < 12 ? " AM" : " PM" );
104      }
105   }
```

Fig. 8.11 Chaining method calls (part 3 of 5).

```
// Fig. 8.11: TimeTest.java
// Chaining method calls together with the this reference
import javax.swing.*;
import com.deitel.jhtp3.ch08.Time4;

public class TimeTest {
   public static void main( String args[] )
   {
      Time4 t = new Time4();
      String output;

      t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );

      output = "Universal time: " + t.toUniversalString() +
               "\nStandard time: " + t.toString() +
               "\n\nNew standard time: " +
               t.setTime( 20, 20, 20 ).toString();

      JOptionPane.showMessageDialog( null, output,
         "Chaining Method Calls",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

Fig. 8.11 Chaining method calls (part 4 of 5).

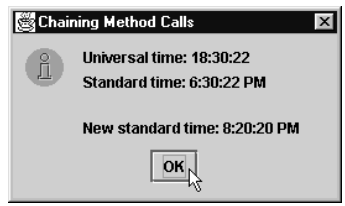


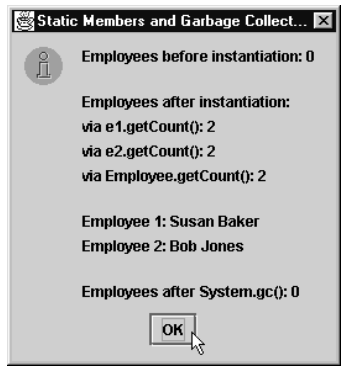
Fig. 8.11 Chaining method calls (part 5 of 5).

```
1  // Fig. 8.12: Employee.java
2  // Declaration of the Employee class.
3  public class Employee extends Object {
4     private String firstName;
5     private String lastName;
6     private static int count;  // # of objects in memory
7
8     public Employee( String fName, String lName )
9     {
10       firstName = fName;
11       lastName = lName;
12
13       ++count;  // increment static count of employees
14       System.out.println( "Employee object constructor: " +
15                           firstName + " " + lastName );
16    }
17
18    protected void finalize()
19    {
20       --count;  // decrement static count of employees
21       System.out.println( "Employee object finalizer: " +
22                           firstName + " " + lastName +
23                           "; count = " + count );
24    }
25
26    public String getFirstName() { return firstName; }
27
28    public String getLastName() { return lastName; }
29
30    public static int getCount() { return count; }
31 }
```

Fig. 8.12 Using a **`static`** class variable to maintain a count of the number of objects of a class (part 1 of 3).

```
// Fig. 8.12: EmployeeTest.java
// Test Employee class with static class variable,
// static class method, and dynamic memory.
import javax.swing.*;

public class EmployeeTest {
   public static void main( String args[] )
   {
      String output;

      output = "Employees before instantiation: " +
               Employee.getCount();

      Employee e1 = new Employee( "Susan", "Baker" );
      Employee e2 = new Employee( "Bob", "Jones" );

      output += "\n\nEmployees after instantiation: " +
                "\nvia e1.getCount(): " + e1.getCount() +
                "\nvia e2.getCount(): " + e2.getCount() +
                "\nvia Employee.getCount(): " +
                Employee.getCount();

      output += "\n\nEmployee 1: " + e1.getFirstName() +
                " " + e1.getLastName() +
                "\nEmployee 2: " + e2.getFirstName() +
                " " + e2.getLastName();

      // mark objects referred to by e1 and e2
      // for garbage collection
      e1 = null;
      e2 = null;

      System.gc(); // suggest that garbage collector be called

      output += "\n\nEmployees after System.gc(): " +
                Employee.getCount();

      JOptionPane.showMessageDialog( null, output,
         "Static Members and Garbage Collection",
         JOptionPane.INFORMATION_MESSAGE );
      System.exit( 0 );
   }
}
```

Fig. 8.12 Using a **`static`** class variable to maintain a count of the number of objects of a class (part 2 of 3).




```
Employee object constructor: Susan Baker
Employee object constructor: Bob Jones
Employee object finalizer: Susan Baker; count = 1
Employee object finalizer: Bob Jones; count = 0
```

Fig. 8.12 Using a **`static`** class variable to maintain a count of the number of objects of a class (part 3 of 3).