Chapter 6 Object-Based Programming (Main Page)

```
1  // Fig. 6.1: Time1.java
2  // Time1 class definition
3  import java.text.DecimalFormat;  // used for number formatting
4
5  public class Time1 {
6     private int hour;     // 0 - 23
7     private int minute;   // 0 - 59
8     private int second;   // 0 - 59
9
10    // Time1 constructor initializes each instance variable
11    // to zero. Ensures that each Time1 object starts in a
12    // consistent state.
13    public Time1() { setTime( 0, 0, 0 ); }
14
15    // Set a new time value using military time. Perform
16    // validity checks on the data. Set invalid values
17    // to zero.
18    public void setTime( int h, int m, int s )
19    {
20       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
21       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
22       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
23    }
24
25    // Convert time to String in military-time format
26    public String toMilitaryString()
27    {
28       DecimalFormat twoDigits = new DecimalFormat( "00" );
29
30       return twoDigits.format( hour ) +
31              twoDigits.format( minute );
32    }
33
34    // Convert time to String in standard-time format
35    public String toString()
36    {
37       DecimalFormat twoDigits = new DecimalFormat( "00" );
38
39       return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
40              ":" + twoDigits.format( minute ) +
41              ":" + twoDigits.format( second ) +
42              ( hour < 12 ? " AM" : " PM" );
43    }
44 }
```

Fig. 6.1 Abstract data type **Time1** implementation as a class (part 1 of 3).

```
45 // Fig. 6.1: TimeTest.java
46 // Class TimeTest to exercise class Time1
47 import java.awt.Graphics;
48 import java.applet.Applet;
49
50 public class TimeTest extends Applet {
51    private Time1 t;
52
53    public void init()
54    {
55       t = new Time1();
```

```
56      }
57
58      public void paint( Graphics g )
59      {
60         g.drawString( "The initial military time is: " +
61                       t.toMilitaryString(), 25, 25 );
62         g.drawString( "The initial standard time is: " +
63                       t.toString(), 25, 40 );
64
65         t.setTime( 13, 27, 6 );
66         g.drawString( "Military time after setTime is: " +
67                       t.toMilitaryString(), 25, 70 );
68         g.drawString( "Standard time after setTime is: " +
69                       t.toString(), 25, 85 );
70
71         t.setTime( 99, 99, 99 );
72         g.drawString( "After attempting invalid settings:",
73                       25, 115 );
74         g.drawString( "Military time: " +
75                       t.toMilitaryString(), 25, 130 );
76         g.drawString( "Standard time: " + t.toString(),
77                       25, 145 );
78      }
79   }
```

Fig. 6.1 Abstract data type **Time1** implementation as a class (part 2 of 3).

.

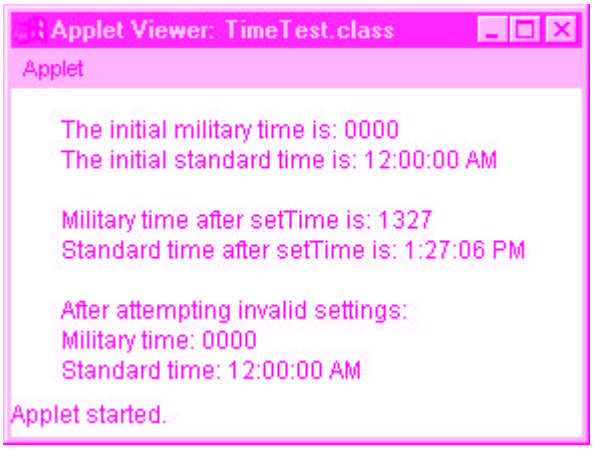


Fig. 6.1 Abstract data type **Time1** implementation as a class (part 3 of 3).

```
1  // Fig. 6.2: TimeTest.java
2  // Demonstrate errors resulting from attempts
3  // to access private class members.
4  import java.awt.Graphics;
5  import java.applet.Applet;
6
7  public class TimeTest extends Applet {
8     private Time1 t;
9
10    public void init()
11    {
12       t = new Time1();
13    }
14
15    public void paint( Graphics g )
16    {
17       t.hour = 7;
18
19       g.drawString( "minute = " + t.minute, 25, 25 );
20    }
21 }
```

Fig. 6.2 Erroneous attempt to access private members of a class (part 1 of 2).

```
TimeTest.java:17: Variable hour in class Time1 not
                  accessible from class TimeTest.
      t.hour = 7;
       ^
TimeTest.java:19: Variable minute in class Time1 not
                  accessible from class TimeTest.
      g.drawString( "minute = " + t.minute, 25, 25 );
                                  ^
2 errors
```

