Server-side Web Programming

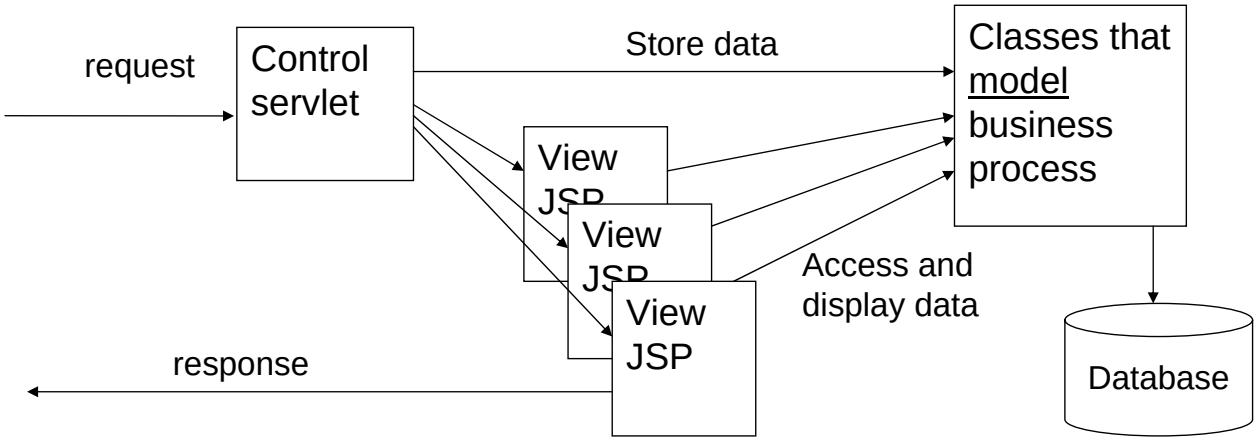
Lecture 9: **Shopping Carts and the Model-View-Control Architecture**

Modeling the Business Process

- What information should a session deal with?
  - What information must be gathered via forms?
    - Items to purchase
    - Customer information, etc.
  - What information must be displayed to the user in response pages?
    - Shopping cart contents, etc.
  - What information must be stored long term?
    - Throughout session
    - Longer (in databases, etc.)
- Model of business process
  - Defined by organization you develop site for

Model-View-Control Architecture

- Model = software classes that store/manipulate information gathered in session
  - Usually separate from servlets/JSPs
  - Servlets/JSPs interact with those classes
  - Often interact with databases



Model Classes and Session

- Bad design:
  Store all session information as separate attributes
  - May be dozens of attributes
  - Servlet/JSP responsible for manipulating each individually

All session data

…	…
Session ID = fieh4K39Rdk	Session data: name = "Fred"; email = "fred@aolrock"
…	…

Model Classes and Session

- Better design:
  Create classes for blocks of information
  - Store objects as session attributes
  - Use class methods to store/access/manipulate data in object

All session data

…	…
Session ID = fieh4K39Rdk	Session data customerInfo: Customer object cart: Cart object
…	…

Model Class Properties

- State variables for all information stored
- **`set`*`Variable`*** methods for each variable
  - Takes value as parameter and stores it in state variable
  - Usually called by servlet to store data from parameters
- **`get`*`Variable`*** methods for each variable
  - Returns current state variable
  - Usually called by JSP to display data stored earlier

These model classes often created by other programmers

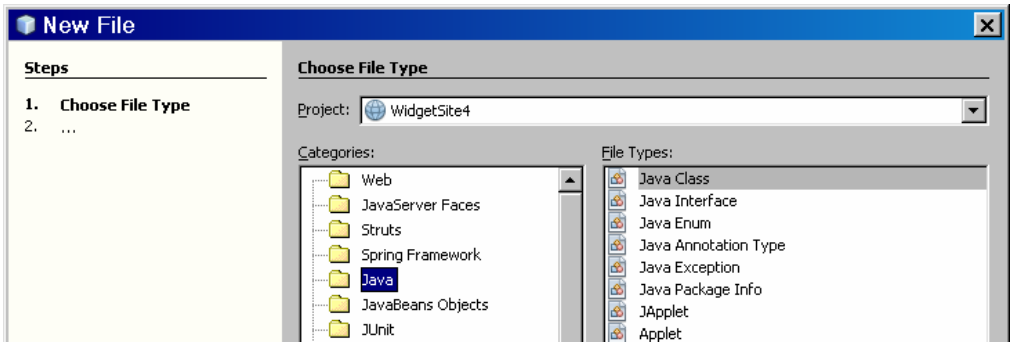
- Business logic specialists create model
- Server programming specialists create servlets/JSPs

