Chapter 8 – Object-Based Programming

- Object Oriented Programming (OOP)
  - *Encapsulates* data (attributes) and methods (behaviors)
    - Objects
  - Allows objects to communicate
    - Well-defined *interfaces*
- Procedural programming language
  - C is an example
  - Action-oriented
  - Functions are units of programming
- Object-oriented programming language
  - Java is an Object-oriented example
  - Classes are units of programming
    - Functions, or *methods*, are encapsulated in classes


```
2    // Time1 class definition maintains the time in 24-hour format.
3
4    // Java core packages
5    import java.text.DecimalFormat;
6
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 universal time. Perform
21      // validity checks on the data. Set invalid values to zero.
22      public void setTime( int h, int m, int s )
23      {
24         hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
25         minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
26         second = ( ( s >= 0 && s < 60 ) ? s : 0 );
27      }
28
29      // convert to String in universal-time format
30      public String toUniversalString()
31      {
32         DecimalFormat twoDigits = new DecimalFormat( "00" );
33
```

Outline

`Time1.java`

Line 7
`Time1` (subclass) inherits superclass **`java.lang.Object`**

Lines 8-10
`private` variables

Lines 15, 22 and 30
`public` methods

Lines 15-18
`Time1` constructor then invokes method **`setTime`**

Lines 22-27
Method **`setTime`** sets **`private`** variables according to arguments

```
34          return twoDigits.format( hour ) + ":" +
35             twoDigits.format( minute ) + ":" +
36             twoDigits.format( second );
37       }
38
39       // convert to String in standard-time format
40       public String toString()
41       {
42          DecimalFormat twoDigits = new DecimalFormat( "00" );
43
44          return ( (hour == 12 || hour == 0) ? 12 : hour % 12 ) +
45             ":" + twoDigits.format( minute ) +
46             ":" + twoDigits.format( second ) +
47             ( hour < 12 ? " AM" : " PM" );
48       }
49
50    }  // end class Time1
```

Outline

`Time1.java`

Line 40
Method **`toString`** is a method from **`java.lang.Object`**

Line 40
`Time1` *overrides* this method to provide more appropriate method

Implementing a Abstract Data Type with a Class

- Every Java class must extend another class
  - **`Time1`** extends **`java.lang.Object`**
  - If class does not explicitly **`extend`** another class
    - class implicitly extends **`Object`**
- Class *constructor*
  - *Same name as class*
  - Initializes instance variables of a class object
  - Called when program instantiates an object of that class
  - Can take arguments, but *cannot return data types*
  - Class can have several constructors, through *overloading*
  - Class **`Time1`** (lines 15-18)

8.2 Implementing a Abstract Data Type with a Class

- Overriding methods
  - Method **`Object.toString`**
    - Gives **`String`** representation of object
    - We want **`String`** in standard-time format
  - **`Time1`** *overrides* method **`toString`** (lines 40-48)
    - Overriding method gives **`String`** in standard-time format
    - If we remove lines 40-48
      - **`Time1`** still compiles correctly
      - Uses method **`java.lang.Object.toString`**
      - But gives **`String`** representation of object

5

```
2  // Class TimeTest1 to exercise class Time1
3
4  // Java extension packages
5  import javax.swing.JOptionPane;
6
7  public class TimeTest1 {
8
9     // create Time1 object and manipulate it
10    public static void main( String args[] )
11    {
12       Time1 time = new Time1();  // calls Time1 constructor
13
14       // append String version of time to String output
15       String output = "The initial universal time is: " +
16          time.toUniversalString() +
17          "\nThe initial standard time is: " +  time.toString() +
18          "\nImplicit toString() call: " + time;
19
20       // change time and append String version of time to output
21       time.setTime( 13, 27, 6 );
22       output += "\n\nUniversal time after setTime is: " +
23          time.toUniversalString() +
24          "\nStandard time after setTime is: " + time.toString();
25
26       // use invalid values to change time and append String
27       // version of time to output
28       time.setTime( 99, 99, 99 );
29       output += "\n\nAfter attempting invalid settings: " +
30          "\nUniversal time: " + time.toUniversalString() +
31          "\nStandard time: " + time.toString();
32
33       JOptionPane.showMessageDialog( null, output,
34          "Testing Class Time1",
35          JOptionPane.INFORMATION_MESSAGE );
```

Outline

`TimeTest1.java`

Line 12
Declare and initialize instance of class **`Time1`** by calling **`Time1`** constructor

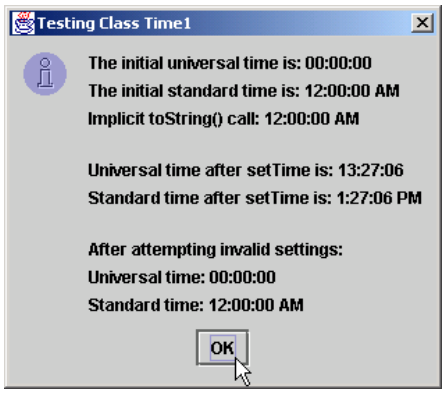
Lines 16-31
`TimeTest` interacts with **`Time1`** by calling **`Time1`** **`public`** methods

6

```
36
37          System.exit( 0 );
38       }
39
40    }  // end class TimeTest1
```

Outline

TimeTest1.java



7

Implementing a Abstract Data Type with a Class

- We introduce classes **`Time1`** and **`TimeTest`**
  - **`Time1.java`** defines class **`Time1`**
  - **`TimeTest.java`** defines class **`TimeTest`**
  - **`public`** classes must be defined in separate files
  - Class **`Time1`** will not execute by itself
    - Does not have method **`main`**
    - Does not extend **`JApplet`**
    - **`TimeTest`**, which has method **`main`**, creates (*instantiates*) and uses **`Time1`** object