Fig. 6.2 Erroneous attempt to access private members of a class (part 2 of 2).

```
1  // Fig. 6.3: Time1.java
2  // Time1 class definition
3  package com.deitel.jhtp2.ch06;   // place Time1 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  public class Time1 {
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() { setTime( 0, 0, 0 ); }
15
16    // Set a new time value using military time. Perform
17    // validity checks on the data. Set invalid values
18    // to zero.
19    public void setTime( int h, int m, int s )
20    {
21       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
```

```
22          minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
23          second = ( ( s >= 0 && s < 60 ) ? s : 0 );
24       }
25
26       // Convert time to String in military-time format
27       public String toMilitaryString()
28       {
29          DecimalFormat twoDigits = new DecimalFormat( "00" );
30
31          return twoDigits.format( hour ) +
32                 twoDigits.format( minute );
33       }
34
35       // Convert time to String in standard-time format
36       public String toString()
37       {
38          DecimalFormat twoDigits = new DecimalFormat( "00" );
39
40          return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
41                 ":" + twoDigits.format( minute ) +
42                 ":" + twoDigits.format( second ) +
43                 ( hour < 12 ? " AM" : " PM" );
44       }
45    }
```

Fig. 6.3 Creating a package for software reuse (part 1 of 2).

```
46    // Fig. 6.3: TimeTest.java
47    // Class TimeTest to use imported class Time1
48    import java.awt.Graphics;
49    import java.applet.Applet;
50    import com.deitel.jhtp2.ch06.Time1;  // import our Time1 class
51
52    public class TimeTest extends Applet {
53       private Time1 t;
54
55       public void init()
56       {
57          t = new Time1();
58          t.setTime( 13, 27, 06 );
59       }
60
61       public void paint( Graphics g )
62       {
63          g.drawString( "Military time is: " +
64                        t.toMilitaryString(), 25, 25 );
65          g.drawString( "Standard time is: " +
66                        t.toString(), 25, 40 );
67       }
68    }
```

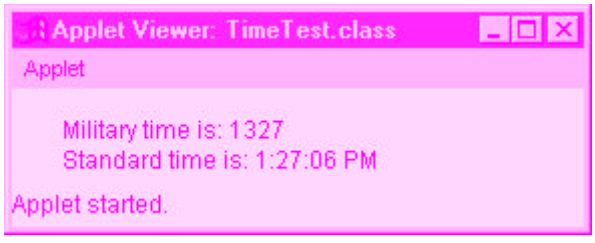


Fig. 6.3 Creating a package for software reuse (part 2 of 2).

```
1  // Fig. 6.4: Time2.java
2  // Time2 class definition
3  package com.deitel.jhtp2.ch06;   // place Time2 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  public class Time2 {
7     private int hour;     // 0 - 23
8     private int minute;   // 0 - 59
9     private int second;   // 0 - 59
10
11    // Time2 constructor initializes each instance variable
12    // to zero. Ensures that Time object starts in a
13    // consistent state.
14    public Time2() { setTime( 0, 0, 0 ); }
15
16    // Time2 constructor: hour supplied, minute and second
17    // defaulted to 0.
18    public Time2( int h ) { setTime( h, 0, 0 ); }
19
20    // Time2 constructor: hour and minute supplied, second
21    // defaulted to 0.
22    public Time2( int h, int m ) { setTime( h, m, 0 ); }
23
24    // Time2 constructor: hour, minute and second supplied.
25    public Time2( int h, int m, int s ) { setTime( h, m, s ); }
26
27    // Set a new Time value using military time. Perform
28    // validity checks on the data. Set invalid values
29    // to zero.
30    public void setTime( int h, int m, int s )
31    {
32       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
33       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
34       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
35    }
36
37    // Convert time to String in military-time format
38    public String toMilitaryString()
39    {
40       DecimalFormat twoDigits = new DecimalFormat( "00" );
41
42       return twoDigits.format( hour ) +
43              twoDigits.format( minute );
44    }
45
```

Fig. 6.4 Using overloaded constructors (part 1 of 4).