Example Customer Class

```
1   package Model;
2
3   import java.util.regex.*;
4   public class Customer {
5       private String name;
6       private String email;
7       public Customer() {}
8       public void setName(String n) {name = n;}
9       public void setEmail(String e) {email = e;}
10      public String getName() {return name;}
11      public String getEmail() {return email;}
12  }
```

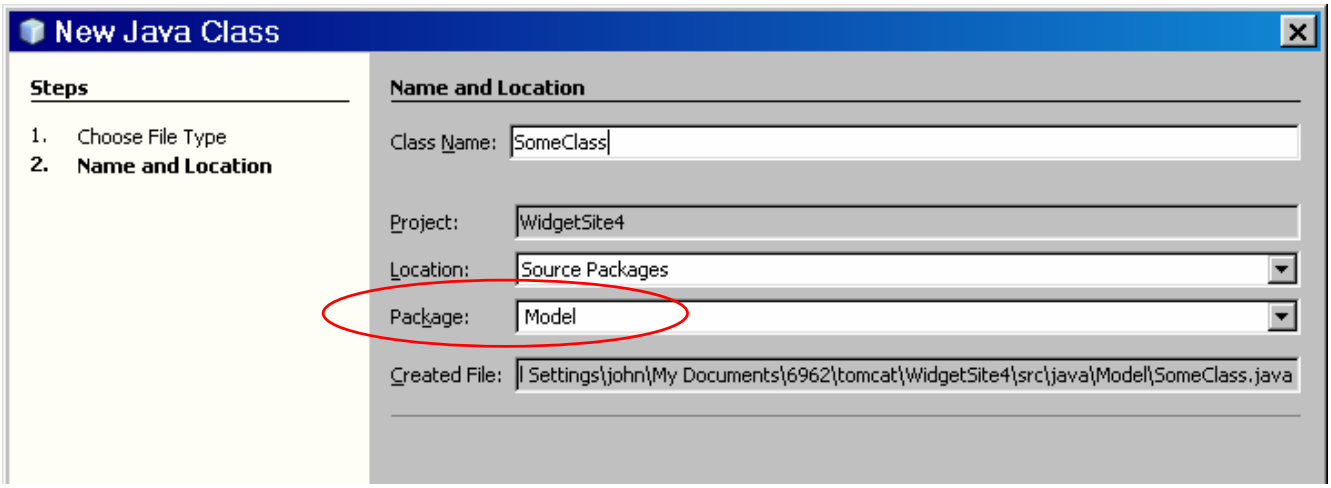
Creating Support Classes

- File → New File
- Choose category Java, type class



Creating Support Classes

- As before, enter name
- Tomcat requires support classes to be in a package
  - "Library" of related classes
  - Can define one at this point (added to drop down in future)



Using Model Classes

- Must include package name at top of file
  `package packagename;`
- Other classes that use class must import the package
  - Servlets use import statement
  - JSPs use <@ page import tag

```
7  import java.io.*;
8  import javax.servlet.*;
9  import javax.servlet.http.*;
10 import Model.*;
11
12 public class CheckQuantity extends HttpServlet {
13
14     /**
15     * Processes requests for both HTTP <code>GET</cc
16     * @param request servlet request
17     * @param response servlet response
18     */
19     protected void processRequest(HttpServletRequest
20     throws ServletException, IOException {
21         HttpSession session = request.getSession();
```

```
13 <body>
14   <%@ page import="Model.*"  %>
15   <%
16     Customer customer = (Customer)session.getA
17     String name = customer.getName();
18     String email = customer.getEmail();
19     Order order = (Order)session.getAttribute(
20     int quantity = order.getQuantity();
21     double cost = order.getCost();
22     //int quantity = 1;
```

Using Model Classes

In servlet:

- Construct a new instance of the model object
- Use its ***set*** methods to store parameters from form
- Store the object as a session attribute

```
26
27          String name = request.getParameter("customerName");
28          String email = request.getParameter("customerEmail");
29
30          // Construct the business model object
31          Customer customer = new Customer();
32          customer.setName(name);
33          customer.setEmail(email);
34
35          session.setAttribute("customer", customer);
36
```

