9.8 An Online Store with a Shopping Cart and Session Tracking

This section gives an extended example of how you might build an online store that uses session tracking. The first subsection shows how to build pages that display items for sale. The code for each display page simply lists the page title and the identifiers of the items listed on the page. The actual page is then built automatically by methods in the parent class, based on item descriptions stored in the catalog. The second subsection shows the page that handles the orders. It uses session tracking to associate a shopping cart with each user and permits the user to modify orders for any previously selected item. The third subsection presents the implementation of the shopping cart, the data structures representing individual items and orders, and the catalog.

Creating the Front End

Listing 9.4 presents an abstract base class used as a starting point for servlets that want to display items for sale. It takes the identifiers for the items for sale, looks them up in the catalog, and uses the descriptions and prices found there to present an order page to the user. Listing 9.5 (with the result shown in Figure 9-6) and Listing 9.6 (with the result shown in Figure 9-7) show how easy it is to build actual pages with this parent class.

Listing 9.4 CatalogPage.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Base class for pages showing catalog entries.
 *  Servlets that extend this class must specify
 *  the catalog entries that they are selling and the page
 *  title <I>before</I> the servlet is ever accessed. This
 *  is done by putting calls to setItems and setTitle
 *  in init.
 */


public abstract class CatalogPage extends HttpServlet {
  private CatalogItem[] items;
  private String[] itemIDs;
  private String title;

  /** Given an array of item IDs, look them up in the
   *  Catalog and put their corresponding CatalogItem entry
   *  into the items array. The CatalogItem contains a short
   *  description, a long description, and a price,
   *  using the item ID as the unique key.
   *  <P>
   *  Servlets that extend CatalogPage <B>must</B> call
   *  this method (usually from init) before the servlet
   *  is accessed.
   */

  protected void setItems(String[] itemIDs) {
    this.itemIDs = itemIDs;
    items = new CatalogItem[itemIDs.length];
    for(int i=0; i<items.length; i++) {
```

```
    items[i] = Catalog.getItem(itemIDs[i]);
  }
}

/** Sets the page title, which is displayed in
 *  an H1 heading in resultant page.
 *  <P>
 *  Servlets that extend CatalogPage <B>must</B> call
 *  this method (usually from init) before the servlet
 *  is accessed.
 */

protected void setTitle(String title) {
  this.title = title;
}

/** First display title, then, for each catalog item,
 *  put its short description in a level-two (H2) heading
 *  with the price in parentheses and long description
 *  below. Below each entry, put an order button
 *  that submits info to the OrderPage servlet for
 *  the associated catalog entry.
 *  <P>
 *  To see the HTML that results from this method, do
 *  "View Source" on KidsBooksPage or TechBooksPage, two
 *  concrete classes that extend this abstract class.
 */

public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  if (items == null) {
    response.sendError(response.SC_NOT_FOUND,
                       "Missing Items.");
    return;
  }
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String docType =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">\n";
  out.println(docType +
              "<HTML>\n" +
              "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
              "<BODY BGCOLOR=\"#FDF5E6\">\n" +
              "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
  CatalogItem item;
  for(int i=0; i<items.length; i++) {
    out.println("<HR>");
    item = items[i];
    // Show error message if subclass lists item ID
    // that's not in the catalog.
    if (item == null) {
      out.println("<FONT COLOR=\"RED\">" +
                  "Unknown item ID " + itemIDs[i] +
                  "</FONT>");
    } else {
      out.println();
      String formURL =
        "/servlet/coreservlets.OrderPage";
      // Pass URLs that reference own site through encodeURL.
      formURL = response.encodeURL(formURL);
      out.println
        ("<FORM ACTION=\"" + formURL + "\">\n" +
```

```
            "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\" " +
            "       VALUE=\"" + item.getItemID() + "\">\n" +
            "<H2>" + item.getShortDescription() +
            " ($" + item.getCost() + ")</H2>\n" +
            item.getLongDescription() + "\n" +
            "<P>\n<CENTER>\n" +
            "<INPUT TYPE=\"SUBMIT\" " +
            "VALUE=\"Add to Shopping Cart\">\n" +
            "</CENTER>\n<P>\n</FORM>");
      }
    }
    out.println("<HR>\n</BODY></HTML>");
  }
}
```

Listing 9.5 KidsBooksPage.java