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

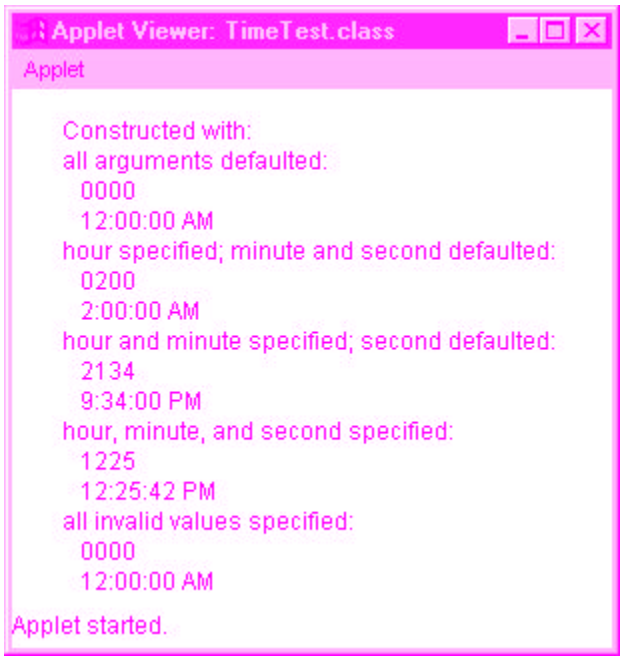
Fig. 6.4 Using overloaded constructors (part 2 of 4).

```
57  // Fig. 6.4: TimeTest.java
58  // Using overloaded constructors
59  import java.awt.Graphics;
60  import java.applet.Applet;
61  import com.deitel.jhtp2.ch06.Time2;
62
63  public class TimeTest extends Applet {
64     private Time2 t1, t2, t3, t4, t5;
65
66     public void init()
67     {
68        t1 = new Time2();
69        t2 = new Time2( 2 );
70        t3 = new Time2( 21, 34 );
71        t4 = new Time2( 12, 25, 42 );
72        t5 = new Time2( 27, 74, 99 );
73     }
74
75     public void paint( Graphics g )
76     {
77        g.drawString( "Constructed with:", 25, 25 );
78        g.drawString( "all arguments defaulted:", 25, 40 );
79        g.drawString( "   " + t1.toMilitaryString(),
80                      25, 55 );
81        g.drawString( "   " + t1.toString(), 25, 70 );
82
83        g.drawString( "hour specified; minute " +
84                      "and second defaulted:", 25, 85 );
85        g.drawString( "   " + t2.toMilitaryString(),
86                      25, 100 );
87        g.drawString( "   " + t2.toString(), 25, 115 );
88
```

Fig. 6.4 Using overloaded constructors (part 3 of 4).

```
89        g.drawString( "hour and minute specified; " +
90                      "second defaulted:", 25, 130 );
91        g.drawString( "   " + t3.toMilitaryString(),
92                      25, 145 );
93        g.drawString( "   " + t3.toString(), 25, 160 );
94
95        g.drawString( "hour, minute, and second specified:",
96                      25, 175 );
97        g.drawString( "   " + t4.toMilitaryString(),
98                      25, 190 );
99        g.drawString( "   " + t4.toString(), 25, 205 );
100
101       g.drawString( "all invalid values specified:",
102                     25, 220 );
103       g.drawString( "   " + t5.toMilitaryString(),
104                     25, 235 );
105       g.drawString( "   " + t5.toString(), 25, 250 );
106    }
107 }
```

Fig. 6.4 Using overloaded constructors (part 4 of 4).



```
1  // Fig. 6.5: Time3.java
2  // Time3 class definition
3  package com.deitel.jhtp2.ch06;   // place Time3 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  public class Time3 {
7     private int hour;     // 0 - 23
8     private int minute;   // 0 - 59
9     private int second;   // 0 - 59
10
11    // Time3 constructor initializes each instance variable
12    // to zero. Ensures that Time object starts in a
13    // consistent state.
14    public Time3() { setTime( 0, 0, 0 ); }
15
16    // Time3 constructor: hour supplied, minute and second
17    // defaulted to 0.
18    public Time3( int h ) { setTime( h, 0, 0 ); }
19
20    // Time3 constructor: hour and minute supplied, second
21    // defaulted to 0.
22    public Time3( int h, int m ) { setTime( h, m, 0 ); }
23
24    // Time3 constructor: hour, minute and second supplied.
25    public Time3( int h, int m, int s ) { setTime( h, m, s ); }
26
27    // Set Methods
28    // Set a new Time3 value using military time. Perform
29    // validity checks on the data. Set invalid values
30    // to zero.
31    public void setTime( int h, int m, int s )
32    {
33       setHour( h );    // set the hour
34       setMinute( m );  // set the minute
35       setSecond( s );  // set the second
36    }
37
```

Fig. 6.5 Using set and get methods (part 1 of 7).

```
38    // set the hour
39    public void setHour( int h )
40       { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
41
42    // set the minute
43    public void setMinute( int m )
44       { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
45
46    // set the second
47    public void setSecond( int s )
48       { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
49
50    // Get Methods
51    // get the hour
52    public int getHour() { return hour; }
53
54    // get the minute
55    public int getMinute() { return minute; }
56
57    // get the second
58    public int getSecond() { return second; }
59
```

```
60     // Convert time to String in military-time format
61     public String toMilitaryString()
62     {
63        DecimalFormat twoDigits = new DecimalFormat( "00" );
64
65        return twoDigits.format( hour ) +
66               twoDigits.format( minute );
67     }
68
69     // Convert time to String in standard-time format
70     public String toString()
71     {
72        DecimalFormat twoDigits = new DecimalFormat( "00" );
73
74        return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
75               ":" + twoDigits.format( minute ) +
76               ":" + twoDigits.format( second ) +
77               ( hour < 12 ? " AM" : " PM" );
78     }
79  }
```

