

```

1 // foj/4/estore/ESTore.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 /**
7  * An EStore object simulates the behavior of a simple on line
8  * shopping web site.
9
10 * It contains a Terminal object to model the customer's browser
11 * and a Catalog of Items that may be purchased and
12 * then added to the customer's shoppingCart.
13
14 * @version 4
15 */
16
17 public class EStore
18 {
19     private String  storeName;
20     private Terminal browser;
21     private Catalog catalog;
22
23     /**
24      * Construct a new EStore.
25      *
26      * @param storeName the name of the EStore
27      * @param browser the visitor's Terminal.
28      */
29
30     public EStore( String storeName, Terminal browser )
31     {
32         this.browser = browser;
33         this.storeName = storeName;
34         this.catalog = new Catalog();
35         catalog.addItem( new Item("quaffle", 55) );
36         catalog.addItem( new Item("bludger", 15) );
37         catalog.addItem( new Item("snitch", 1000) );
38     }
39
40     /**
41      * Visit this EStore.
42      *
43      * Execution starts here when the store opens for
44      * business. User can visit as a customer, act as
45      * the manager, or exit.
46      */
47
48     public void visit()
49     {
50         // Print a friendly welcome message.
51         browser.println( "Welcome to " + storeName );
52         while (true) { // an infinite loop ...
53             browser.println();
54             String whoAreYou = browser.readWord(
55                 storeName + " (manager, visit, exit): ");
56             if (whoAreYou.equals("exit")) {

```

```

57         break; // leave the while loop
58     }
59     if (whoAreYou.equals("manager")) {
60         managerVisit();
61     }
62     if (whoAreYou.equals("visit")) {
63         customerVisit();
64     }
65 }
66
67 /**
68  * Manager options:
69  *
70  * examine the catalog
71  * add an Item to the catalog
72  * quit
73 */
74
75 private void managerVisit( )
76 {
77     while (true) {
78         String cmd =
79             browser.readWord("manager command (show, new, quit):");
80         if (cmd.equals("quit")) {
81             break; // leave manager command while loop
82         }
83         else if (cmd.equals("show")) {
84             catalog.show(browser);
85         }
86         else if (cmd.equals("new")) {
87             String itemName = browser.readWord(" item name: ");
88             int cost = browser.readInt(" cost: ");
89             catalog.addItem( new Item(itemName, cost) );
90         }
91         else {
92             browser.println("unknown manager command: " + cmd);
93         }
94     }
95 }
96
97 /**
98  * Customer visits this EStore.
99  *
100 * Loop allowing customer to select items to add to her
101 * shoppingCart.
102 */
103
104 private void customerVisit( )
105 {
106     // Create a new, empty ShoppingCart.
107     ShoppingCart basket = new ShoppingCart();
108     browser.println( "Currently available:");
109     catalog.show(browser);
110     while ( true ) { // loop forever ...
111         String nextPurchase = browser.readWord(
112

```

```
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 Minimal"; //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 }
```

```

1 // foj/4/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 4
11 */
12
13 public class ShoppingCart
14 {
15     /** replace these two fields by a single ArrayList
16     private int count; // number of Items in this ShoppingCart
17     private int cost; // total cost of Items in this ShoppingCart
18
19     /**
20     * Construct a new empty ShoppingCart.
21     */
22
23     public ShoppingCart()
24     {
25         count = 0;
26         cost = 0;
27     }
28
29     /**
30     * Add an Item to this ShoppingCart.
31     *
32     * @param item the Item to add.
33     */
34
35     public void addItem( Item item )
36     {
37         /** this code just keeps track of the totals
38         /** replace it with code that adds the item to the list
39         count++;
40         this.cost += item.getCost(); // Java idiom: a += b means a = a +
41         }
42
43     /**
44     * Return an Item from this ShoppingCart.
45     *
46     * @param item the Item to return.
47     */
48
49     public void returnItem( Item item )
50     {
51         /** look through the list looking for Item
52         /** remove it if it's there
53         }
54
55     /**
56     * What happens when this ShoppingCart is asked how many

```

```

57
58     * Items it contains.
59     * @return the number of items in this ShoppingCart.
60     */
61
62     public int getCount()
63     {
64         /** get this information from the list,
65         /** since the count field no longer exists
66         return count;
67     }
68
69     /**
70     * What happens when this ShoppingCart is asked the total
71     * cost of the Items it contains.
72     *
73     * @return the total cost of the items in this ShoppingCart.
74     */
75     public int getCost()
76     {
77         /** get this information from the list,
78         /** since the cost field no longer exists
79         return cost;
80     }
81
82     /**
83     * Write the contents of this ShoppingCart to a Terminal.
84     *
85     * @param t the Terminal to use for output.
86     */
87
88     public void showContents( Terminal t )
89     {
90         /** work to do here ...
91         t.println(" [sorry, can't yet print ShoppingCart contents]");
92     }
93 }

```

```
1 // fo1/4/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 // foj/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 }

```