Using Model Classes

In JSP:

- Retrieve the object from the session attributes
  - Must cast back to its original type
- Use its ***get*** methods to retrieve data
- Display the data on the page

```
14    <%@ page import="Model.*"  %>
15    <%
16      Customer customer = (Customer)session.getAttribute("customer");
17      String name = customer.getName();
18      String email = customer.getEmail();
```

Business Model Objects

Key idea:

- Methods in model objects should implement business model
  - Methods to validate values
  - Methods to perform computations
  - Methods to store information in database

Goal:

- Separate web programming and business knowledge as much as possible

Business Model Example

- **Order** class
  - Methods to set/get quantity
  - Method to get total cost of order
    - Computed from quantity here instead of in JSP/servlet
  - Method to check whether quantity valid in terms of business model
    - Servlet makes sure quantity is a number
    - Business model class makes sure quantity is at least 1

Order Class

```
1   package Model;
2
3   public class Order {
4       private int quantity;
5       public Order() {}
6       public void setQuantity(int q) {quantity = q;}
7       public int getQuantity() {return quantity;}
8
9       // COmputes and returns the cost of this many widgets
10      public double getCost() {return quantity * 9.95;}
11
12      // Static method to check whether a quantity is valid
13      // Call it using the name of the class (Order.isValidQuantity)
14      public static boolean isValidQuantity(int q) {
15          if (q > 0) {return true;}
16          else return false;
17      }
18  }
```

Validation in Servlet

```
24          String quantity = request.getParameter("quantity");
25          // This is a nested structure which determines the type of error
26          // replated to the quantity, and sets the appropriate error message.
27          int quantityNumber = 0;
28          if (quantity.equals("")) {
29              url = "/getQuantity.jsp";
30              request.setAttribute("quantityError", "You must enter a quantity");
31          }
32          else {
33            //Parse quantity entered (exception if not an integer)
34            try {
35               quantityNumber = Integer.parseInt(quantity);
36               if (!Order.isValidQuantity(quantityNumber)) {
37                   request.setAttribute("quantityError", "Quantity not legal value");
38                   url = "/getQuantity.jsp";
39               }
40               else {
41                   // Construct the business model object if the quantity is ok
42                   Order order = new Order();
43                   order.setQuantity(quantityNumber);
44                   // Store as session attribute
45                   session.setAttribute("order", order);
46               }
47            }
48            catch (NumberFormatException ex) { // Can't parse quantity
49                request.setAttribute("quantityError", "Please enter a number for quantity");
50              url = "/getQuantity.jsp";
51            }
52          }
```

Getting Cost in JSP

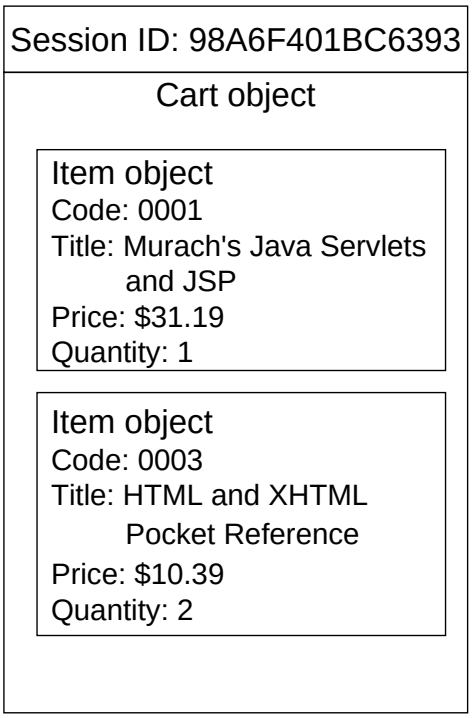
```
13  <body>
14    <%@ page import="Model.*"  %>
15    <%
16      Customer customer = (Customer)session.getAttribute("customer");
17      String name = customer.getName();
18      String email = customer.getEmail();
19      Order order = (Order)session.getAttribute("order");
20      int quantity = order.getQuantity();
21      double cost = order.getCost();
22    %>
23
24    <h2>Order Confirmation</h2>
25
26    <p>
27      Thank you for your order of <%= quantity %> widgets, <%= name %>.
28    </p>
29    <p>
30        Your bill will be $<%= cost %>.
31    </p>
```