Fig. 6.5 Using set and get methods (part 2 of 7).

```
80  // Fig. 6.5: TimeTest.java
81  // Demonstrating the Time3 class set and get methods
82  import java.awt.*;
83  import java.awt.event.*;
84  import java.applet.Applet;
85  import com.deitel.jhtp2.ch06.Time3;
86
87  public class TimeTest extends Applet implements ActionListener {
88     private Time3 t;
89     private Label hourLabel, minuteLabel, secondLabel;
90     private TextField hourField, minuteField,
91                       secondField, display;
92     private Button tickButton;
93
94     public void init()
95     {
96        t = new Time3();
97
98        hourLabel = new Label( "Set Hour" );
99        hourField = new TextField( 10 );
100       hourField.addActionListener( this );
101       add( hourLabel );
102       add( hourField );
103
104       minuteLabel = new Label( "Set minute" );
105       minuteField = new TextField( 10 );
106       minuteField.addActionListener( this );
107       add( minuteLabel );
108       add( minuteField );
109
110       secondLabel = new Label( "Set Second" );
111       secondField = new TextField( 10 );
112       secondField.addActionListener( this );
113       add( secondLabel );
114       add( secondField );
115
116       display = new TextField( 30 );
117       display.setEditable( false );
118       add( display );
119
120       tickButton = new Button( "Add 1 to Second" );
```

```
121         tickButton.addActionListener( this );
122         add( tickButton );
123
124         updateDisplay();
125      }
```

Fig. 6.5 Using set and get methods (part 3 of 7).

```
126
127      public void actionPerformed( ActionEvent e )
128      {
129         if ( e.getSource() == tickButton )
130            tick();
131         else if ( e.getSource() == hourField ) {
132            t.setHour(
133               Integer.parseInt( e.getActionCommand() ) );
134            hourField.setText( "" );
135         }
136         else if ( e.getSource() == minuteField ) {
137            t.setMinute(
138               Integer.parseInt( e.getActionCommand() ) );
139            minuteField.setText( "" );
140         }
141         else if ( e.getSource() == secondField ) {
142            t.setSecond(
143               Integer.parseInt( e.getActionCommand() ) );
144            secondField.setText( "" );
145         }
146
147         updateDisplay();
148      }
149
150      public void updateDisplay()
151      {
152         display.setText( "Hour: " + t.getHour() +
153            "; Minute: " + t.getMinute() +
154            "; Second: " + t.getSecond() );
155         showStatus( "Standard time is: " + t.toString() +
156            "; Military time is: " + t.toMilitaryString() );
157      }
158
159      public void tick()
160      {
161         t.setSecond( ( t.getSecond() + 1 ) % 60 );
162
163         if ( t.getSecond() == 0 ) {
164            t.setMinute( ( t.getMinute() + 1 ) % 60 );
165
166            if ( t.getMinute() == 0 )
167               t.setHour( ( t.getHour() + 1 ) % 24 );
168         }
169      }
170   }
```

Fig. 6.5 Using set and get methods (part 4 of 7).