8

Class Scope

- Class scope
  - Class instance variables and methods
    Ex. Ch08_time1 tt;
  - Members are accessible to all class methods
    Ex. Hour in Ch08_time1 object
  - Members can be referenced by name
    Ex. tt.setTime(hh,mm,ss);
- Block scope
  - Variables defined in methods (Known only to that method)
- Member access modifiers
  - Control access to class' instance variables and methods
  - **`Public`** (Variables and methods are accessible to class clients)
  - **`private`**
    - Variables and methods are not accessible to class clients
    - Accessor method ("get" method) → Allow clients to read **`private`** data
    - Mutator method ("set" method) → Allow clients to modify **`private`** data

9

```
1   // Fig. 8.3: TimeTest2.java
2   // Demonstrate errors resulting from attempts
3   // to access private members of class Time1.
4   public class TimeTest2 {
5
6      public static void main( String args[] )
7      {
8         Time1 time = new Time1();
9
10        time.hour = 7;
11     }
12  }
```

Outline

`TimeTest2.java`

Line 10
Compiler error – **`TimeTest2`** cannot directly access **`Time1`**'s **`private`** data

```
TimeTest2.java:10: hour has private access in Time1
      time.hour = 7;
           ^
1 error
```

Fig. 8.3 Erroneous attempt to access private members of class `Time1`.

10

Creating Packages

- We can **`import`** *packages* into programs
  - Group of related classes and interfaces
  - Help manage complexity of application components
  - Facilitate software reuse
  - Provide convention for unique class names
    - Popular package-naming convention
      - Reverse Internet domain name
        - e.g., **`com.deitel`**
  - **How to build package**
    1. **Add package statement in xxx.java**
    2. **Use javac –d . xxx.java to compile**
    3. **Add import ….xxxx; in the used java program**

11

```
2  // Time1 class definition in a package
3  package com.deitel.jhtp4.ch08;
4
5  // Java core packages
6  import java.text.DecimalFormat;
7
8  public class Time1 extends Object {
9     private int hour;     // 0 - 23
10    private int minute;   // 0 - 59
11    private int second;   // 0 - 59
12
13    // Time1 constructor initializes each instance variable
14    // to zero. Ensures that each Time1 object starts in a
15    // consistent state.
16    public Time1()
17    {
18       setTime( 0, 0, 0 );
19    }
20
21    // Set a new time value using universal time. Perform
22    // validity checks on the data. Set invalid values 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
```

Outline

`Time1.java`

Line 3
Class **`Time1`** is placed in this **`package`**

Line 3
Class **`Time1`** is in directory **`com/deitel/jhtp4/ch08`**

Line 6
`import` class **`DecimalFormat`** from **`package java.text`**

Line 33
`DecimalFormat` from **`package java.text`**

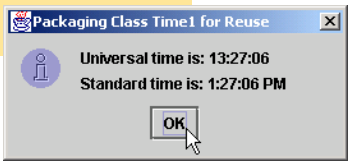
12

```
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
51    }  // end class Time1
```

Outline

Time1.java


```
1    // Fig. 8.5: TimeTest3.java
2    // Class TimeTest3 to use imported class Time1
3
4    // Java extension packages
5    import javax.swing.JOptionPane;
6
7    // Deitel packages
8    import com.deitel.jhtp4.ch08.Time1;  // import Time1 class
9
10   public class TimeTest3 {
11
12      // create an object of class Time1 and manipulate it
13      public static void main( String args[] )
14      {
15         Time1 time = new Time1();    // create Time1 object
16
17         time.setTime( 13, 27, 06 );  // set new time
18         String output =
19            "Universal time is: " + time.toUniversalString() +
20            "\nStandard time is: " + time.toString();
21
22         JOptionPane.showMessageDialog( null, output,
23            "Packaging Class Time1 for Reuse",
24            JOptionPane.INFORMATION_MESSAGE );
25
26         System.exit( 0 );
27      }
28
29   }  // end class TimeTest3
```



Outline

TimeTest3.java

Line 5
import class **JOptionPane** from **package javax.swing**

Line 8
import class **Time1** from **package com.deitel.jhtp4.ch08**

Line 15
TimeTest3 can declare **Time1** object

Initializing Class Objects: Constructors

- Class constructor
  - Same name as class
  - Initializes instance variables of a class object
  - Call class constructor to instantiate object of that class

    `ref = new` *ClassName*`(` *arguments* `);`

    - **`ref`** is reference to appropriate data type
    - **`new`** indicates that **new object is created**
    - *ClassName* indicates type of object created
    - *arguments* specifies constructor argument values
- Overloaded constructors
  - Methods (in same class) may have same name
  - Must have different parameter lists

15

```
2  // Time2 class definition with overloaded constructors.
3  package com.deitel.jhtp4.ch08;
4
5  // Java core packages
6  import java.text.DecimalFormat;
7
8  public class Time2 extends Object {
9     private int hour;     // 0 - 23
10    private int minute;   // 0 - 59
11    private int second;   // 0 - 59
12
13    // Time2 constructor initializes each instance variable
14    // to zero. Ensures that Time object starts in a
15    // consistent state.
16    public Time2()
17    {
18       setTime( 0, 0, 0 );
19    }
20
21    // Time2 constructor: hour supplied, minute and second
22    // defaulted to 0
23    public Time2( int h )
24    {
25       setTime( h, 0, 0 );
26    }
27
28    // Time2 constructor: hour and minute supplied, second
29    // defaulted to 0
30    public Time2( int h, int m )
31    {
32       setTime( h, m, 0 );
33    }
34
```

Outline

`Time2.java`

Lines 16-19
Default constructor has no arguments

Lines 23-26
Overloaded constructor has one **`int`** argument

Lines 30-33
Second overloaded constructor has two **`int`** arguments

16

```
35      // Time2 constructor: hour, minute and second supplied
36      public Time2( int h, int m, int s )
37      {
38         setTime( h, m, s );
39      }
40
41      // Time2 constructor: another Time2 object supplied
42      public Time2( Time2 time )
43      {
44         setTime( time.hour, time.minute, time.second );
45      }
46
47      // Set a new time value using universal time. Perform
48      // validity checks on data. Set invalid values to zero.
49      public void setTime( int h, int m, int s )
50      {
51         hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
52         minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
53         second = ( ( s >= 0 && s < 60 ) ? s : 0 );
54      }
55
56      // convert to String in universal-time format
57      public String toUniversalString()
58      {
59         DecimalFormat twoDigits = new DecimalFormat( "00" );
60
61         return twoDigits.format( hour ) + ":" +
62            twoDigits.format( minute ) + ":" +
63            twoDigits.format( second );
64      }
65
66      // convert to String in standard-time format
67      public String toString()
68      {
69         DecimalFormat twoDigits = new DecimalFormat( "00" );
```