```
package coreservlets;

/** A specialization of the CatalogPage servlet that
 *  displays a page selling three famous kids-book series.
 *  Orders are sent to the OrderPage servlet.
 */

public class KidsBooksPage extends CatalogPage {
  public void init() {
    String[] ids = { "lewis001", "alexander001", "rowling001" };
    setItems(ids);
    setTitle("All-Time Best Children's Fantasy Books");
  }
}
```

Listing 9.6 TechBooksPage.java

```
package coreservlets;

/** A specialization of the CatalogPage servlet that
 *  displays a page selling two famous computer books.
 *  Orders are sent to the OrderPage servlet.
 */

public class TechBooksPage extends CatalogPage {
  public void init() {
    String[] ids = { "hall001", "hall002" };
    setItems(ids);
    setTitle("All-Time Best Computer Books");
  }
}
```

Figure 9-6. Result of the `KidsBooksPage` servlet.

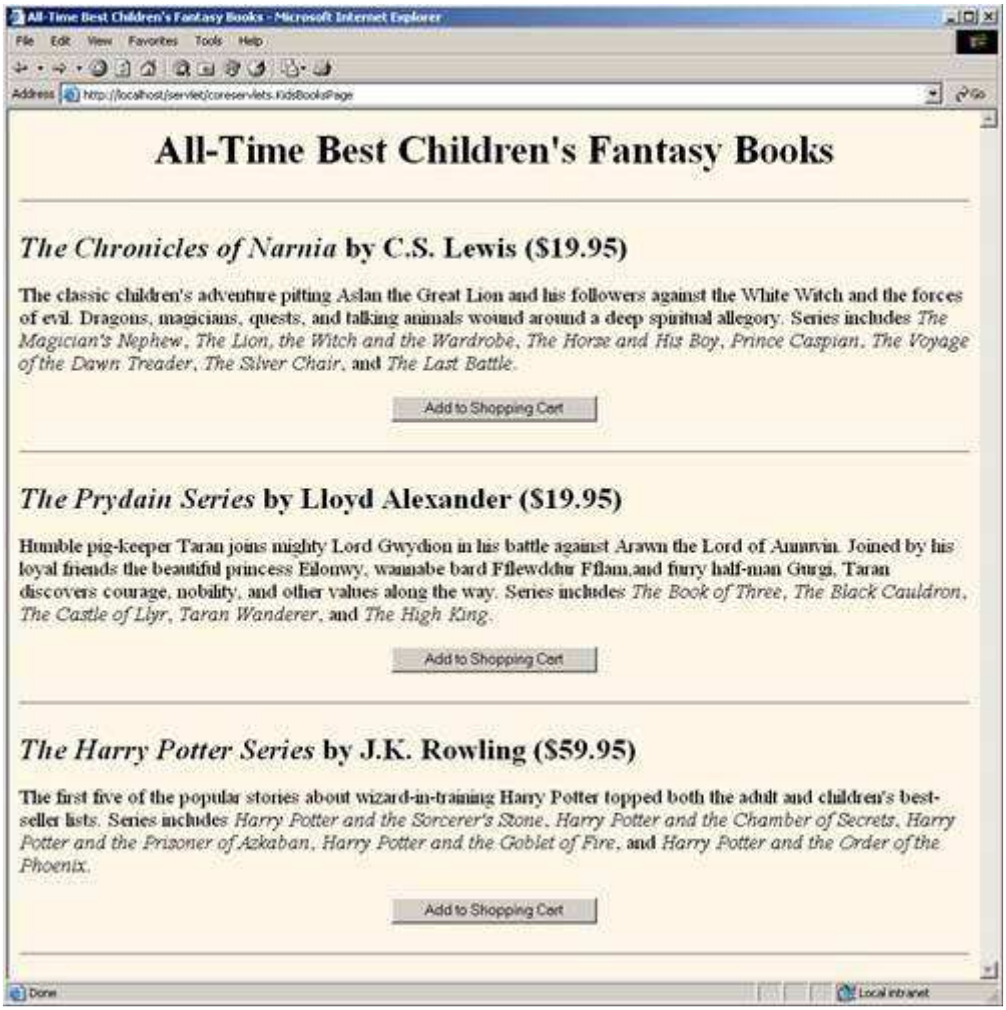
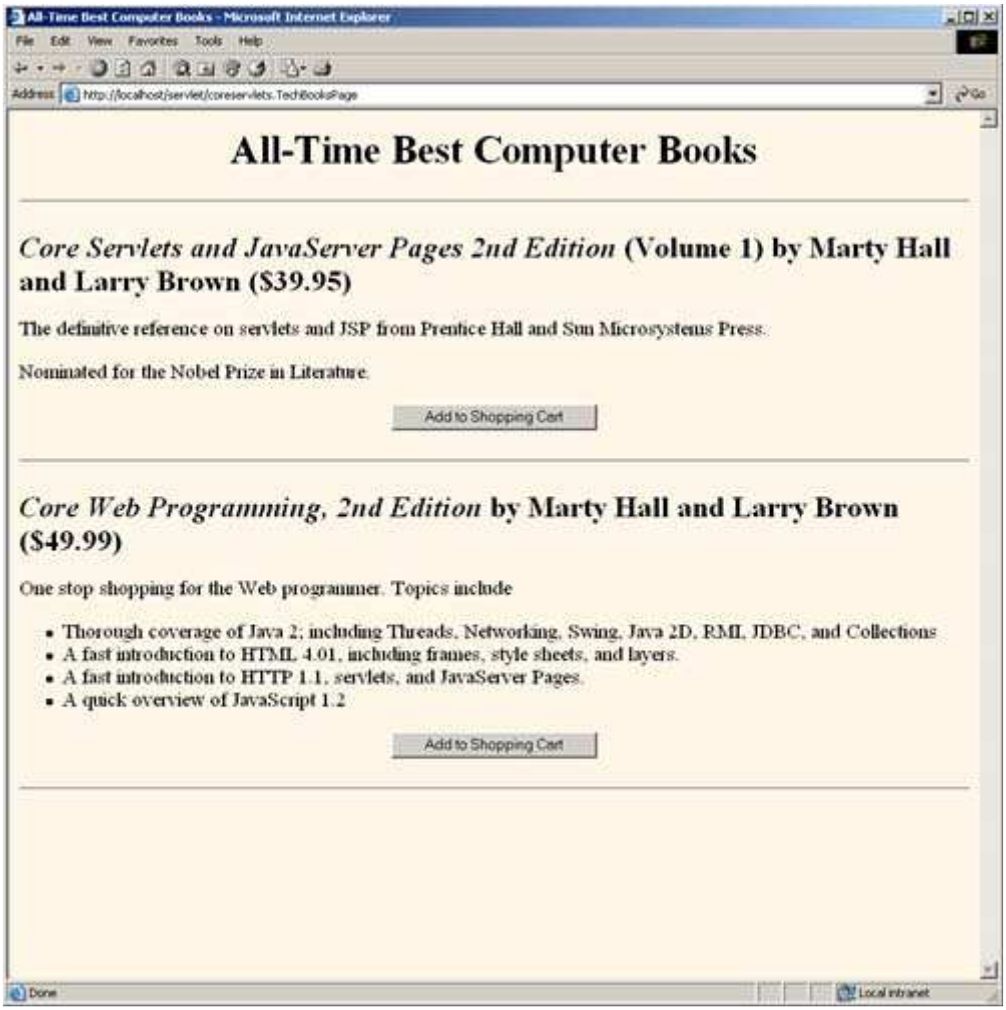


Figure 9-7. Result of the `TechBooksPage` servlet.



Handling the Orders

Listing 9.7 shows the servlet that handles the orders coming from the various catalog pages shown in the previous subsection. It uses session tracking to associate a shopping cart with each user. Since each user has a separate session, it is unlikely that multiple threads will be accessing the same shopping cart simultaneously. However, if you were paranoid, you could conceive of a few circumstances in which concurrent access could occur, such as when a single user has multiple browser windows open and sends updates from more than one in quick succession. So, just to be safe, the code synchronizes access based upon the session object. This synchronization prevents other threads that use the same session from accessing the data concurrently, while still allowing simultaneous requests from different users to proceed. Figures 9-8 and 9-9 show some typical results.

Listing 9.7 OrderPage.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.text.*;

/** Shows all items currently in ShoppingCart. Clients
 *  have their own session that keeps track of which
 *  ShoppingCart is theirs. If this is their first visit
 *  to the order page, a new shopping cart is created.