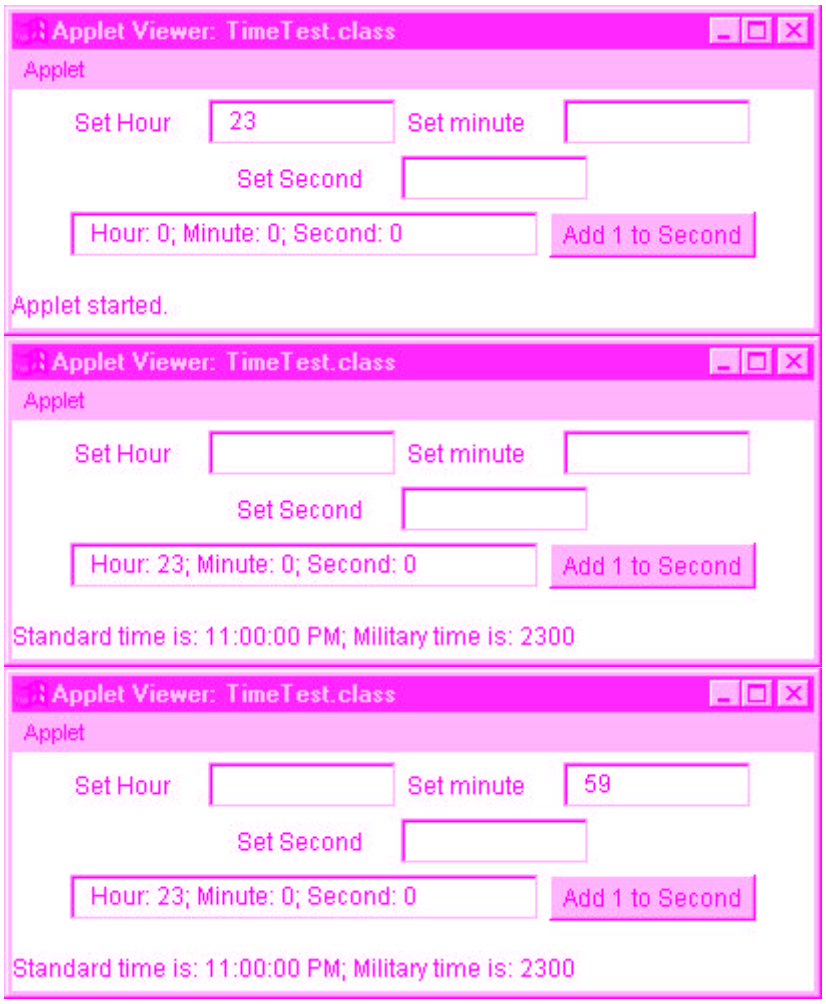
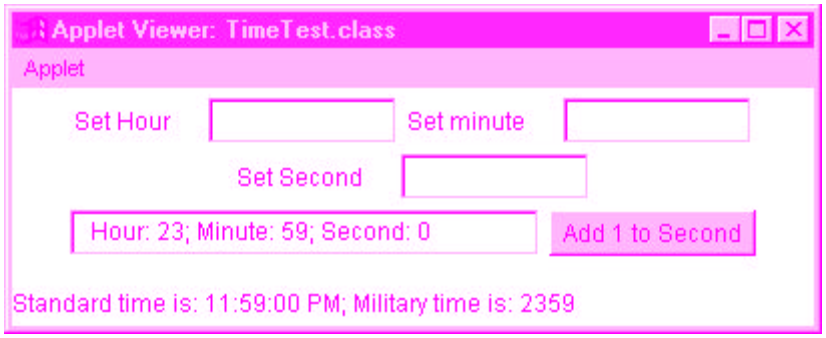


Fig. 6.5 Using set and get methods (part 5 of 7).



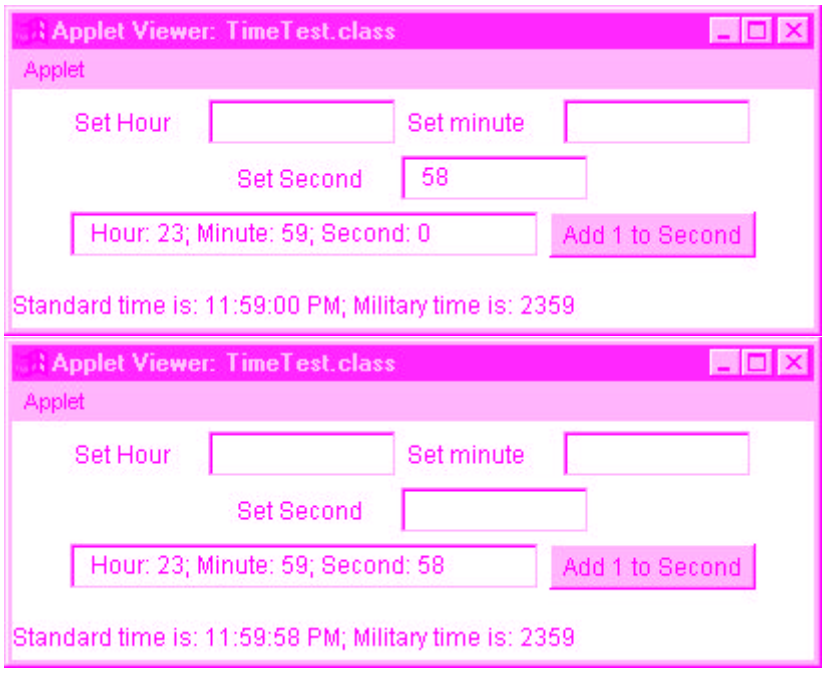


Fig. 6.5 Using set and get methods (part 6 of 7).