Outline

Time2.java

Lines 36-39
Third overloaded constructor has three **int** arguments

Lines 42-45
Fourth overloaded constructor has **Time2** argument

17

```
70
71         return ( (hour == 12 || hour == 0) ? 12 : hour % 12 ) +
72            ":" + twoDigits.format( minute ) +
73            ":" + twoDigits.format( second ) +
74            ( hour < 12 ? " AM" : " PM" );
75      }
76
77   }  // end class Time2
```

Outline

Time2.java

18

```
2  // Using overloaded constructors
3
4  // Java extension packages
5  import javax.swing.*;
6
7  // Deitel packages
8  import com.deitel.jhtp4.ch08.Time2;
9
10 public class TimeTest4 {
11
12    // test constructors of class Time2
13    public static void main( String args[] )
14    {
15       Time2 t1, t2, t3, t4, t5, t6;
16
17       t1 = new Time2();               // 00:00:00
18       t2 = new Time2( 2 );            // 02:00:00
19       t3 = new Time2( 21, 34 );       // 21:34:00
20       t4 = new Time2( 12, 25, 42 );   // 12:25:42
21       t5 = new Time2( 27, 74, 99 );   // 00:00:00
22       t6 = new Time2( t4 );           // 12:25:42
23
24       String output = "Constructed with: " +
25          "\nt1: all arguments defaulted" +
26          "\n      " + t1.toUniversalString() +
27          "\n      " + t1.toString();
28
29       output += "\nt2: hour specified; minute and " +
30          "second defaulted" +
31          "\n      " + t2.toUniversalString() +
32          "\n      " + t2.toString();
33
```

Outline

TimeTest4.java

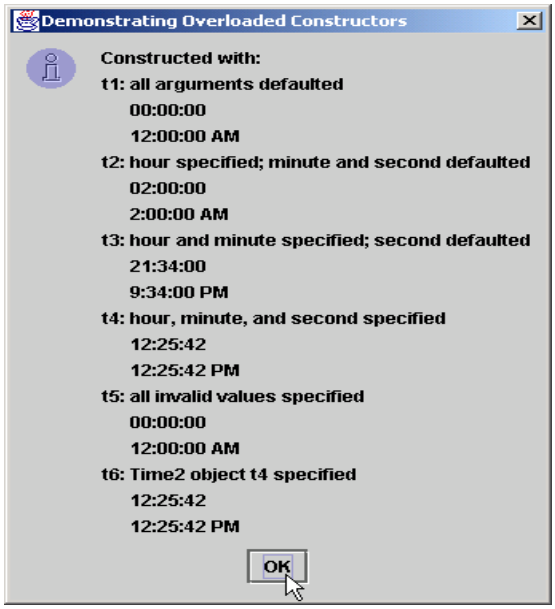
Line 15
Declare six references to **Time2** objects

Lines 17-22
Instantiate each **Time2** reference using a different constructor


```
34       output += "\nt3: hour and minute specified; " +
35          "second defaulted" +
36          "\n      " + t3.toUniversalString() +
37          "\n      " + t3.toString();
38
39       output += "\nt4: hour, minute, and second specified" +
40          "\n      " + t4.toUniversalString() +
41          "\n      " + t4.toString();
42
43       output += "\nt5: all invalid values specified" +
44          "\n      " + t5.toUniversalString() +
45          "\n      " + t5.toString();
46
47       output += "\nt6: Time2 object t4 specified" +
48          "\n      " + t6.toUniversalString() +
49          "\n      " + t6.toString();
50
51       JOptionPane.showMessageDialog( null, output,
52          "Demonstrating Overloaded Constructors",
53          JOptionPane.INFORMATION_MESSAGE );
54
55       System.exit( 0 );
56    }
57
58 }  // end class TimeTest4
```

Outline

TimeTest4.java



Outline

`TimeTest4.java`

Different outputs, because each **`Time2`** reference was instantiated with a different constructor

21

Using Set and Get Methods

- Accessor method ("get" method)
  - **`public`** method
  - Allow clients to read **`private`** data
- Mutator method ("set" method)
  - **`public`** method
  - Allow clients to modify **`private`** data

22

```
2  // Time3 class definition with set and get methods
3  package com.deitel.jhtp4.ch08;
4
5  // Java core packages
6  import java.text.DecimalFormat;
7
8  public class Time3 extends Object {
9     private int hour;     // 0 - 23
10    private int minute;   // 0 - 59
11    private int second;   // 0 - 59
12
13    // Time3 constructor initializes each instance variable
14    // to zero. Ensures that Time object starts in a
15    // consistent state.
16    public Time3()
17    {
18       setTime( 0, 0, 0 );
19    }
20
21    // Time3 constructor: hour supplied, minute and second
22    // defaulted to 0
23    public Time3( int h )
24    {
25       setTime( h, 0, 0 );
26    }
27
28    // Time3 constructor: hour and minute supplied, second
29    // defaulted to 0
30    public Time3( int h, int m )
31    {
32       setTime( h, m, 0 );
33    }
34
```

Outline

Time3.java

Lines 9-11
private variables cannot be accessed directly by objects in different classes

