The controller

Figure 12-10 lists the code for the controller of the Shopping Cart application. As usual, this code is stored in the index.php file. This file uses the functions in the cart.php file to add items to the cart and to update the items in the cart. In addition, it uses the add_item_view.php and cart_view.php files to display the Add Item page and the Cart page. These three files are discussed in the next three figures.

The controller begins by starting a session. Here, the first two statements use the session_set_cookie_params() function to create a session that persists for two weeks. As a result, if the user closes the browser and returns within two weeks, the user can continue his or her session.

After starting the session, the code checks whether the cart array is empty in the $_SESSION array. If so, this code creates an empty array to store the cart. Note that this code uses a key of 'cart12' to access the cart in the $_SESSION array. This key indicates that this cart is for chapter 12.

After initializing the cart, this code creates a multi-dimensional array of products. In a real-world application, this data would typically be read from a database. However, for the sake of simplicity, this application has hard-coded these values. If you want, you can add new products to the array by adding new lines that follow the same format.

After creating the table of products, the controller loads the cart.php file. This file contains functions for working with the cart array. These functions can add a new item to the cart or update the quantity of an existing item.

After including the cart functions, the controller gets the action to be performed based on the action parameter of the POST or GET request. If the action parameter hasn't been set, this code uses a default action of show_add_item.

After getting the action, the controller uses a switch statement to perform the action. If the action is add, this code retrieves the new product key and quantity from the POST request and passes it to the add_item() function, which adds the item to the cart. Then, this case displays the Cart page.

If the action is update, the code retrieves the array of new quantities from the POST request. Then, it uses a loop to check each quantity in the array. If the new quantity is different than the old quantity, the code calls the update_item() function to update the quantity for the item. Finally, it displays the Cart page.

If the action is show_cart, the code displays Cart page. This happens when the user clicks on the View Cart link on the Add Item page.

If the action is show_add_item, the code displays the Add Item page. This happens when the user clicks on the Add Item link on the Cart page.

If the action is empty_cart, the code uses the unset() function to remove the cart from the $_SESSION variable. Then, it displays the Cart page. This happens when the user clicks on the Empty Cart link on the Cart page. Note that this code doesn't end the session for the user. However, it does unset the only session variable that's used by this application. From the user's point of view, this effectively ends the session.

The index.php file

```
<?php
// Start session management with a persistent cookie
$lifetime = 60 * 60 * 24 * 14;    // 2 weeks in seconds
session_set_cookie_params($lifetime, '/');
session_start();

// Create a cart array if needed
if (empty($_SESSION['cart12'])) { $_SESSION['cart12'] = array(); }

// Create a table of products
$products = array();
$products['MMS-1754'] = array('name' => 'Flute', 'cost' => '149.50');
$products['MMS-6289'] = array('name' => 'Trumpet', 'cost' => '199.50');
$products['MMS-3408'] = array('name' => 'Clarinet', 'cost' => '299.50');

// Include cart functions
require_once('cart.php');

// Get the action to perform
$action = filter_input(INPUT_POST, 'action');
if ($action === NULL) {
    $action = filter_input(INPUT_GET, 'action');
    if ($action === NULL) {
        $action = 'show_add_item';
    }
}
// Add or update cart as needed
switch($action) {
    case 'add':
        $product_key = filter_input(INPUT_POST, 'productkey');
        $item_qty = filter_input(INPUT_POST, 'itemqty');
        add_item($product_key, $item_qty);
        include('cart_view.php');
        break;
    case 'update':
        $new_qty_list = filter_input(INPUT_POST, 'newqty', FILTER_DEFAULT,
                                     FILTER_REQUIRE_ARRAY);
        foreach($new_qty_list as $key => $qty) {
            if ($_SESSION['cart12'][$key]['qty'] != $qty) {
                update_item($key, $qty);
            }
        }
        include('cart_view.php');
        break;
    case 'show_cart':
        include('cart_view.php');
        break;
    case 'show_add_item':
        include('add_item_view.php');
        break;
    case 'empty_cart':
        unset($_SESSION['cart12']);
        include('cart_view.php');
        break;
}
```

The model

Figure 12-11 shows the code that models the behavior of the shopping cart. This code defines three functions. These functions let you add an item to the cart, update an item in the cart, and get the subtotal for the items in the cart.

The add_item() function takes an item key and quantity as its parameters and uses them to add the specified item to the cart. To start, this code gets access to the global products array. Then, it checks if the quantity is less than 1. If so, this code exits the function.

If the item is already in the cart, the add_item() function adds the existing quantity to the new quantity and calls the update_item() function to update the quantity for the item. Then, it exits the function.