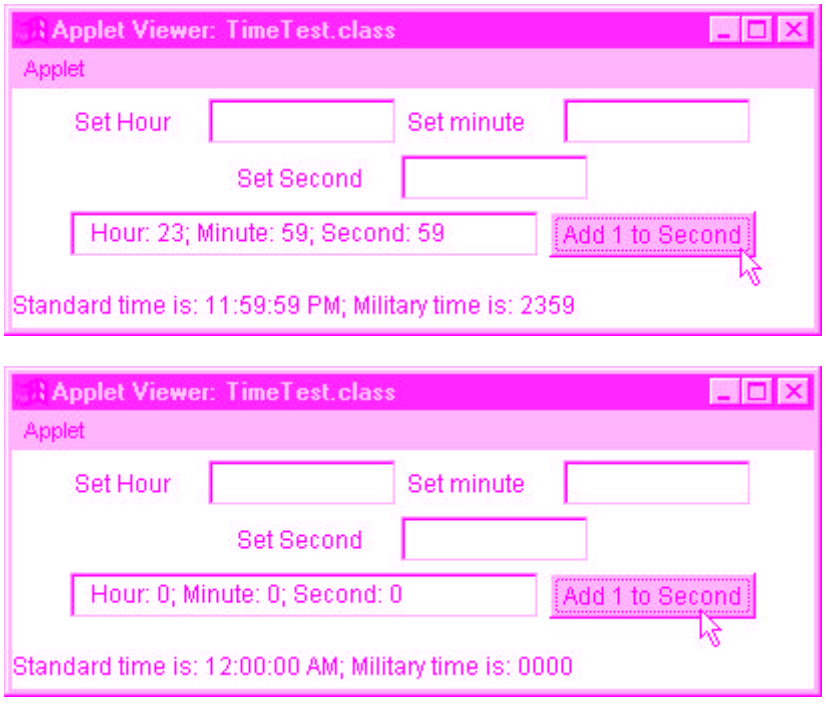


Fig. 6.5 Using set and get methods (part 7 of 7).

```
1  // Fig. 6.6: Increment.java
2  // Initializing a final variable
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.applet.Applet;
6
7  public class Increment extends Applet
8              implements ActionListener {
9     private int count, total;
10    private final int increment = 5; // constant variable
11    private Button incr;
12
13    public void init()
14    {
15       count = 0;
16       total = 0;
17       incr = new Button( "Click to increment" );
18       incr.addActionListener( this );
19       add( incr );
20    }
21
22    public void actionPerformed( ActionEvent e )
23    {
24       total += increment;
25       count++;
26       showStatus( "After increment " + count +
27                   ": total = " + total );
28    }
29 }
30
```

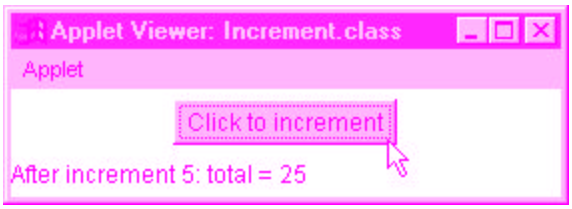


Fig. 6.6 Initializing a **`final`** variable .

```
Increment.java:7: Blank final variable 'increment' may
                  not have been initialized. It must be
                  assigned a value in an initializer, or
                  in every constructor.
public class Increment extends Applet
             ^
1 error
```

Fig. 6.7 Compiler error message as a result of not initializing **`increment`**.

```
1  // Fig. 6.8: Date.java
2  // Declaration of the Date class.
3  package com.deitel.jhtp2.ch06;
4
5  public class Date {
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 function 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    }
```

Fig. 6.8 Demonstrating an object with a member object (part 1 of 5).

```
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
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. 6.8 Demonstrating an object with a member object (part 2 of 5).

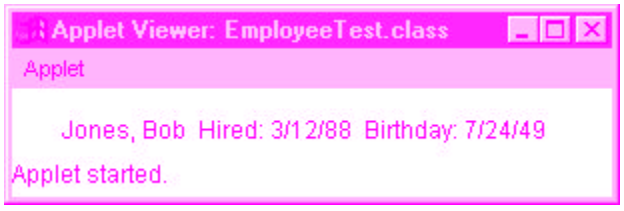
```
57  // Fig. 6.8: Employee.java
58  // Declaration of the Employee class.
59  package com.deitel.jhtp2.ch06;
60
61  public class Employee {
62     private String firstName;
63     private String lastName;
64     private Date birthDate;
65     private Date hireDate;
66
67     public Employee( String 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     }
```

Fig. 6.8 Demonstrating an object with a member object (part 3 of 5).

```
76
77     public String toString()
78     {
79        return lastName + ", " + firstName +
80               "  Hired: " + hireDate.toString() +
81               "  Birthday: " + birthDate.toString();
82     }
83  }
```

Fig. 6.8 Demonstrating an object with a member object (part 4 of 5).