23

```
35    // Time3 constructor: hour, minute and second supplied
36    public Time3( int h, int m, int s )
37    {
38       setTime( h, m, s );
39    }
40
41    // Time3 constructor: another Time3 object supplied
42    public Time3( Time3 time )
43    {
44       setTime( time.getHour(), time.getMinute(),
45          time.getSecond() );
46    }
47
48    // Set Methods
49    // Set a new time value using universal time. Perform
50    // validity checks on data. Set invalid values to zero.
51    public void setTime( int h, int m, int s )
52    {
53       setHour( h );    // set the hour
54       setMinute( m );  // set the minute
55       setSecond( s );  // set the second
56    }
57
58    // validate and set hour
59    public void setHour( int h )
60    {
61       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
62    }
63
64    // validate and set minute
65    public void setMinute( int m )
66    {
67       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
68    }
69
```

Outline

Time3.java

Lines 51-68
Mutator methods allows objects to manipulate **private** variables

24

```
70      // validate and set second
71      public void setSecond( int s )
72      {
73         second = ( ( s >= 0 && s < 60 ) ? s : 0 );
74      }
75
76      // Get Methods
77      // get hour value
78      public int getHour()
79      {
80         return hour;
81      }
82
83      // get minute value
84      public int getMinute()
85      {
86         return minute;
87      }
88
89      // get second value
90      public int getSecond()
91      {
92         return second;
93      }
94
95      // convert to String in universal-time format
96      public String toUniversalString()
97      {
98         DecimalFormat twoDigits = new DecimalFormat( "00" );
99
100        return twoDigits.format( getHour() ) + ":" +
101           twoDigits.format( getMinute() ) + ":" +
102           twoDigits.format( getSecond() );
103     }
104
```

Outline

Time3.java

Lines 78-93
Accessor methods allows objects to read **private** variables

25

```
105     // convert to String in standard-time format
106     public String toString()
107     {
108        DecimalFormat twoDigits = new DecimalFormat( "00" );
109
110        return ( ( getHour() == 12 || getHour() == 0 ) ?
111           12 : getHour() % 12 ) + ":" +
112           twoDigits.format( getMinute() ) + ":" +
113           twoDigits.format( getSecond() ) +
114           ( getHour() < 12 ? " AM" : " PM" );
115     }
116
117  }  // end class Time3
```

Outline

Time3.java

26

```
2    // Demonstrating the Time3 class set and get methods.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   // Deitel packages
12   import com.deitel.jhtp4.ch08.Time3;
13
14   public class TimeTest5 extends JApplet
15      implements ActionListener {
16
17      private Time3 time;
18      private JLabel hourLabel, minuteLabel, secondLabel;
19      private JTextField hourField, minuteField,
20         secondField, displayField;
21      private JButton tickButton;
22
23      // Create Time3 object and set up GUI
24      public void init()
25      {
26         time = new Time3();
27
28         Container container = getContentPane();
29         container.setLayout( new FlowLayout() );
30
31         // set up hourLabel and hourField
32         hourLabel = new JLabel( "Set Hour" );
33         hourField = new JTextField( 10 );
34         hourField.addActionListener( this );
```

Outline

`TimeTest5.java`

Lines 17 and 26
Declare and instantiate **`Time3`** object

27

```
35         container.add( hourLabel );
36         container.add( hourField );
37
38         // set up minuteLabel and minuteField
39         minuteLabel = new JLabel( "Set minute" );
40         minuteField = new JTextField( 10 );
41         minuteField.addActionListener( this );
42         container.add( minuteLabel );
43         container.add( minuteField );
44
45         // set up secondLabel and secondField
46         secondLabel = new JLabel( "Set Second" );
47         secondField = new JTextField( 10 );
48         secondField.addActionListener( this );
49         container.add( secondLabel );
50         container.add( secondField );
51
52         // set up displayField
53         displayField = new JTextField( 30 );
54         displayField.setEditable( false );
55         container.add( displayField );
56
57         // set up tickButton
58         tickButton = new JButton( "Add 1 to Second" );
59         tickButton.addActionListener( this );
60         container.add( tickButton );
61
62         updateDisplay();  // update text in displayField
63      }
64
65      // handle button and text field events
66      public void actionPerformed( ActionEvent actionEvent )
67      {
```

Outline

`TimeTest5.java`

Lines 39-55
`JTextField`s allow user to specify time

28

```
68          // process tickButton event
69          if ( actionEvent.getSource() == tickButton )
70             tick();
71
72          // process hourField event
73          else if ( actionEvent.getSource() == hourField ) {
74             time.setHour(
75                Integer.parseInt( actionEvent.getActionCommand() ) );
76             hourField.setText( "" );
77          }
78
79          // process minuteField event
80          else if ( actionEvent.getSource() == minuteField ) {
81             time.setMinute(
82                Integer.parseInt( actionEvent.getActionCommand() ) );
83             minuteField.setText( "" );
84          }
85
86          // process secondField event
87          else if ( actionEvent.getSource() == secondField ) {
88             time.setSecond(
89                Integer.parseInt( actionEvent.getActionCommand() ) );
90             secondField.setText( "" );
91          }
92
93          updateDisplay();  // update displayField and status bar
94       }
95
96       // update displayField and applet container's status bar
97       public void updateDisplay()
98       {
99          displayField.setText( "Hour: " + time.getHour() +
100            "; Minute: " + time.getMinute() +
101            "; Second: " + time.getSecond() );
102
```

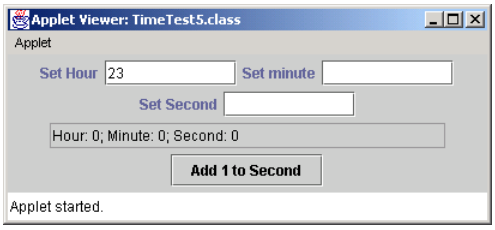
Outline

`TimeTest5.java`

Lines 74-76
Lines 81-83
Lines 88-90
`TimeTest5` uses **`Time3`** mutator methods to set **`Time3`** **`private`** variables

29

```
103         showStatus( "Standard time is: " + time.toString() +
104            "; Universal time is: " + time.toUniversalString() );
105      }
106
107      // add one to second and update hour/minute if necessary
108      public void tick()
109      {
110         time.setSecond( ( time.getSecond() + 1 ) % 60 );
111
112         if ( time.getSecond() == 0 ) {
113            time.setMinute( ( time.getMinute() + 1 ) % 60 );
114
115            if ( time.getMinute() == 0 )
116               time.setHour( ( time.getHour() + 1 ) % 24 );
117         }
118      }
119
120   }  // end class TimeTest5
```