 *  Usually, people come to this page by way of a page
```

```
 *  showing catalog entries, so this page adds an additional
 *  item to the shopping cart. But users can also
 *  bookmark this page, access it from their history list,
 *  or be sent back to it by clicking on the "Update Order"
 *  button after changing the number of items ordered.
 */

public class OrderPage extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession();
    ShoppingCart cart;
    synchronized(session) {
      cart = (ShoppingCart)session.getAttribute("shoppingCart");
      // New visitors get a fresh shopping cart.
      // Previous visitors keep using their existing cart.
      if (cart == null) {
        cart = new ShoppingCart();
        session.setAttribute("shoppingCart", cart);
      }
      if (itemID != null) {
        String numItemsString =
          request.getParameter("numItems");
        if (numItemsString == null) {
          // If request specified an ID but no number,
          // then customers came here via an "Add Item to Cart"
          // button on a catalog page.
          cart.addItem(itemID);
        } else {
          // If request specified an ID and number, then
          // customers came here via an "Update Order" button
          // after changing the number of items in order.
          // Note that specifying a number of 0 results
          // in item being deleted from cart.
          int numItems;
          try {
            numItems = Integer.parseInt(numItemsString);
          } catch(NumberFormatException nfe) {
            numItems = 1;
          }
          cart.setNumOrdered(itemID, numItems);
        }
      }
    }
    // Whether or not the customer changed the order, show
    // order status.
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Status of Your Order";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
    synchronized(session) {
      List itemsOrdered = cart.getItemsOrdered();
      if (itemsOrdered.size() == 0) {
        out.println("<H2><I>No items in your cart...</I></H2>");
      } else {
        // If there is at least one item in cart, show table
```

```
        // of items ordered.
        out.println
      String itemID = request.getParameter("itemID");
