```
 1  // joi/1/bank/BankAccount.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A BankAccount object has a private field to keep track
 8   * of this account's current balance, and public methods to
 9   * return and change the balance.
10   *
11   * @see Bank
12   * @version 1
13   */
14
15  public class BankAccount
16  {
17      private int balance;  // work only in whole dollars
18
19      /**
20       * A constructor for creating a new bank account.
21       *
22       * @param initialBalance the opening balance.
23       */
24
25      public BankAccount( int initialBalance )
26      {
27          this.deposit( initialBalance );
28      }
29
30      /**
31       * Withdraw the amount requested.
32       *
33       * @param amount the amount to be withdrawn.
34       */
35
36      public void withdraw( int amount )
37      {
38          balance = balance - amount;
39      }
40
41      /**
42       * Deposit the amount requested.
43       *
44       * @param amount the amount to be deposited.
45       */
46
47      public void deposit( int amount )
48      {
49          balance = balance + amount;
50      }
51
52      /**
53       * The current account balance.
54       *
55       * @return the current balance.
56       */
```



```
57
58      public int getBalance()
59      {
60          return balance;
61      }
62  }
```

```
 1  // joi/1/bank/Bank.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A Bank object simulates the behavior of a simple bank/ATM.
 8   * It contains a Terminal object and two BankAccount objects.
 9   *
10   * Its single public method is open, which opens the this Bank
11   * for business, prompting the customer for input.
12   *
13   * To create a Bank and open it for business issue the command
14   * <code>java Bank</code>.
15   *
16   * @see BankAccount
17   * @version 1
18   */
19
20  public class Bank
21  {
22      private String bankName;      // the name of this Bank
23
24      private Terminal atm;         // for talking with the customer
25
26      private BankAccount account1; // two accounts to play with
27      private BankAccount account2;
28
29      private static final int INITIAL_BALANCE = 200;
30      private static final String HELPSTRING =
31          "Transactions: exit, help, deposit, withdraw, balance";
32
33      /**
34       * Construct a Bank with the given name.
35       * Create two new BankAccounts, each with a starting balance
36       * of initialBalance.
37       *
38       * @param name the name of the Bank.
39       */
40
41      public Bank( String name )
42      {
43          bankName = name;
44          atm      = new Terminal();
45          account1 = new BankAccount( INITIAL_BALANCE );
46          account2 = new BankAccount( INITIAL_BALANCE );
47      }
48
49      /**
50       * Open the Bank for business.
51       *
52       * Send a whichAccount message prompting for a BankAccount
53       * number, then send a processTransactionsForAccount
54       * message to do the work.
55       */
56
```



```
57      public void open()
58      {
59          atm.println( "Welcome to " + bankName );
60          boolean bankIsOpen = true;
61          while ( bankIsOpen ) {
62              BankAccount account = this.whichAccount();
63              if ( account == null ) {
64                  bankIsOpen = false;
65              }
66              else {
67                  this.processTransactionsForAccount(account);
68              }
69          }
70          atm.println( "Goodbye from " + bankName );
71      }
72
73      // Prompt the user for an account number and return the
74      // corresponding BankAccount object. Return null when
75      // the Bank is about to close.
76
77      private BankAccount whichAccount()
78      {
79          int  accountNumber =
80              atm.readInt("Account number (1 or 2), 0 to shut down: ");
81
82          if ( accountNumber == 1 ) {
83              return account1;
84          }
85          else if ( accountNumber == 2 ) {
86              return account2;
87          }
88          else if ( accountNumber == 0 ) {
89              return null;
90          }
91          else {
92              atm.println( "No account numbered " +
93                                  accountNumber + "; try again" );
94              return this.whichAccount();
95          }
96      }
97
98      // Prompt the user for transaction to process.
99      // Then send an appropriate message to account.