Shopping Carts

- Usually list of items
  - List = **`Vector`** or **`ArrayList`** type in Java
  - Has methods to **`add`** new element to end, **`get`** i^{th} element, and **`remove`** i^{th} element
- Each list element is business model object
  - Has fields for all relevant data about item purchased
    - Set and get methods for each
    - One field should be unique identifier (key field)
    - Some fields populated by database lookup
  - May have way to set quantity purchased
    - Not all models have quantity – course registration, for example

Example “Bookstore” Cart

- **`Cart`**: list of **`Item`** objects
- Each customer has own **`Cart`** object stored in their session
- Each **`Item`** has a unique **`code`**
  - Given a **`code`**, should look up other fields in database
  - **`title`** and **`price`**
  - Not implemented yet!



“Bookstore” Cart Methods

Usual methods:

- **`void addItem(code)`**
  - add a new item to the cart with the given ID
- **`Item lookup(code)`**
  - Lookup and return the Item in the list with the given ID
- **`void removeItem(code)`**
  - Lookup and delete the Item in the list with the given ID
- **`void getItem(int)`**
  - Return the i^{th} Item in the list
- **`void size()`**
  - Return the number of Items in the list

Used together to loop through and show all Items on “shopping cart” page

"Bookstore" Cart Methods

```
24      // Utility method to look up an item given its index
25      public Item lookup(String code) {
26        for (int i = 0; i < items.size(); i++) {
27          Item item = (Item)items.get(i);
28          if (code.equals(item.getCode())) return item;
29          }
30        return null;
31        }
32
33      // Get the ith item in the list
34      public Item getItem(int i) {
35          return (Item)items.get(i);
36      }
37
38      // Return the size of the list (useful if need to
39      // loop through it to display cart)
40      public int getSize() {
41        return items.size();
42        }
43
44      // Add a new item to the end of the cart. Constructs a
45      // new item
46      public void addItem(String c, int q) {
47        items.add(new Item(c, q));
48        }
49
50      // Look up the item in the list with the given code,
51      // and remove it from the list.
52      public void removeItem(String code) {
53        for (int i = 0; i < items.size(); i++) {
54          Item item = (Item)items.get(i);
55          if (code.equals(item.getCode())) items.removeElementAt(i);
56          }
57        }
```

"Bookstore" Item Methods

Usual methods:

- **`Item(code)`**
  - construct a new item
  - Look up rest of fields in database based on code
- **`String get_____()`**
  - Return the value of the given field
  - For bookstore: code, title, price, cost
- **`void setQuantity(int)`**
  `int getQuantity()`
  - Since can order multiple copies, need ways to change quantity

“Bookstore” Item Methods

```
12      public class Item {
13
14        private String code, title;
15        private int quantity;
16        private double price;
17
18        public Item(String c, int q) {
19          code = c;
20          // This is the point at which we would go out and look up the
21          // information for the item in a database. For now, we'll just
22          // hardwire it in.
23           price = 0; title ="";
24           quantity = q;
25           if (c.equals("0001")) {price = 31.19; title = "Murach's Java Servlets and
26           if (c.equals("0002")) {price = 33.08; title = "Murach's ASP.NET 2.0 Web F
27           if (c.equals("0003")) {price = 10.39; title = "HTML and XHTML Pocket Refe
28          }
29
30        public String getCode() {return code;}
31        public String getTitle() {return title;}
32        public double getPrice() {return price;}
33        public double getCost() {return price * quantity;}
34        public int getQuantity() {return quantity;}
35
36        // Can update the quantity of item in cart
37        public void setQuantity(int q) {quantity = q;}
```

Displaying Cart Contents

- Get cart from session

```
Cart cart = (Cart) session.getAttribute("cart");
```

- Inside of a table:
  - Get number of items in cart
  - Use loop to get each item in sequence
    - Number of items = length of loop
    - Get the i^{th} Item object from the Cart
  - Get desired fields of that item
  - Create new table row showing those fields

```
18  <h3>Your shopping cart:</h3>
19  <table border="1" cellpadding="5">
20    <tr>
21      <th>Title</th><th>Price</th><th>Quantity</th><th>Amount</th>
22    </tr>
23
24  <%
25     int numberOfItems = cart.getSize();
26     for (int i = 0; i<numberOfItems; i++) {            // For each item
27       Item item = cart.getItem(i);          // Get the  ith item
28       String productCode = item.getCode();  // Get the product code, etc.
29       String title = item.getTitle();
30       int quantity = item.getQuantity();
31       double price = item.getPrice();
32       double cost = item.getCost();
33  %>
34
35    <tr valign="top">
36
37      <td><%= title %></td>
38      <td>$<%= price %></td>
39      <td><%= quantity %></td>
40      <td>$<%= cost %>
41
42  <% } %>
43
44  </table>
```