          ("<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
           "<TR BGCOLOR=\"#FFAD00\">\n" +
           "  <TH>Item ID<TH>Description\n" +
           "  <TH>Unit Cost<TH>Number<TH>Total Cost");
        ItemOrder order;
        // Rounds to two decimal places, inserts dollar
        // sign (or other currency symbol), etc., as
        // appropriate in current Locale.
        NumberFormat formatter =
          NumberFormat.getCurrencyInstance();
        // For each entry in shopping cart, make
        // table row showing ID, description, per-item
        // cost, number ordered, and total cost.
        // Put number ordered in textfield that user
        // can change, with "Update Order" button next
        // to it, which resubmits to this same page
        // but specifying a different number of items.
        for(int i=0; i<itemsOrdered.size(); i++) {
          order = (ItemOrder)itemsOrdered.get(i);
          out.println
            ("<TR>\n" +
             "  <TD>" + order.getItemID() + "\n" +
             "  <TD>" + order.getShortDescription() + "\n" +
             "  <TD>" +
             formatter.format(order.getUnitCost()) + "\n" +
             "  <TD>" +
             "<FORM>\n" +  // Submit to current URL
             "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\"\n" +
             "       VALUE=\"" + order.getItemID() + "\">\n" +
             "<INPUT TYPE=\"TEXT\" NAME=\"numItems\"\n" +
             "       SIZE=3 VALUE=\"" +
             order.getNumItems() + "\">\n" +
             "<SMALL>\n" +
             "<INPUT TYPE=\"SUBMIT\"\n "+
             "       VALUE=\"Update Order\">\n" +
             "</SMALL>\n" +
             "</FORM>\n" +
             "  <TD>" +
             formatter.format(order.getTotalCost()));
        }
        String checkoutURL =
          response.encodeURL("../Checkout.html");
        // "Proceed to Checkout" button below table
        out.println
          ("</TABLE>\n" +
           "<FORM ACTION=\"" + checkoutURL + "\">\n" +
           "<BIG><CENTER>\n" +
           "<INPUT TYPE=\"SUBMIT\"\n" +
           "       VALUE=\"Proceed to Checkout\">\n" +
           "</CENTER></BIG></FORM>");
      }
      out.println("</BODY></HTML>");
    }
  }
}
```