```
84  // Fig. 6.8: EmployeeTest.java
85  // Demonstrating an object with a member object.
86  import java.awt.Graphics;
87  import java.applet.Applet;
88  import com.deitel.jhtp2.ch06.Employee;
89
90  public class EmployeeTest extends Applet {
91     private Employee e;
92
93     public void init()
94     {
95        e = new Employee( "Bob", "Jones", 7, 24, 49,
96                          3, 12, 88 );
97     }
98
99     public void paint( Graphics g )
100    {
101       g.drawString( e.toString(), 25, 25 );
102    }
103 }
```



```
Date object constructor for date 7/24/49
Date object constructor for date 3/12/88
```

Fig. 6.8 Demonstrating an object with a member object (part 5 of 5).

```
1  // Fig. 6.9: 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 java.awt.Graphics;
6  import java.applet.Applet;
7
8  public class PackageDataTest extends Applet {
9     private PackageData d;
10
11    public void init()
12    {
13       d = new PackageData();
14    }
15
```

Fig. 6.9 Package access to members of a class (part 1 of 2).

```
16    public void paint( Graphics g )
17    {
18       g.drawString( "After instantiation: ", 25, 25 );
19       g.drawString( d.toString(), 40, 40 );
20
21       d.x = 77;
22       d.s = "Good bye";
23       g.drawString( "After changing values: ", 25, 55 );
24       g.drawString( d.toString(), 40, 70 );
25    }
26 }
27
28 class PackageData {
29    int x;     // Package access instance variable
30    String s;  // Package access instance variable
31
32    // constructor
33    public PackageData()
34    {
35       x = 0;
36       s = "Hello";
37    }
38
39    public String toString()
40    {
41       return "x: " + x + "    s: " + s;
42    }
43 }
```

Fig. 6.9 Package access to members of a class (part 2 of 2).


```
1  // Fig. 6.10: ThisTest.java
2  // Using the this reference to refer to
3  // instance variables and methods.
4  import java.awt.Graphics;
5  import java.applet.Applet;
6
7  public class ThisTest extends Applet {
8     private int x = 12;
9
10    public void paint( Graphics g )
11    {
12       g.drawString( this.toString(), 25, 25 );
13    }
14
15    public String toString()
16    {
17       return "x = " + x + "   this.x = " + this.x;
18    }
19 }
```

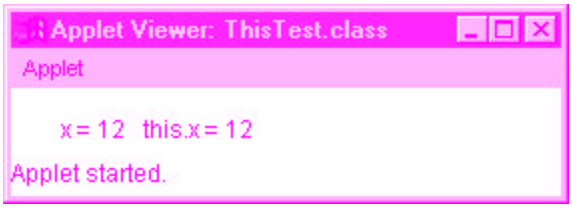


Fig. 6.10 Using the **this** reference.

```
1  // Fig. 6.11: Time4.java
2  // Time4 class definition
3  package com.deitel.jhtp2.ch06;   // place Time4 in a package
4  import java.text.DecimalFormat;  // used for number formatting
5
6  public class Time4 {
7     private int hour;     // 0 - 23
8     private int minute;   // 0 - 59
9     private int second;   // 0 - 59
10
11    // Time4 constructor initializes each instance variable
12    // to zero. Ensures that Time object starts in a
13    // consistent state.
14    public Time4() { setTime( 0, 0, 0 ); }
15
16    // Time4 constructor: hour supplied, minute and second
17    // defaulted to 0.
18    public Time4( int h ) { setTime( h, 0, 0 ); }
```

```
19
20      // Time4 constructor: hour and minute supplied, second
21      // defaulted to 0.
22      public Time4( int h, int m ) { setTime( h, m, 0 ); }
23
24      // Time4 constructor: hour, minute and second supplied.
25      public Time4( int h, int m, int s ) { setTime( h, m, s ); }
26
27      // Set Methods
28      // Set a new Time value using military time. Perform
29      // validity checks on the data. Set invalid values
30      // to zero.
31      public Time4 setTime( int h, int m, int s )
32      {
33         setHour( h );    // set the hour
34         setMinute( m );  // set the minute
35         setSecond( s );  // set the second
36
37         return this;     // enables chaining
38      }
39
```

Fig. 6.11 Chaining method calls (part 1 of 5).

```
40      // set the hour
41      public Time4 setHour( int h )
42      {
43         hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
44
45         return this;     // enables chaining
46      }
47
48      // set the minute
49      public Time4 setMinute( int m )
50      {
51         minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
52
53         return this;     // enables chaining
54      }
55
56      // set the second
57      public Time4 setSecond( int s )
58      {
59         second = ( ( s >= 0 && s < 60 ) ? s : 0 );
60
61         return this;     // enables chaining
62      }
63
64      // Get Methods
65      // get the hour
66      public int getHour() { return hour; }
67
68      // get the minute
69      public int getMinute() { return minute; }
70
71      // get the second
72      public int getSecond() { return second; }
73
74      // Convert time to String in military-time format
75      public String toMilitaryString()
76      {
77         DecimalFormat twoDigits = new DecimalFormat( "00" );
78
79         return twoDigits.format( hour ) +
```

```
80                twoDigits.format( minute );
81     }
82
```

Fig. 6.11 Chaining method calls (part 2 of 5).