100
101     private void processTransactionsForAccount( BankAccount account)
102     {
103         atm.println( HELPSTRING );
104
105         boolean moreTransactions = true;
106         while ( moreTransactions ) {
107             String command = atm.readWord( "transaction: " );
108             if ( command.equals( "exit" ) ) {
109                 moreTransactions = false;
110             }
111             else if ( command.equals( "help" ) ) {
112                 atm.println( HELPSTRING );
```

```
113             }
114             else if ( command.equals( "deposit" ) ) {
115                 int amount = atm.readInt( "amount: " );
116                 account.deposit( amount );
117             }
118             else if ( command.equals( "withdraw" ) ) {
119                 int amount = atm.readInt( "amount: " );
120                 account.withdraw( amount );
121             }
122             else if ( command.equals( "balance" ) ) {
123                 atm.println( account.getBalance() );
124             }
125             else{
126                 atm.println("sorry, unknown transaction" );
127             }
128         }
129     }
130
131     /**
132      * The Bank simulation program begins here when the user
133      * issues the command <code>java Bank</code>.
134
135      * @param args the command line arguments (ignored).
136      */
137
138     public static void main( String[] args )
139     {
140         Bank javaBank = new Bank( "Engulf and Devour" );
141         javaBank.open();
142     }
143 }
```

```
 1  // joi/1/lights/TrafficLight.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  import java.awt.*;
 7  import java.awt.event.*;
 8
 9  /**
10   * A TrafficLight has three lenses: red, yellow and green.
11   * It can be set to signal Go, Caution, Stop or Walk.
12   *
13   * @version 1
14   */
15
16  public class TrafficLight extends Panel
17  {
18      // Three Lenses and a Button
19
20      private Lens red          = new Lens( Color.red );
21      private Lens yellow       = new Lens( Color.yellow );
22      private Lens green        = new Lens( Color.green );
23      private Button nextButton = new Button("Next");
24
25      /**
26       * Construct a traffic light.
27       */
28
29      public TrafficLight()
30      {
31          this.setLayout(new BorderLayout());
32
33          // create a Panel for the Lenses
34          Panel lensPanel = new Panel();
35          lensPanel.setLayout( new GridLayout( 3, 1 ) );
36          lensPanel.add( red );
37          lensPanel.add( yellow );
38          lensPanel.add( green );
39          this.add( BorderLayout.NORTH, lensPanel );
40
41          // configure the "Next" button
42          Sequencer sequencer = new Sequencer( this );
43          NextButtonListener payAttention =
44              new NextButtonListener( sequencer );
45          nextButton.addActionListener( payAttention );
46          this.add( BorderLayout.CENTER, nextButton);
47      }
48
49      // Methods that change the light
50
51      /**
52       * Set the light to stop (red).
53       */
54
55      public void setStop()
56      {
```



```
57          red.turnOn();
58          yellow.turnOff();
59          green.turnOff();
60      }
61
62      /**
63       * Set the light to caution (yellow).
64       */
65
66      public void setCaution()
67      {
68          red.turnOff();
69          yellow.turnOn();
70          green.turnOff();
71      }
72
73      /**
74       * Set the light to go (green).
75       */
76
77      public void setGo()
78      {
79          red.turnOff();
80          yellow.turnOff();
81          green.turnOn();
82      }
83
84      /**
85       * Set the light to walk.
86       *
87       * (In Boston, red and yellow signal walk.)
88       */
89
90      public void setWalk()
91      {
92          red.turnOn();
93          yellow.turnOn();
94          green.turnOff();
95      }
96
97      /**
98       * The traffic light simulation starts at main.
99       *
100      * @param args ignored.
101      */
102
103     public static void main( String[] args )
104     {
105         Frame frame         = new Frame();
106         TrafficLight light  = new TrafficLight();
107         frame.add( light );
108         frame.addWindowListener( new ShutDownLight() );
109         frame.pack();
110         frame.show();
111     }
112
```

```
   113       // A ShutDownLight instance handles close events generated
   114       // by the underlying window system with its windowClosing
   115       // method.
   116       //
   117       // This is an inner class, declared inside the
   118       // TrafficLight class since it's used only here.
   119
   120      private static class ShutDownLight extends WindowAdapter
   121      {
   122           // Close the window by shutting down the light.
   123
   124          public void windowClosing (WindowEvent e)
   125          {
   126              System.exit(0);
   127          }
   128      }
   129  }
   130
   131
```

```
 1  // joi/1/lights/NextButtonListener.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  import java.awt.event.*;
 7
 8  /**
 9   * A NextButtonListener sends a "next" message to its
10   * Sequencer each time a button to which it is listening
11   * is pressed.