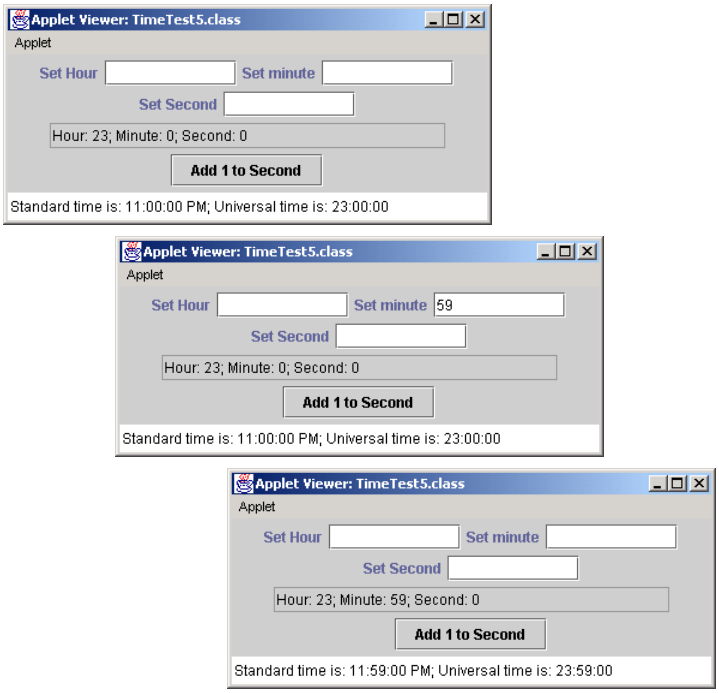


Outline

`TimeTest5.java`

Lines 110-116
`TimeTest5` uses **`Time3`** accessor methods to read **`Time3`** **`private`** variables

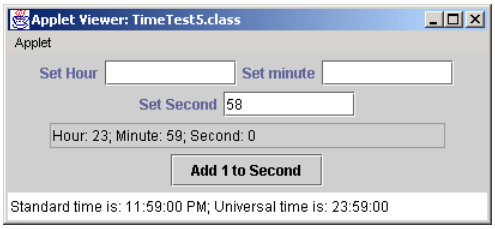
30

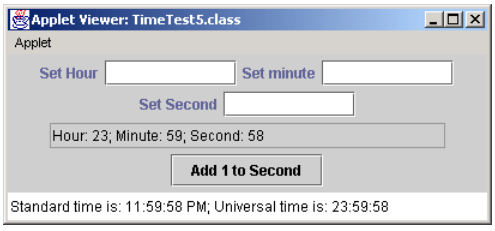


Outline

`TimeTest5.java`

31





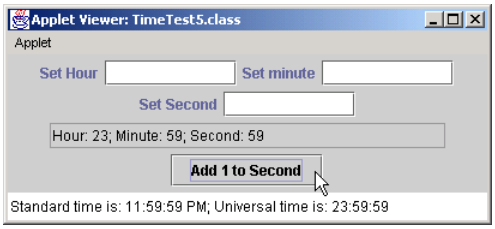
Outline

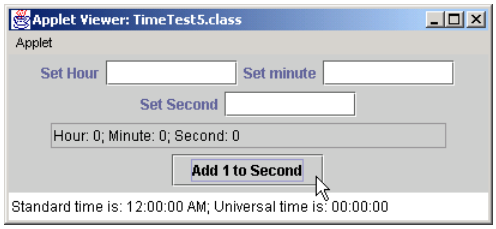
`TimeTest5.java`

32

Outline

`TimeTest5.java`





33

Uses Programmer-Defined Packages

- **`jar`**
  - Java archive utility
  - Bundles classes and packages
    - Classes and packages are downloaded as one unit (JAR file)
      - Client makes only one call to server
    - Create **Jar** file

      Ex. Jar cf name.jar xxxx.class object-hierarchy
    - See **Jar** file

      Ex. Jar tvf name.jar

34

Outline

TimeTest5.jar

```
   0 Fri May 25 14:13:14 EDT 2001 META-INF/
  71 Fri May 25 14:13:14 EDT 2001 META-INF/MANIFEST.MF
2959 Fri May 25 13:42:32 EDT 2001 TimeTest5.class
   0 Fri May 18 17:35:18 EDT 2001 com/deitel/
   0 Fri May 18 17:35:18 EDT 2001 com/deitel/jhtp4/
   0 Fri May 18 17:35:18 EDT 2001 com/deitel/jhtp4/ch08/
1765 Fri May 18 17:35:18 EDT 2001 com/deitel/jhtp4/ch08/Time3.class
```

Fig. 8.10 Contents of TimeTest5.jar.

35

Software Reusability

- Java
  - Framework for achieving software reusability
  - Rapid applications development (RAD)
    - e.g., creating a GUI application quickly
- **`final`** keyword
  - Indicates that variable is not modifiable
    - Any attempt to modify **`final`** variable results in error

```
private final int INCREMENT = 5;
```

    - Declares variable **`INCREMENT`** as a *constant*
  - Enforces *principle of least privilege*

36

```
2   // Initializing a final variable
3
4   // Java core packages
5   import java.awt.*;
6   import java.awt.event.*;
7
8   // Java extension packages
9   import javax.swing.*;
10
11  public class Increment extends JApplet
12     implements ActionListener {
13
14     private int count = 0, total = 0;
15     private final int INCREMENT = 5;   // constant variable
16
17     private JButton button;
18
19     // set up GUI
20     public void init()
21     {
22        Container container = getContentPane();
23
24        button = new JButton( "Click to increment" );
25        button.addActionListener( this );
26        container.add( button );
27     }
28
29     // add INCREMENT to total when user clicks button
30     public void actionPerformed( ActionEvent actionEvent )
31     {
32        total += INCREMENT;
33        count++;
34        showStatus( "After increment " + count +
35           ": total = " + total );
```

Outline

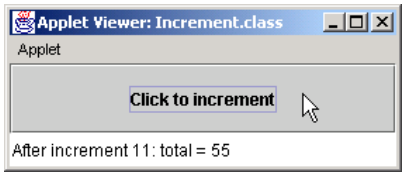
Increment.java

Line 15
final keyword declares **INCREMENT** as constant

Line 32
final variable **INCREMENT** must be initialized before using it

37

```
36     }
37
38  }  // end class Increment
```




```
Increment.java:11: variable INCREMENT might not have been initialized
public class Increment extends Japplet
       ^
1 error
```

Compiler error message as a result of not initializing increment.