If the item isn't already in the cart, the add_item() function gets the cost of the item from the products array and calculates the total for the item. Then, it creates an array that contains the item's name, cost, quantity, and total. Next, it stores the item array in the cart array using the item's key as the index.

The update_item() function also takes an item key and quantity as its parameters and uses them to update the item's quantity and total. To start, this code checks if the specified item is in the cart. If not, the update_item() function doesn't execute any code. However, if the item is in the cart, this function checks whether the quantity is less than or equal to zero. If so, this code uses the unset function to remove the item from the cart. Otherwise, this code stores the new quantity in the cart array. In addition, it calculates a new item total and stores it in the cart array.

The get_subtotal() function adds the totals for each item in the cart and returns it as a formatted number. To start, it sets the subtotal to zero. Then, it loops over each item in the cart and adds its total to the subtotal. Finally, it returns the subtotal as a number with two digits after the decimal.

The cart.php file

```
<?php
// Add an item to the cart
function add_item($key, $quantity) {
    global $products;
    if ($quantity < 1) return;

    // If item already exists in cart, update quantity
    if (isset($_SESSION['cart12'][$key])) {
        $quantity += $_SESSION['cart12'][$key]['qty'];
        update_item($key, $quantity);
        return;
    }

    // Add item
    $cost = $products[$key]['cost'];
    $total = $cost * $quantity;
    $item = array(
        'name' => $products[$key]['name'],
        'cost' => $cost,
        'qty'  => $quantity,
        'total' => $total
    );
    $_SESSION['cart12'][$key] = $item;
}

// Update an item in the cart
function update_item($key, $quantity) {
    $quantity = (int) $quantity;
    if (isset($_SESSION['cart12'][$key])) {
        if ($quantity <= 0) {
            unset($_SESSION['cart12'][$key]);
        } else {
            $_SESSION['cart12'][$key]['qty'] = $quantity;
            $total = $_SESSION['cart12'][$key]['cost'] *
                     $_SESSION['cart12'][$key]['qty'];
            $_SESSION['cart12'][$key]['total'] = $total;
        }
    }
}

// Get cart subtotal
function get_subtotal() {
    $subtotal = 0;
    foreach ($_SESSION['cart12'] as $item) {
        $subtotal += $item['total'];
    }
    $subtotal_f = number_format($subtotal, 2);
    return $subtotal_f;
```

2-11 The cart.php file of the Shopping Cart application

The Add Item view

Figure 12-12 shows the code for the Add Item page. This page displays a form that lets the user add an item to the cart by selecting a product and a quantity for that item. In addition, this page displays a link that lets the user view the cart without adding an item.

The form uses the POST method to submit the data back to the index.php controller for processing. This form includes a hidden field with a name of "action" and a value of "add" to indicate that the controller should add the item to the cart.

The first <select> tag has a name of "productkey". This tag lets the user select a product from a drop-down list. Within this tag, the PHP code uses a foreach loop to generate the <option> tags for the drop-down list. To start, this code loops through each item in the products array. At the beginning of the loop, this code formats the cost of each item as a number with two digits, and it uses the item name and formatted cost to generate the text to display for the item.

Within the loop, this code sets the value of the <option> tag to the key for the product. As a result, the key for the selected product is submitted to the controller. This code sets the text for the <option> tag to a string that includes the name and cost of the product, so this text is displayed in the drop-down list.

The second <select> tag has a name of "itemqty". This tag lets the user select the quantity from a drop-down list. This code uses a for loop to generate the <option> tags for the drop-down list. These tags display values from 1 to 10. Here, the code uses the index of the loop as both the value that's submitted to the control and the text that's displayed in the drop-down list.

At the end of the page, the View Cart link lets the user go to the Cart view page without adding an item. Here, the href attribute of the <a> tag links to the controller and submits an action parameter of show_cart. As a result, this parameter is passed to the controller as part of a GET request. That's why the controller checks both POST and GET requests for the action to perform.

