```
// joi/4/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 a Catalog of Items that may be purchased and
 * then added to the customer's shoppingCart.
 *
 * @version 4
 */

public class EStore
{
    private String   storeName;
    private Terminal browser;
    private Catalog  catalog;

    /**
     *  Construct a new EStore.
     *
     *  @param storeName the name of the EStore
     *  @param browser the visitor's Terminal.
     */

    public EStore( String storeName, Terminal browser )
    {
        this.browser   = browser;
        this.storeName = storeName;
        this.catalog   = new Catalog();
        catalog.addItem( new Item("quaffle", 55) );
        catalog.addItem( new Item("bludger", 15) );
        catalog.addItem( new Item("snitch",  1000) );
    }

    /**
     *  Visit this EStore.
     *
     *  Execution starts here when the store opens for
     *  business. User can visit as a customer, act as
     *  the manager, or exit.
     */

    public void visit()
    {
        // Print a friendly welcome message.
        browser.println( "Welcome to " + storeName );
        while (true) { // an infinite loop ...
            browser.println();
            String whoAreYou = browser.readWord(
               storeName + " (manager, visit, exit): ");
            if (whoAreYou.equals("exit")) {
```



```
                break; // leave the while loop
            }
            if (whoAreYou.equals("manager")) {
                managerVisit();
            }
            if (whoAreYou.equals("visit")) {
                customerVisit();
            }
        }
    }

    /**
     *  Manager options:
     *
     *  examine the catalog
     *  add an Item to the catalog
     *  quit
     */
    private void managerVisit( )
    {
        while (true) {
            String cmd =
              browser.readWord("manager command (show, new, quit):");
            if (cmd.equals("quit")) {
                break;  // leave manager command while loop
            }
            else if (cmd.equals("show")) {
                catalog.show(browser);
            }
            else if (cmd.equals("new")) {
                String itemName = browser.readWord("  item name:     ");
                int cost        = browser.readInt( "  cost:          ");
                catalog.addItem( new Item(itemName, cost) );
            }
            else {
                browser.println("unknown manager command: " + cmd);
            }
        }
    }

    /**
     * Customer visits this EStore.
     *
     * Loop allowing customer to select items to add to her
     * shoppingCart.
     */

    private void customerVisit( )
    {
        // Create a new, empty ShoppingCart.
        ShoppingCart basket = new ShoppingCart();

        browser.println( "Currently available:");
        catalog.show(browser);
        while ( true ) { // loop forever ...
            String nextPurchase = browser.readWord(
```

```
113                     "select your purchase, checkout, help: ");
114
115             if ( nextPurchase.equals("checkout" )) break; // leave loop!
116
117             if ( nextPurchase.equals("help" )) {
118                 catalog.show(browser);
119                 continue; // go back to top of while loop
120             }
121             // customer has entered the name of an Item
122             basket.addItem( catalog.getItem(nextPurchase) );
123         }
124
125         int numberPurchased = basket.getCount();
126         browser.println("We are shipping these " +
127                         basket.getCount() + " Items:");
128         basket.showContents(browser);
129         browser.println("and charging your account $" + basket.getCost())
130         browser.println("Thank you for shopping at " + storeName);
131     }
132
133     /**
134      *  The EStore simulation program begins here when the user
135      *  issues the command <code>java EStore</code>
136      *
137      *  If first command line argument is "-e" instantiate a
138      *  Terminal that echoes its input.
139      *
140      *  The next command line argument (if there is one)
141      *  is the name of the EStore.
142      *
143      *  @param args <-e> <storeName>
144      */
145     public static void main( String[] args )
146     {
147
148         String storeName = "Virtual Minimal Minimall"; //default
149
150         // check to see if first argument is "-e"
151         boolean echo = ( (args.length > 0) && (args[0].equals("-e")) );
152
153         // if first argument was "-e" then look at second for store name
154         int nextArg = (echo ? 1 : 0 );
155
156         if (args.length > nextArg) {
157             storeName = args[nextArg];
158         }
159
160         // Print this to simulate internet search.