38

Composition: Objects as Instance Variables of Other Classes

- Composition
  - Class contains references to objects of other classes
    - These references are members

39

```
2  // Declaration of the Date class.
3  package com.deitel.jhtp4.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 theMonth, int theDay, int theYear )
14    {
15       if ( theMonth > 0 && theMonth <= 12 )  // validate month
16          month = theMonth;
17       else {
18          month = 1;
19          System.out.println( "Month " + theMonth +
20             " invalid. Set to month 1." );
21       }
22
23       year = theYear;                // could validate year
24       day = checkDay( theDay );      // validate 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[] =
35          { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

Outline

Date.java

Line 5
Class **Date** encapsulates data that describes date

Lines 13-28
Date constructor instantiates **Date** object based on specified arguments

40

```
36
37          // check if day in range for month
38          if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
39             return testDay;
40
41          // check for leap year
42          if ( month == 2 && 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       {
56          return month + "/" + day + "/" + year;
57       }
58
59    }  // end class Date
```

Outline

Date.java


```
2    // Definition of class Employee.
3    package com.deitel.jhtp4.ch08;
4
5    public class Employee extends Object {
6       private String firstName;
7       private String lastName;
8       private Date birthDate;
9       private Date hireDate;
10
11      // constructor to initialize name, birth date and hire date
12      public Employee( String first, String last,
13        int birthMonth, int birthDay, int birthYear,
14        int hireMonth, int hireDay, int hireYear )
15      {
16         firstName = first;
17         lastName = last;
18         birthDate = new Date( birthMonth, birthDay, birthYear );
19         hireDate = new Date( hireMonth, hireDay, hireYear );
20      }
21
22      // convert Employee to String format
23      public String toString()
24      {
25         return lastName + ", " + firstName +
26            "  Hired: " + hireDate.toString() +
27            "  Birthday: " + birthDate.toString();
28      }
29
30   }  // end class Employee
```

Outline

Employee.java

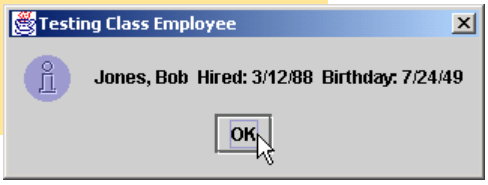
Lines 9-10
Employee is composed of two references to **Date** objects

Lines 18-19
instantiate **Date** objects by calling constructors

Outline

EmployeeTest.java

```
2  // Demonstrating an object with a member object.
3
4  // Java extension packages
5  import javax.swing.JOptionPane;
6
7  // Deitel packages
8  import com.deitel.jhtp4.ch08.Employee;
9
10 public class EmployeeTest {
11
12    // test class Employee
13    public static void main( String args[] )
14    {
15       Employee employee = new Employee( "Bob", "Jones",
16          7, 24, 49, 3, 12, 88 );
17
18       JOptionPane.showMessageDialog( null,
19          employee.toString(), "Testing Class Employee",
20          JOptionPane.INFORMATION_MESSAGE );
21
22       System.exit( 0 );
23    }
24
25 }  // end class EmployeeTest
```




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

43

Package Access

- Package access
  - Variable or method does not have member access modifier

44

```
2  // Classes in the same package (i.e., the same directory) can
3  // use package access data of other classes in the same package.
4
5  // Java extension packages
6  import javax.swing.JOptionPane;
7
8  public class PackageDataTest {
9
10    // Java extension packages
11    public static void main( String args[] )
12    {
13       PackageData packageData = new PackageData();
14
15       // append String representation of packageData to output
16       String output =
17          "After instantiation:\n" + packageData.toString();
18
19       // change package access data in packageData object
20       packageData.number = 77;
21       packageData.string = "Good bye";
22
23       // append String representation of packageData to output
24       output += "\nAfter changing values:\n" +
25          packageData.toString();
26
27       JOptionPane.showMessageDialog( null, output,
28          "Demonstrating Package Access",
29          JOptionPane.INFORMATION_MESSAGE );
30
31       System.exit( 0 );
32    }
33
34 }  // end class PackageDataTest
35
```

Outline

PackageDataTest.java

Line 13
Instantiate reference to **PackageData** object

Lines 17-25
PackageDataTest can access **PackageData** data, because each class shares same **package**

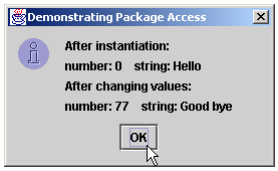
45

```
36 // class with package-access instance variables
37 class PackageData {
38    int number;     // package-access instance variable
39    String string;  // package-access instance variable
40
41    // constructor
42    public PackageData()
43    {
44       number = 0;
45       string = "Hello";
46    }
47
48    // convert PackageData object to String representation
49    public String toString()
50    {
51       return "number: " + number + "    string: " + string;
52    }
53
54 }  // end class PackageData
```

Outline

PackageDataTest.java

Line 37
No access modifier, so class has package-access variables



46

Using the `this` Reference

- This reference
  - Allows an object to refers to itself

47

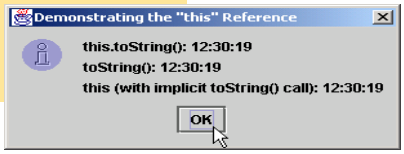
Outline

ThisTest.java