Inside table

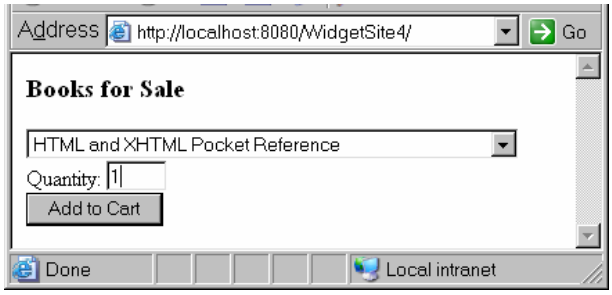
Size of loop

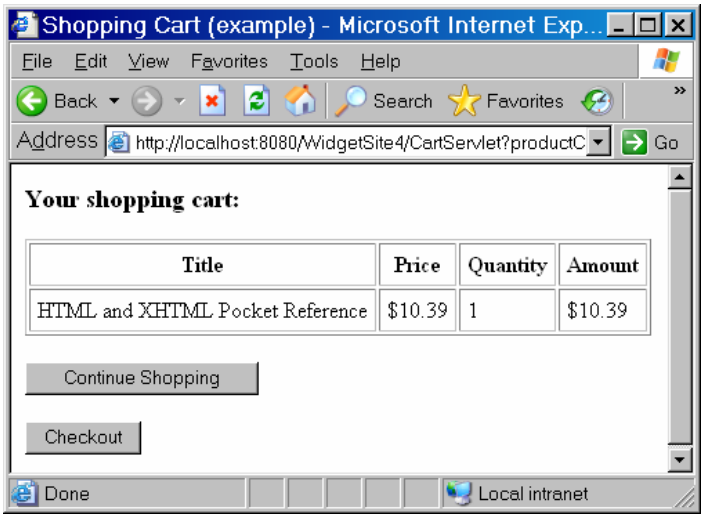
Get ith item object

Get its fields

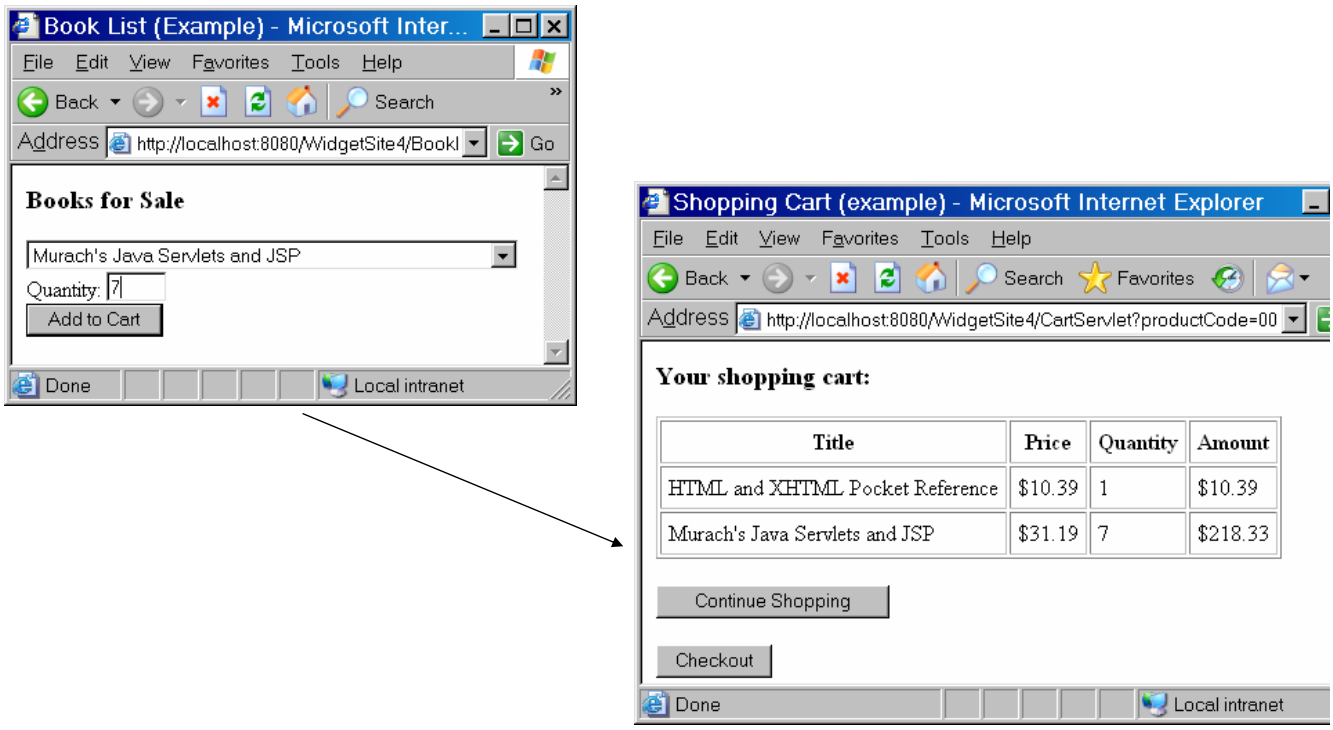
Inside loop, create new table row showing those fields