Listing 9.8 Checkout.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
```

```
  <TITLE>Checking Out</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Checking Out</H1>
We are sorry, but our electronic credit-card-processing
system is currently out of order. Please send a check
to:
<PRE>
  Marty Hall
  coreservlets.com, Inc.
  300 Red Brook Blvd., Suite 400
  Owings Mills, MD 21117
</PRE>
Since we have not yet calculated shipping charges, please
sign the check but do not fill in the amount. We will
generously do that for you.
</BODY></HTML>
```

Figure 9-8. Result of `OrderPage` servlet after user clicks on "Add to Shopping Cart" in `KidsBooksPage`.

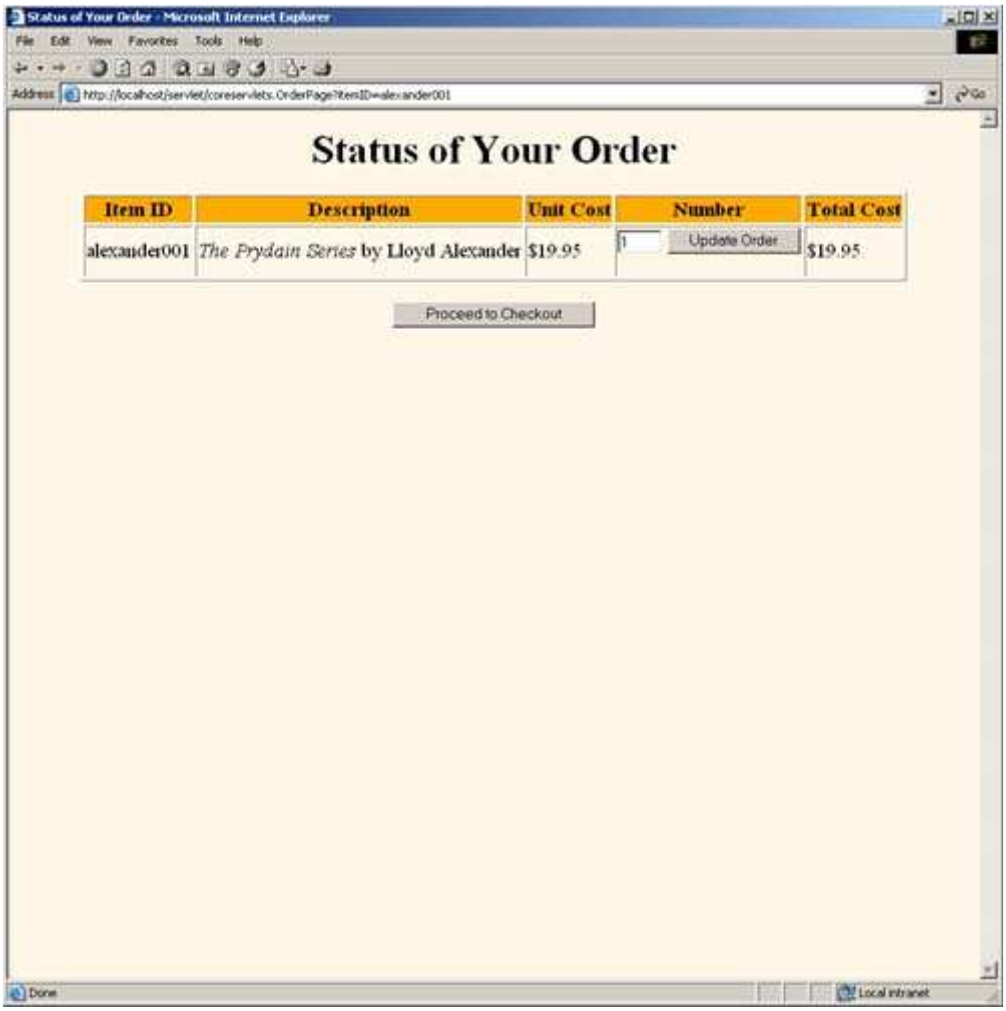


Figure 9-9. Result of `OrderPage` servlet after several additions and changes to the order.

Behind the Scenes: Implementing the Shopping Cart and Catalog Items

Listing 9.9 gives the shopping cart implementation. It simply maintains a `List` of orders, with methods to add and update these orders. Listing 9.10 shows the code for an individual catalog item, Listing 9.11 presents the class representing the order status of a particular item, and Listing 9.12 gives the catalog implementation.

Listing 9.9 ShoppingCart.java

```
package coreservlets;

import java.util.*;

/** A shopping cart data structure used to track orders.
 *  The OrderPage servlet associates one of these carts
 *  with each user session.
 */

public class ShoppingCart {
  private ArrayList itemsOrdered;