```
2  // Using the this reference to refer to
3  // instance variables and methods.
4
5  // Java core packages
6  import java.text.DecimalFormat;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class ThisTest {
12
13    // test class SimpleTime
14    public static void main( String args[] )
15    {
16       SimpleTime time = new SimpleTime( 12, 30, 19 );
17
18       JOptionPane.showMessageDialog( null, time.buildString(),
19          "Demonstrating the \"this\" Reference",
20          JOptionPane.INFORMATION_MESSAGE );
21
22       System.exit( 0 );
23    }
24
25 }  // end class ThisTest
26
27 // class SimpleTime demonstrates the "this" reference
28 class SimpleTime {
29    private int hour, minute, second;
30
```

48

```
31    // constructor uses parameter names identical to instance
32    // variable names, so "this" reference required to distinguish
33    // between instance variables and parameters
34    public SimpleTime( int hour, int minute, int second )
35    {
36       this.hour = hour;      // set "this" object's hour
37       this.minute = minute;  // set "this" object's minute
38       this.second = second;  // set "this" object's second
39    }
40
41    // call toString explicitly via "this" reference, explicitly
42    // via implicit "this" reference, implicitly via "this"
43    public String buildString()
44    {
45       return "this.toString(): " + this.toString() +
46          "\ntoString(): " + toString() +
47          "\nthis (with implicit toString() call): " + this;
48    }
49
50    // convert SimpleTime to String format
51    public String toString()
52    {
53       DecimalFormat twoDigits = new DecimalFormat( "00" );
54
55       // "this" not required, because toString does not have
56       // local variables with same names as instance variables
57       return twoDigits.format( this.hour ) + ":" +
58          twoDigits.format( this.minute ) + ":" +
59          twoDigits.format( this.second );
60    }
61
62 }  // end class SimpleTime
```



Outline

ThisTest.java

Lines 36-37
this used to distinguish between arguments and **ThisTest** variables

Lines 45, 57-59
Implicit use of **this**

```
2  // Time4 class definition
3  package com.deitel.jhtp4.ch08;
4
5  // Java core packages
6  import java.text.DecimalFormat;
7
8  public class Time4 extends Object {
9     private int hour;     // 0 - 23
10    private int minute;   // 0 - 59
11    private int second;   // 0 - 59
12
13    // Time4 constructor initializes each instance variable
14    // to zero. Ensures that Time object starts in a
15    // consistent state.
16    public Time4()
17    {
18       this.setTime( 0, 0, 0 );
19    }
20
21    // Time4 constructor: hour supplied, minute and second
22    // defaulted to 0
23    public Time4( int hour )
24    {
25       this.setTime( hour, 0, 0 );
26    }
27
28    // Time4 constructor: hour and minute supplied, second
29    // defaulted to 0
30    public Time4( int hour, int minute )
31    {
32       this.setTime( hour, minute, 0 );
33    }
34
```

Outline

Time4.java

```
35    // Time4 constructor: hour, minute and second supplied
36    public Time4( int hour, int minute, int second )
37    {
38       this.setTime( hour, minute, second );
39    }
40
41    // Time4 constructor: another Time4 object supplied.
42    public Time4( Time4 time )
43    {
44       this.setTime( time.getHour(), time.getMinute(),
45          time.getSecond() );
46    }
47
48    // Set Methods
49    // set a new Time value using universal time
50    public Time4 setTime( int hour, int minute, int second )
51    {
52       this.setHour( hour );      // set hour
53       this.setMinute( minute );  // set minute
54       this.setSecond( second );  // set second
55
56       return this;     // enables chaining
57    }
58
59    // validate and set hour
60    public Time4 setHour( int hour )
61    {
62       this.hour = ( hour >= 0 && hour < 24 ? hour : 0 );
63
64       return this;     // enables chaining
65    }
66
```

Outline

`Time4.java`

Lines 56 and 64
Method returns **`this`** reference to enable *chaining* (concatenated method calls)

51

```
67    // validate and set minute
68    public Time4 setMinute( int minute )
69    {
70       this.minute =
71          ( minute >= 0 && minute < 60 ) ? minute : 0;
72
73       return this;     // enables chaining
74    }
75
76    // validate and set second
77    public Time4 setSecond( int second )
78    {
79       this.second =
80          ( second >= 0 && second < 60 ) ? second : 0;
81
82       return this;     // enables chaining
83    }
84
85    // Get Methods
86    // get value of hour
87    public int getHour() { return this.hour; }
88
89    // get value of minute
90    public int getMinute() { return this.minute; }
91
92    // get value of second
93    public int getSecond() { return this.second; }
94
95    // convert to String in universal-time format
96    public String toUniversalString()
97    {
98       DecimalFormat twoDigits = new DecimalFormat( "00" );
99
```

Outline

`Time4.java`

Lines 73 and 82
Method returns **`this`** reference to enable *chaining* (concatenated method calls)

52

```
100       return twoDigits.format( this.getHour() ) + ":" +
101          twoDigits.format( this.getMinute() ) + ":" +
102          twoDigits.format( this.getSecond() );
103    }
104
105    // convert to String in standard-time format
106    public String toString()
107    {
108       DecimalFormat twoDigits = new DecimalFormat( "00" );
109
110       return ( this.getHour() == 12 || this.getHour() == 0 ?
111          12 : this.getHour() % 12 ) + ":" +
112          twoDigits.format( this.getMinute() ) + ":" +
113          twoDigits.format( this.getSecond() ) +
114          ( this.getHour() < 12 ? " AM" : " PM" );
115    }
116
117 }  // end class Time4
```

Outline

Time4.java

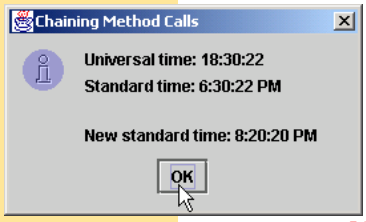