Displaying Cart Contents





Displaying Cart Contents



Adding to the Cart in a Servlet

- Get current Cart object from session using getAttribute
  - If null, no Cart exists yet
  - In that case, construct one

```
HttpSession session = request.getSession();
// Get the client's current cart. If none exists, constr
Cart cart = (Cart) session.getAttribute("cart");
if (cart == null){
  cart = new Cart();
  session.setAttribute("cart", cart);
  }
```

Adding to the Cart in a Servlet

- Get data from request and pass to Cart
  - Cart will construct and store a new Item
  - Will need to validate request first
- Will need to check whether item already in Cart
  - Need to avoid duplicate entries in Cart
  - Business model defines how handled
    - Error message
    - Change quantity,
    - Add to quantity, etc.

Adding to the Cart in a Servlet

```
// Find out whether this item is already in the cart.
Item item = cart.lookup(productCode);

// If so, and the quantity > 0, add to the quantity
if (item != null && quantity > 0) {
  int oldQuantity = item.getQuantity();
  item.setQuantity(quantity+oldQuantity);
  }

// Otherwise, add a new item for this product
if (item == null && quantity > 0) {
  cart.addItem(productCode, quantity);
  }
```

Adding to the Cart in a Servlet

- Can use servlet to modify Cart in other ways
  - Example: Remove from Cart if quantity = 0

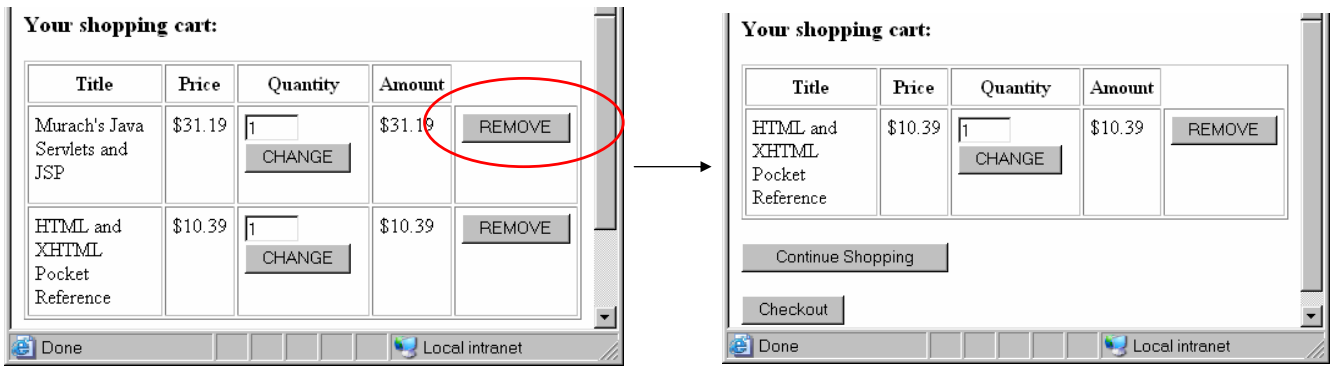
```
// If the quantity is set to zero, remove the item.
if (item != null && quantity <= 0) {
    cart.removeItem(productCode);
    }
```