The add_item_view.php file

```
<!DOCTYPE html>
<html>
<head>
    <title>My Guitar Shop</title>
    <link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
    <header>
        <h1>My Guitar Shop</h1>
    </header>
    <main>
        <h1>Add Item</h1>
        <form action="." method="post">
            <input type="hidden" name="action" value="add">

            <label>Name:</label>
            <select name="productkey">
            <?php foreach($products as $key => $product) :
                $cost = number_format($product['cost'], 2);
                $name = $product['name'];
                $item = $name . ' ($' . $cost . ')';
            ?>
                <option value="<?php echo $key; ?>">
                    <?php echo $item; ?>
                </option>
            <?php endforeach; ?>
            </select><br>

            <label>Quantity:</label>
            <select name="itemqty">
            <?php for($i = 1; $i <= 10; $i++) : ?>
                <option value="<?php echo $i; ?>">
                    <?php echo $i; ?>
                </option>
            <?php endfor; ?>
            </select><br>

            <label> </label>
            <input type="submit" value="Add Item">
        </form>
        <p><a href=".?action=show_cart">View Cart</a></p>
    </main>
</body>
</html>
```

ure 12-12 The add_item_view.php file of the Shopping Cart application

The Cart view

Figure 12-13 shows the code for the Cart page. Here, the first line of PHP code checks the cart array in session. If the cart doesn't exist or the number of items in the cart is 0, this code displays a message stating that there aren't any items in the cart. Otherwise, this code displays the contents of the cart in a table.

The cart table is contained within a form that lets the user update the quantity of each item in the cart. This form uses the POST method and submits the data to the index.php controller for processing. In addition, it contains a hidden field with a name of "action" and a value of "update". This tells the controller to update the quantities in the cart.

The cart table begins with a row that displays the headers for each of the columns in the table. Within this table, the <th> and <td> tags include class attributes. These attributes allow the CSS file for this application to control the alignment of the text in these columns.

This form uses a foreach loop to display each item in the cart. To start, the top of the loop formats the item's cost and total as a number with two digits after the decimal. Then, this code displays the item's name, cost, and total in a row. In addition, it displays the quantity. However, it displays the quantity within a text field so the user can update it. The tag for the text field has a name of "newqty" with the item key in brackets. This defines the newqty field as an array. In addition, this code defines the value of the text field as the item's quantity.

The cart_view.php file

Page 1

```
<!DOCTYPE html>
<html>
<head>
    <title>My Guitar Shop</title>
    <link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
    <header>
        <h1>My Guitar Shop</h1>
    </header>
    <main>
        <h1>Your Cart</h1>
        <?php if (empty($_SESSION['cart12']) ||
                  count($_SESSION['cart12']) == 0) : ?>
            <p>There are no items in your cart.</p>
        <?php else: ?>
        <form action="." method="post">
        <input type="hidden" name="action" value="update">
        <table>
            <tr id="cart_header">
                <th class="left">Item</th>
                <th class="right">Item Cost</th>
                <th class="right">Quantity</th>
                <th class="right">Item Total</th>
            </tr>

        <?php foreach( $_SESSION['cart12'] as $key => $item ) :
            $cost  = number_format($item['cost'],  2);
            $total = number_format($item['total'], 2);
        ?>
            <tr>
                <td>
                    <?php echo $item['name']; ?>
                </td>
                <td class="right">
                    $<?php echo $cost; ?>
                </td>
                <td class="right">
                    <input type="text" class="cart_qty"
                        name="newqty[<?php echo $key; ?>]"
                        value="<?php echo $item['qty']; ?>">
                </td>
                <td class="right">
                    $<?php echo $total; ?>
                </td>
            </tr>
        <?php endforeach; ?>
```

After the item rows generated by the foreach loop, the table displays the cart subtotal in one row and a submit button in another. In addition, it displays instructions that explain how to update the items in the cart and how to remove items from the cart.

At the end of the page, the Add Item link lets the user return to the Add Item page. To do that, it sends an action of show_add_item to the controller. In addition, the Empty Cart link lets the user remove all items form the cart. To do that, it sends an action of empty_cart to the controller.

If you prefer, you can use buttons instead of the Add Item, View Cart, and Empty Cart links. To do that, you can code a form for each button instead of coding a link. Then, you can include a hidden field for the action parameter, and you can submit the form using either the GET or POST method. However, both techniques for passing an action are acceptable and there are advantages and disadvantages to each approach.

The cart_view.php file

Page 2

```
                <tr id="cart_footer">
                    <td colspan="3"><b>Subtotal</b></td>
                    <td>$<?php echo get_subtotal(); ?></td>
                </tr>
                <tr>
                    <td colspan="4" class="right">
                        <input type="submit" value="Update Cart">
                    </td>
                </tr>
            </table>
            <p>Click "Update Cart" to update quantities in your
                cart. Enter a quantity of 0 to remove an item.
            </p>
            </form>
        <?php endif; ?>
        <p><a href=".?action=show_add_item">Add Item</a></p>
        <p><a href=".?action=empty_cart">Empty Cart</a></p>
    </main>
</body>
</html>
```