```
83     // Convert time to String in standard-time format
84     public String toString()
85     {
86        DecimalFormat twoDigits = new DecimalFormat( "00" );
87
88        return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
89               ":" + twoDigits.format( minute ) +
90               ":" + twoDigits.format( second ) +
91               ( hour < 12 ? " AM" : " PM" );
92     }
93  }
```

Fig. 6.11 Chaining method calls (part 3 of 5).

```
94  // Fig. 6.11: TimeTest.java
95  // Chaining method calls together with the this reference
96  import java.awt.Graphics;
97  import java.applet.Applet;
98  import com.deitel.jhtp2.ch06.Time4;
99
100 public class TimeTest extends Applet {
101    private Time4 t;
102
103    public void init()
104    {
105       t = new Time4();
106    }
107
108    public void paint( Graphics g )
109    {
110       t.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
111       g.drawString( "Military time: " +
112                     t.toMilitaryString(), 25, 25 );
113       g.drawString( "Standard time: " + t.toString(),
114                     25, 40 );
115
116       g.drawString( "New standard time: " +
117          t.setTime( 20, 20, 20 ).toString(), 25, 70 );
118    }
119 }
```

Fig. 6.11 Chaining method calls (part 4 of 5).

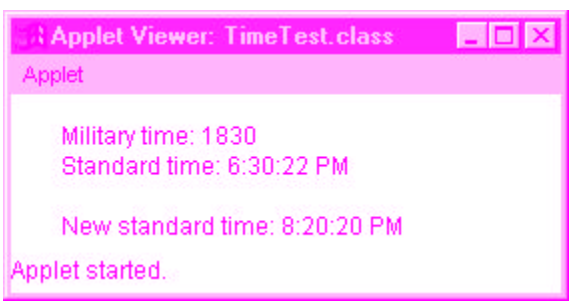


Fig. 6.11 Chaining method calls (part 5 of 5).

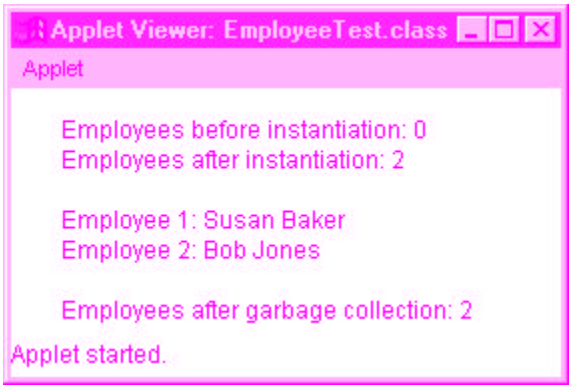
```
1  // Fig. 6.12: Employee.java
2  // Declaration of the Employee class.
3  public class Employee {
4     private String firstName;
5     private String lastName;
6     private static int count;  // # of objects instantiated
7
8     public Employee( String fName, String lName )
9     {
10       firstName = 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    }
24
25    public String getFirstName() { return firstName; }
26
27    public String getLastName() { return lastName; }
28
29    public static int getCount() { return count; }
30 }
```

Fig. 6.12 Using a **`static`** class variable to maintain a count of the number of objects of a class (part 1 of 3).

```
31 // Fig. 6.12: EmployeeTest.java
32 // Test Employee class with static class variable,
33 // static class method, and dynamic memory.
34 import java.awt.Graphics;
35 import java.applet.Applet;
36
37 public class EmployeeTest extends Applet {
38    public void paint( Graphics g )
39    {
40       g.drawString( "Employees before instantiation: " +
41                     Employee.getCount(), 25, 25 );
42       Employee e1 = new Employee( "Susan", "Baker" );
43       Employee e2 = new Employee( "Bob", "Jones" );
44
45       g.drawString( "Employees after instantiation: " +
46                     e1.getCount(), 25, 40 );
47
48       g.drawString( "Employee 1: " + e1.getFirstName() +
49                     " " + e1.getLastName(), 25, 70 );
50       g.drawString( "Employee 2: " + e2.getFirstName() +
51                     " " + e2.getLastName(), 25, 85 );
52
53       // mark objects referred to by e1 and e2
54       // for garbage collection
55       e1 = null;
56       e2 = null;
57
58       System.gc();  // explicit call to garbage collector
59
60       g.drawString( "Employees after garbage collection: " +
```

```
61                      Employee.getCount(), 25, 115 );
62      }
63   }
```

Fig. 6.12 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
Employee object finalizer: Bob Jones
```

Fig. 6.12 Using a **`static`** class variable to maintain a count of the number of objects of a class (part 3 of 3).