- Store modified Cart to session using setAttribute

```
// Store the new copy of the cart in the session
session.setAttribute("cart", cart);
```

Embedded Forms in Cart Pages

- Often embed buttons and other form elements into the rows of a Cart page
  - Example: simple REMOVE button and quantity updates



Embedded Forms in Cart Pages

- Must nest entire form inside a **<td>** element
  - Action = servlet to handle desired change to the cart
  - Contains SUBMIT button to send request
  - Contains form element with data to submit with request

```
37      <td><%= title %></td>
38      <td>$<%= price %></td>
39      <td><%= quantity %></td>
40      <td>$<%= cost%></td>
41      <td>
42      <form action="RemoveServlet">
43          <input type="hidden" name="productCode" value="<%= productCode %>">
44          <input type="submit" value="REMOVE" />
45      </form>
46      </td>
```

Simple Removal Servlet

```
22      */
23      protected void processRequest(HttpServletRequest request, HttpServletRespons
24      throws ServletException, IOException {
25          HttpSession session = request.getSession();
26          // Get the client's current cart. If none exists, construct a new one.
27          Cart cart = (Cart) session.getAttribute("cart");
28          if (cart == null){
29            cart = new Cart();
30            session.setAttribute("cart", cart);
31            }
32          String productCode = request.getParameter("productCode");
33
34          // Check whether it is in the Cart
35          Item item = cart.lookup(productCode);
36          // If so, remove the item.
37          if (item != null) {
38              cart.removeItem(productCode);
39              }
40          // Store the new copy of the cart in the session
41          session.setAttribute("cart", cart);
```

Hidden Form Elements

- Must submit product code for remove to work
- Product code not displayed on page inside a form element
  - Common for most ecommerce pages

Can use hidden form element

- Not shown by browser
- Can store product code or other information that we need to send to server

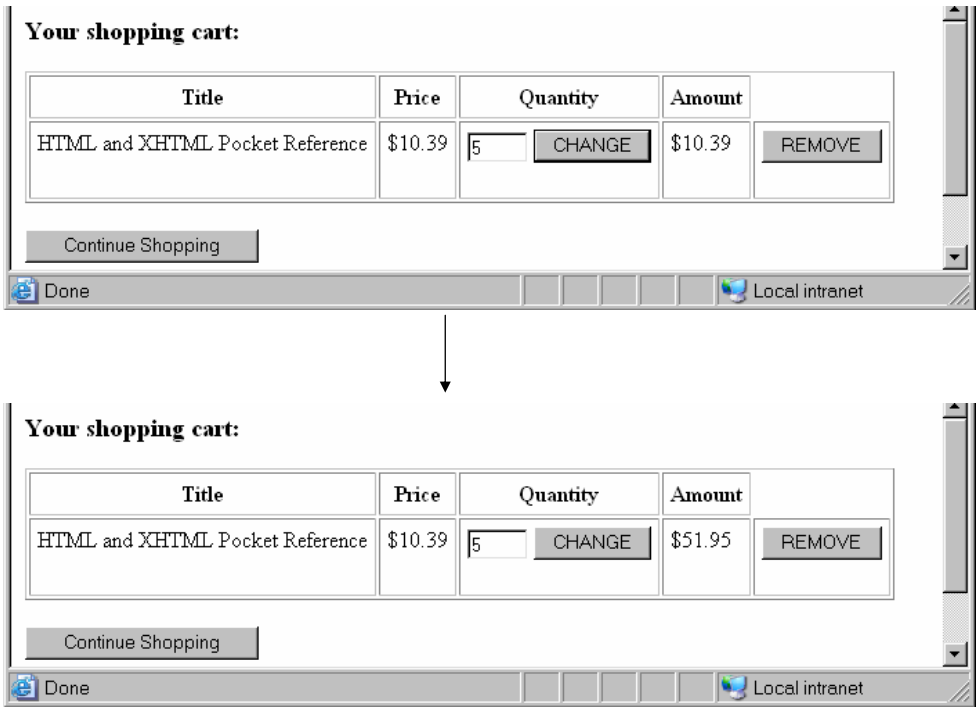
```
<input type="hidden"
       name="parametername"
       value="<%= product code %>"
```