```
2  // Chaining method calls together with the this reference
3
4  // Java extension packages
5  import javax.swing.*;
6
7  // Deitel packages
8  import com.deitel.jhtp4.ch08.Time4;
9
10 public class TimeTest6 {
11
12    // test method call chaining with object of class Time4
13    public static void main( String args[] )
14    {
15       Time4 time = new Time4();
16
17       // chain calls to setHour, setMinute and setSecond
18       time.setHour( 18 ).setMinute( 30 ).setSecond( 22 );
19
20       // use method call chaining to set new time and get
21       // String representation of new time
22       String output =
23          "Universal time: " + time.toUniversalString() +
24          "\nStandard time: " + time.toString() +
25          "\n\nNew standard time: " +
26          time.setTime( 20, 20, 20 ).toString();
27
28       JOptionPane.showMessageDialog( null, output,
29          "Chaining Method Calls",
30          JOptionPane.INFORMATION_MESSAGE );
31
32       System.exit( 0 );
33    }
34
35 }  // end class TimeTest6
```

Outline

Time6.java

Line 15
Declare and instantiate reference to **Time4** object

Lines 18 and 26
Concatenate (chain) **Time4** method calls, because methods return **Time4** reference



Finalizers

- Garbage collection
  - Returns memory to system
  - Java performs this automatically
    - object marked for garbage collection if no references to object
- Finalizer method
  - Returns resources to system
  - Java provides method **`finalize`**
    - Defined in **`java.lang.Object`**
    - Receives no parameters
    - Returns **`void`**
- **`static`** keyword
  - **`static`** class variable
    - Class-wide information
      - All class objects share same data

55

```
2  // Employee class definition.
3  public class Employee extends Object {
4     private String firstName;
5     private String lastName;
6     private static int count;  // number of objects in memory
7
8     // initialize employee, add 1 to static count and
9     // output String indicating that constructor was called
10    public Employee( String first, String last )
11    {
12       firstName = first;
13       lastName = last;
14
15       ++count;  // increment static count of employees
16       System.out.println( "Employee object constructor: " +
17          firstName + " " + lastName );
18    }
19
20    // subtract 1 from static count when garbage collector
21    // calls finalize to clean up object and output String
22    // indicating that finalize was called
23    protected void finalize()
24    {
25       --count;  // decrement static count of employees
26       System.out.println( "Employee object finalizer: " +
27          firstName + " " + lastName + "; count = " + count );
28    }
29
30    // get first name
31    public String getFirstName()
32    {
33       return firstName;
34    }
35
```

Outline

`Employee.java`

Line 6
`Employee` objects share one instance of **`count`**

Lines 23-28
Called when **`Employee`** is marked for garbage collection

56

```
36    // get last name
37    public String getLastName()
38    {
39       return lastName;
40    }
41
42    // static method to get static count value
43    public static int getCount()
44    {
45       return count;
46    }
47
48 }  // end class Employee
```

Outline

`Employee.java`

Lines 43-46 **`static`** method accesses **`static`** variable **`count`**


```
2  // Test Employee class with static class variable,
3  // static class method, and dynamic memory.
4  import javax.swing.*;
5
6  public class EmployeeTest {
7
8     // test class Employee
9     public static void main( String args[] )
10    {
11       // prove that count is 0 before creating Employees
12       String output = "Employees before instantiation: " +
13          Employee.getCount();
14
15       // create two Employees; count should be 2
16       Employee e1 = new Employee( "Susan", "Baker" );
17       Employee e2 = new Employee( "Bob", "Jones" );
18
19       // Prove that count is 2 after creating two Employees.
20       // Note: static methods should be called only via the
21       // class name for the class in which they are defined.
22       output += "\n\nEmployees after instantiation: " +
23          "\nvia e1.getCount(): " + e1.getCount() +
24          "\nvia e2.getCount(): " + e2.getCount() +
25          "\nvia Employee.getCount(): " + Employee.getCount();
26
27       // get names of Employees
28       output += "\n\nEmployee 1: " + e1.getFirstName() +
29          " " + e1.getLastName() + "\nEmployee 2: " +
30          e2.getFirstName() + " " + e2.getLastName();
31
```

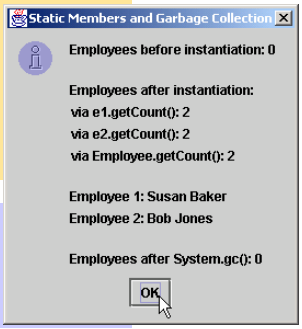
Outline

`EmployeeTest.java`

Line 13 **`EmployeeTest`** can invoke **`Employee`** **`static`** method, even though **`Employee`** has not been instantiated

```
32         // If there is only one reference to each employee (as
33         // on this example), the following statements mark
34         // those objects for garbage collection. Otherwise,
35         // these statement simply decrement the reference count
36         // for each object.
37         e1 = null;
38         e2 = null;
39
40         System.gc(); // suggest call to garbage collector
41
42         // Show Employee count after calling garbage collector.
43         // Count displayed may be 0, 1 or 2 depending on
44         // whether garbage collector executed immediately and
45         // number of Employee objects it collects.
46         output += "\n\nEmployees after System.gc(): " +
47            Employee.getCount();
48
49         JOptionPane.showMessageDialog( null, output,
50            "Static Members and Garbage Collection",
51            JOptionPane.INFORMATION_MESSAGE );
52
53         System.exit( 0 );
54      }
55
56   }  // end class EmployeeTest
```

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



Outline

EmployeeTest.java

Line 40
Calls Java's automatic garbage-collection mechanism

59

Data Abstraction and Encapsulation

- Encapsulation (data hiding)
  - Stack data structure
    - Last in-first out (LIFO)
    - Developer creates stack
      - Hides stack's implementation details from clients
    - Data abstraction
      - Abstract data types (ADTs)
- Abstract Data Type (ADT)
  - Queue
    - Line at grocery store
    - First-in, first-out (FIFO)
      - Enqueue to place objects in queue
      - Dequeue to remove object from queue
      - Enqueue and dequeue hide internal data representation

60