12   *
13   * @version 1
14   */
15
16  public class NextButtonListener implements ActionListener
17  {
18      private Sequencer sequencer;
19
20      /**
21       * Construct a listener that "listens for" a user's
22       * pressing the "Next" button.
23       *
24       * @param sequencer the Sequencer for the TrafficLight.
25       */
26
27      public NextButtonListener( Sequencer sequencer )
28      {
29          this.sequencer = sequencer;
30      }
31
32      /**
33       * The action performed when a push of the button is detected:
34       * send a next message to the Sequencer to advance it to
35       * its next state.
36       *
37       * @param event the event detected at the button.
38       */
39
40      public void actionPerformed( ActionEvent event )
41      {
42          this.sequencer.next();
43      }
44  }
```

```
 1  // joi/1/lights/Sequencer.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A Sequencer controls a TrafficLight. It maintains fields
 8   * for the light itself and the current state of the light.
 9   *
10   * Each time it receives a "next" message, it advances to the
11   * next state and sends the light an appropriate message.
12   *
13   * @version 1
14   */
15
16  public class Sequencer
17  {
18      // the TrafficLight this Sequencer controls
19      private TrafficLight light;
20
21      // represent the states by ints
22      private final static int GO       = 0;
23      private final static int CAUTION  = 1;
24      private final static int STOP     = 2;
25
26      private int currentState;
27
28      /**
29       * Construct a sequencer to control a TrafficLight.
30       *
31       * @param light the TrafficLight we wish to control.
32       */
33
34      public Sequencer( TrafficLight light )
35      {
36          this.light = light;
37          this.currentState = GO;
38          this.light.setGo();
39      }
40
41      /**
42       * How the light changes when a next Button is pressed
43       * depends on the current state. The sequence is
44       * GO -> CAUTION -> STOP -> GO.
45       */
46
47      public void next()
48      {
49          switch ( currentState ) {
50
51          case GO:
52              this.currentState = CAUTION;
53              this.light.setCaution();
54              break;
55
56          case CAUTION:
```



```
57              this.currentState = STOP;
58              this.light.setStop();
59              break;
60
61          case STOP:
62              this.currentState = GO;
63              this.light.setGo();
64              break;
65
66          default: // This will never happen
67              System.err.println("What color is the light?!");
68          }
69      }
70  }
```

```
 1  // joi/1/lights/Lens.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  import java.awt.*;
 7
 8  /**
 9   * A Lens has a certain color and can either be turned on
10   * (the color) or turned off (black).
11   *
12   * @version 1
13   */
14
15  public class Lens extends Canvas
16  {
17      private Color onColor;               // color on
18      private Color offColor = Color.black; // color off
19      private Color currentColor;          // color the lens is now
20
21      private final static int SIZE = 100;  // how big is this Lens?
22      private final static int OFFSET = 20; // offset of Lens in Canvas
23
24      /**
25       * Construct a Lens to display a given color.
26       *
27       * The lens is black when it's turned off.
28       *
29       * @param color the color of the lens when it is turned on.
30       */
31
32      public Lens( Color color )
33      {
34          this.setBackground( Color.black );
35          this.onColor = color;
36          this.setSize( SIZE , SIZE );
37          this.turnOff();
38      }
39
40      /**
41       * How this Lens paints itself.
42       *
43       * @param g a Graphics object to manage brush and color information.
44       */
45
46      public void paint( Graphics g )
47      {
48          g.setColor( this.currentColor );
49          g.fillOval( OFFSET, OFFSET,
50                      SIZE - OFFSET*2, SIZE - OFFSET*2 );
51      }
52
53      /**
54       * Have this Lens display its color.
55       */
56
```



```
57      public void turnOn()
58      {
59          currentColor = onColor;
60          this.repaint();
61      }
62
63      /**
64       * Darken this lens.
65       */
66
67      public void turnOff()
68      {
69          currentColor = offColor;
70          this.repaint();
71      }
72  }
```

```
// joi/1/estore/EStore.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * An EStore object simulates the behavior of a simple on line
 * shopping web site.
 *
 * It contains a Terminal object to model the customer's browser
 * and several Item objects a customer can add to her ShoppingCart.