  /** Builds an empty shopping cart. */

  public ShoppingCart() {
    itemsOrdered = new ArrayList();
  }

  /** Returns List of ItemOrder entries giving
```

```
   *  Item and number ordered. Declared as List instead
   *  of ArrayList so that underlying implementation
   *  can be changed later.
   */

  public List getItemsOrdered() {
    return(itemsOrdered);
  }

  /** Looks through cart to see if it already contains
   *  an order entry corresponding to item ID. If it does,
   *  increments the number ordered. If not, looks up
   *  Item in catalog and adds an order entry for it.
   */

  public synchronized void addItem(String itemID) {
    ItemOrder order;
    for(int i=0; i<itemsOrdered.size(); i++) {
      order = (ItemOrder)itemsOrdered.get(i);
      if (order.getItemID().equals(itemID)) {
        order.incrementNumItems();
        return;
      }
    }
    ItemOrder newOrder = new ItemOrder(Catalog.getItem(itemID));
    itemsOrdered.add(newOrder);
  }

  /** Looks through cart to find order entry corresponding
   *  to item ID listed. If the designated number
   *  is positive, sets it. If designated number is 0
   *  (or, negative due to a user input error), deletes
   *  item from cart.
   */

  public synchronized void setNumOrdered(String itemID,
                                         int numOrdered) {
    ItemOrder order;
    for(int i=0; i<itemsOrdered.size(); i++) {
      order = (ItemOrder)itemsOrdered.get(i);
      if (order.getItemID().equals(itemID)) {
        if (numOrdered <= 0) {
          itemsOrdered.remove(i);
        } else {
          order.setNumItems(numOrdered);
        }
        return;
      }
    }
    ItemOrder newOrder =
      new ItemOrder(Catalog.getItem(itemID));
    itemsOrdered.add(newOrder);
  }
}
```

Listing 9.10 CatalogItem.java

```
package coreservlets;

/** Describes a catalog item for an online store. The itemID
 *  uniquely identifies the item, the short description
 *  gives brief info like the book title and author,
 *  the long description describes the item in a couple
 *  of sentences, and the cost gives the current per-item price.
```

```
 *  Both the short and long descriptions can contain HTML
 *  markup.
 */

public class CatalogItem {
  private String itemID;
  private String shortDescription;
  private String longDescription;
  private double cost;

  public CatalogItem(String itemID, String shortDescription,
                     String longDescription, double cost) {
    setItemID(itemID);
    setShortDescription(shortDescription);
    setLongDescription(longDescription);
    setCost(cost);
  }

  public String getItemID() {
    return(itemID);
  }

  protected void setItemID(String itemID) {
    this.itemID = itemID;
  }

  public String getShortDescription() {
    return(shortDescription);
  }

  protected void setShortDescription(String shortDescription) {
    this.shortDescription = shortDescription;
  }

  public String getLongDescription() {
    return(longDescription);
  }

  protected void setLongDescription(String longDescription) {
    this.longDescription = longDescription;
  }

  public double getCost() {
    return(cost);
  }

  protected void setCost(double cost) {
    this.cost = cost;
  }
}
```

Listing 9.11 ItemOrder.java

```
package coreservlets;

/** Associates a catalog Item with a specific order by
 *  keeping track of the number ordered and the total price.
 *  Also provides some convenience methods to get at the
 *  CatalogItem data without extracting the CatalogItem
 *  separately.
 */
public class ItemOrder {
  private CatalogItem item;
  private int numItems;
```

```
  public ItemOrder(CatalogItem item) {
    setItem(item);
    setNumItems(1);
  }

  public CatalogItem getItem() {
    return(item);
  }

  protected void setItem(CatalogItem item) {
    this.item = item;
  }

  public String getItemID() {
    return(getItem().getItemID());
  }

  public String getShortDescription() {
    return(getItem().getShortDescription());
  }
  public String getLongDescription() {
    return(getItem().getLongDescription());
  }

  public double getUnitCost() {
    return(getItem().getCost());
  }

  public int getNumItems() {
    return(numItems);
  }

  public void setNumItems(int n) {
    this.numItems = n;
  }

  public void incrementNumItems() {
    setNumItems(getNumItems() + 1);
  }

  public void cancelOrder() {
    setNumItems(0);
  }

  public double getTotalCost() {
    return(getNumItems() * getUnitCost());
  }
}
```

Listing 9.12 Catalog.java