161         System.out.println("connecting ...");
162
163         // Create an EStore object and visit it
164         (new EStore(storeName, new Terminal(echo))).visit();
165     }
166 }
```

```
// joi/4/estore/ShoppingCart.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * A ShoppingCart keeps track of a customer's purchases.
 *
 * @see EStore
 * @version 4
 */

public class ShoppingCart
{
    /// replace these two fields by a single ArrayList
    private int count; // number of Items in this ShoppingCart
    private int cost;  // total cost of Items in this ShoppingCart

    /**
     * Construct a new empty ShoppingCart.
     */

    public ShoppingCart()
    {
        count = 0;
        cost  = 0;
    }

    /**
     * Add an Item to this ShoppingCart.
     *
     * @param item the Item to add.
     */

    public void addItem( Item item )
    {
        /// this code just keeps track of the totals
        /// replace it with code that adds the item to the list
        count++;
        this.cost += item.getCost(); // Java idiom: a += b means a = a +
    }

    /**
     * Return an Item from this ShoppingCart.
     *
     * @param item the Item to return.
     */

    public void returnItem( Item item )
    {
        /// look through the list looking for Item
        /// remove it if it's there
    }

    /**
     * What happens when this ShoppingCart is asked how many
```



```
     * Items it contains.
     *
     * @return the number of items in this ShoppingCart.
     */

    public int getCount()
    {
        /// get this information from the list,
        /// since the count field no longer exists
        return count;
    }

    /**
     * What happens when this ShoppingCart is asked the total
     * cost of the Items it contains.
     *
     * @return the total cost of the items in this ShoppingCart.
     */
    public int getCost()
    {
        /// get this information from the list,
        /// since the cost field no longer exists
        return cost;
    }

    /**
     * Write the contents of this ShoppingCart to a Terminal.
     *
     * @param t the Terminal to use for output.
     */

    public void showContents( Terminal t)
    {
        /// work to do here ...
        t.println("  [sorry, can't yet print ShoppingCart contents]");
    }
}
```

```
 1  // joi/4/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 4
11   */
12
13  public class Item
14  {
15      private int cost;
16      private String name;
17
18      /**
19       *  Construct an Item object.
20       *
21       *  @param name the nme of this Item.
22       *  @param cost the cost of this Item.
23       */
24
25      public Item( String name, int cost )
26      {
27          this.name = name;
28          this.cost = cost;
29      }
30
31      /**
32       * How much does this Item cost?
33       *
34       * @return the cost.
35       */
36
37      public int getCost()
38      {
39          return cost;
40      }
41
42      /**
43       * What is this Item called?
44       *
45       * @return the name.
46       */
47
48      public String getName()
49      {
50          return name;
51      }
52  }
```

```
 1  // joi/4/estore/Catalog.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  import java.util.TreeMap;
 7
 8  /**
 9   * A Catalog models the collection of Items that an
10   * EStore might carry.
11   *
12   * @see EStore
13   *
14   * @version 4
15   */
16
17  public class Catalog
18  {
19      private TreeMap items;
20
21      /**
22       *  Construct a Catalog object.
23       */
24
25      public Catalog( )
26      {
27          items = new TreeMap();
28      }
29
30      /**
31       * Add an Item to this Catalog.
32       *
33       * @param item the Item to add.
34       */
35
36      public void addItem( Item item )
37      {
38          items.put( item.getName(), item );
39      }
40
41      /**
42       * Get an Item from this Catalog.
43       *
44       * @param itemName the name of the wanted Item
45       *
46       * @return the Item, null if none.
47       */
48
49      public Item getItem( String itemName )
50      {
51          return (Item)items.get(itemName);
52      }
53
54      /**
55       * Display the contents of this Catalog.
56       *
```



```
57       * @param t the Terminal to print to.
58       */
59
60      public void show( Terminal t )
61      {
62          // loop on items, printing name and cost
63          t.println("  [sorry, can't yet print Catalog contents]");
64      }
65  }
```