 *
 * @version 1
 */

public class EStore
{
    private String storeName = "Virtual Minimal Minimall";

    // Use a Terminal object to communicate with customers.
    private Terminal browser = new Terminal();

    // The store stocks two kinds of Items.
    private Item widget = new Item(10); //  widgets cost $10
    private Item gadget = new Item(13); //  gadgets cost $13

    private String selectionList = "(gadget, widget, checkout)";

    /**
     * Visit this EStore.
     *
     * Loop allowing visitor to select items to add to her
     * ShoppingCart.
     */

    public void visit()
    {
        // Create a new, empty ShoppingCart.
        ShoppingCart basket = new ShoppingCart();

        // Print a friendly welcome message.
        browser.println( "Welcome to " + storeName );

        // Change to false when customer is ready to leave:
        boolean stillShopping = true;

        while ( stillShopping ) {
            Item nextPurchase = selectItem();
            if ( nextPurchase == null ) {
                stillShopping = false;
            }
            else {
                basket.add( nextPurchase );
            }
        }
        int numberPurchased = basket.getCount();
```



```
        int totalCost       = basket.getCost();
        browser.println("We are shipping " + numberPurchased + " Items");
        browser.println("and charging your account $" + totalCost);
        browser.println("Thank you for shopping at " + storeName);
    }

    // Discover what the customer wants to do next:
    //  send browser a message to get customer input
    //  examine response to make a choice
    //  if response makes no sense give customer another chance

    private Item selectItem()
    {
        String itemName =
            browser.readWord("Item " + selectionList + ":");

        if ( itemName.equals("widget")) {
            return widget;
        }
        else if ( itemName.equals("gadget")) {
            return gadget;
        }
        else if ( itemName.equals("checkout" )) {
            return null;
        }
        else {
            browser.println( "No item named " +
                             itemName + "; try again" );
            return selectItem();  // try again
        }
    }

    /**
     *  The EStore simulation program begins here when the user
     *  issues the command <code>java EStore</code>.
     */

    public static void main( String[] args )
    {
        // Print this to simulate delay while browser finds store
        System.out.println("connecting ...");

        // Create the EStore object.
        EStore webSite = new EStore();

        // Visit it.
        webSite.visit();
    }
} // end of class EStore
```

```
 1  // joi/1/estore/Item.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * An Item models an object that might be stocked in a store.
 8   * Each Item has a cost.
 9   *
10   * @version 1
11   */
12
13  public class Item
14  {
15      private int cost;
16
17      /**
18       *  Construct an Item object.
19       *
20       *  @param itemCost the cost of this Item.
21       */
22
23      public Item( int itemCost )
24      {
25          cost = itemCost;
26      }
27
28      /**
29       * How much does this Item cost?
30       *
31       * @return the cost.
32       */
33
34      public int getCost()
35      {
36          return cost;
37      }
38  }
```

```
 1  // joi/1/estore/ShoppingCart.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A ShoppingCart keeps track of a customer's purchases.
 8   *
 9   * @see EStore
10   * @version 1
11   */
12
13  public class ShoppingCart
14  {
15      private int count; // number of Items in this ShoppingCart
16      private int cost;  // cost of Items in this ShoppingCart
17
18      /**
19       *  Construct a new empty ShoppingCart.
20       */
21
22      public ShoppingCart()
23      {
24          count = 0;
25          cost  = 0;
26      }
27
28      /**
29       * When this ShoppingCart is asked to add an Item to itself
30       * it updates its count field and then updates its cost
31       * field by sending the Item a getCost message.
32       *
33       * @param purchase the Item being added to this ShoppingCart.
34       */
35
36      public void add( Item purchase )
37      {
38          count++; // Java idiom for count = count + 1;
39          cost = cost + purchase.getCost();
40      }
41
42      /**
43       * What happens when this ShoppingCart is asked how many
44       * Items it contains.
45       *
46       * @return the count of Items.
47       */
48
49      public int getCount()
50      {
51          return count;
52      }
53
54      /**
55       * What happens when this ShoppingCart is asked the total
56       * cost of the Items it contains.
```



```
57       *
58       * @return the total cost.
59       */
60
61      public int getCost()
62      {
63          return cost;
64      }
65  }
```