```
package coreservlets;

/** A catalog that lists the items available in inventory. */

public class Catalog {
  // This would come from a database in real life.
  // We use a static table for ease of testing and deployment.
  // See JDBC chapters for info on using databases in
  // servlets and JSP pages.
  private static CatalogItem[] items =
    { new CatalogItem
      ("hall001",
```

```
 "<I>Core Servlets and JavaServer Pages " +
   "2nd Edition</I> (Volume 1)" +
   " by Marty Hall and Larry Brown",
 "The definitive reference on servlets " +
   "and JSP from Prentice Hall and \n" +
   "Sun Microsystems Press.<P>Nominated for " +
   "the Nobel Prize in Literature.",
 39.95),
new CatalogItem
  ("hall002",
   "<I>Core Web Programming, 2nd Edition</I> " +
     "by Marty Hall and Larry Brown",
   "One stop shopping for the Web programmer. " +
     "Topics include \n" +
     "<UL><LI>Thorough coverage of Java 2; " +
     "including Threads, Networking, Swing, \n" +
     "Java 2D, RMI, JDBC, and Collections\n" +
     "<LI>A fast introduction to HTML 4.01, " +
     "including frames, style sheets, and layers.\n" +
     "<LI>A fast introduction to HTTP 1.1, " +
     "servlets, and JavaServer Pages.\n" +
     "<LI>A quick overview of JavaScript 1.2\n" +
     "</UL>",
   49.99),
new CatalogItem
  ("lewis001",
   "<I>The Chronicles of Narnia</I> by C.S. Lewis",
     "The classic children's adventure pitting " +
     "Aslan the Great Lion and his followers\n" +
     "against the White Witch and the forces " +
     "of evil. Dragons, magicians, quests, \n" +
     "and talking animals wound around a deep " +
     "spiritual allegory. Series includes\n" +
     "<I>The Magician's Nephew</I>,\n" +
     "<I>The Lion, the Witch and the Wardrobe</I>,\n" +
     "<I>The Horse and His Boy</I>,\n" +
     "<I>Prince Caspian</I>,\n" +
     "<I>The Voyage of the Dawn Treader</I>,\n" +
     "<I>The Silver Chair</I>, and \n" +
     "<I>The Last Battle</I>.",
   19.95),
new CatalogItem
  ("alexander001",
   "<I>The Prydain Series</I> by Lloyd Alexander",
     "Humble pig-keeper Taran joins mighty " +
     "Lord Gwydion in his battle against\n" +
     "Arawn the Lord of Annuvin. Joined by " +
     "his loyal friends the beautiful princess\n" +
     "Eilonwy, wannabe bard Fflewddur Fflam," +
     "and furry half-man Gurgi, Taran discovers " +
     "courage, nobility, and other values along\n" +
     "the way. Series includes\n" +
     "<I>The Book of Three</I>,\n" +
     "<I>The Black Cauldron</I>,\n" +
     "<I>The Castle of Llyr</I>,\n" +
     "<I>Taran Wanderer</I>, and\n" +
     "<I>The High King</I>.",
   19.95),
new CatalogItem
  ("rowling001",
   "<I>The Harry Potter Series</I> by J.K. Rowling",
   "The first five of the popular stories " +
     "about wizard-in-training Harry Potter\n" +
     "topped both the adult and children's " +
```

```
            "best-seller lists. Series includes\n" +
            "<I>Harry Potter and the Sorcerer's Stone</I>,\n" +
            "<I>Harry Potter and the Chamber of Secrets</I>,\n" +
            "<I>Harry Potter and the " +
            "Prisoner of Azkaban</I>,\n" +
            "<I>Harry Potter and the Goblet of Fire</I>, and\n" +
            "<I>Harry Potter and the "+
            "Order of the Phoenix</I>.\n",
          59.95)
        };

  public static CatalogItem getItem(String itemID) {
    CatalogItem item;
    if (itemID == null) {
      return(null);
    }
    for(int i=0; i<items.length; i++) {
      item = items[i];
      if (itemID.equals(item.getItemID())) {
        return(item);
      }
    }
    return(null);
  }
}
```

[Team LiB]

PREVIOUS NEXT