Hidden Form Elements

```
26      for (int i = 0; i<numberOfItems; i++) {            // For each item
27        Item item = cart.getItem(i);          // Get the  ith item
28        String productCode = item.getCode();  // Get the product code, etc.
29        String title = item.getTitle();
30        int quantity = item.getQuantity();
31        double price = item.getPrice();
32        double cost = item.getCost();
33  %>
34
35    <tr valign="top">
36
37      <td><%= title %></td>
38      <td>$<%= price %></td>
39      <td><%= quantity %></td>
40      <td>$<%= cost%></td>
41      <td>
42      <form action="RemoveServlet">
43          <input type="hidden" name="productCode" value="<%= productCode %>">
44          <input type="submit" value="REMOVE" />
45      </form>
46      </td>
```

productCode read in from Cart in session and stored in hidden element inside removal form

Quantity Update Example



Quantity Update Example

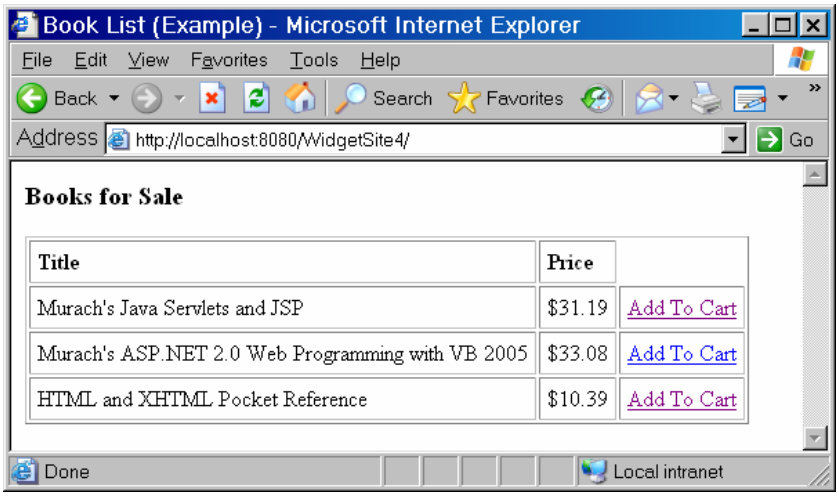
```
27      Item item = cart.getItem(i);          // Get the  ith item
28      String productCode = item.getCode();  // Get the product code, etc.
29      String title = item.getTitle();
30      int quantity = item.getQuantity();
31      double price = item.getPrice();
32      double cost = item.getCost();
33  %>
34
35    <tr valign="top">
36
37      <td><%= title %></td>
38      <td>$<%= price %></td>
39      <td>
40          <form action="QuantityServlet">
41          <input type="hidden" name="productCode" value="<%= productCode %>">
42          <input type="text" name="newquantity"  size="3" value="<%= quantity %>">
43          <input type="submit"value="CHANGE">
44          </form>
45      </td>
```

Pass productCode as hidden field so servlet knows which Item to change

Also pass new quantity entered by user

Passing Data using Links

- Many web sites use html links instead of forms



- Question: How can form information (such as product code) be passed if no form is used?

Passing Data using Links

- Can append "form data" directly to URL in link
  - Result similar to "get" method in form
- Syntax:

```
<a href = "url of servlet?name=value&name=value…">
```

Submit request to the servlet

The '?' indicates form parameters

Each passed as a name=value pair separated by '&'

  - Note: will need to use `response.encodeURL` to insure this works if cookies not enabled

```
<a href =
  "<%=response.encodeURL('servleturl?name=value&…')%>">
```

Passing Data using Links

```
24    <tr valign="top">
25      <td>Murach's Java Servlets and JSP</td>
26      <td>$31.19</td>
27      <td>
28        <a href='<%= response.encodeURL("CartServlet?productCode=0001") %>'>
29          Add To Cart
30        </a>
31      </td>
32    </tr>
33
34    <tr valign="top">
35      <td>Murach's ASP.NET 2.0 Web Programming with VB 2005</td>
36      <td>$33.08</td>
37      <td>
38        <a href='<%= response.encodeURL("CartServlet?productCode=0002") %>'>
39          Add To Cart
40        </a>
41      </td>
42     </tr>
43
44    <tr valign="top">
45      <td>HTML and XHTML Pocket Reference</td>
46      <td><p>$10.39</td>
47      <td>
48        <a href='<%= response.encodeURL("CartServlet?productCode=0003") %>'>
49          Add To Cart
50        </a>
```