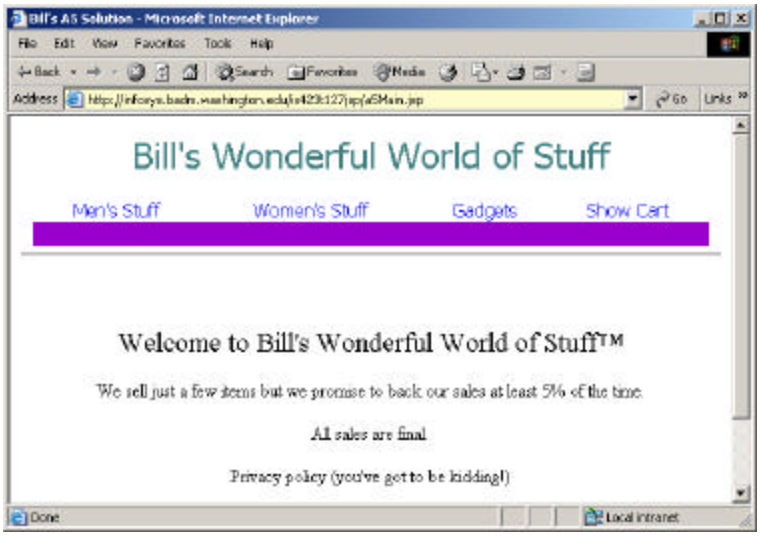
Objective

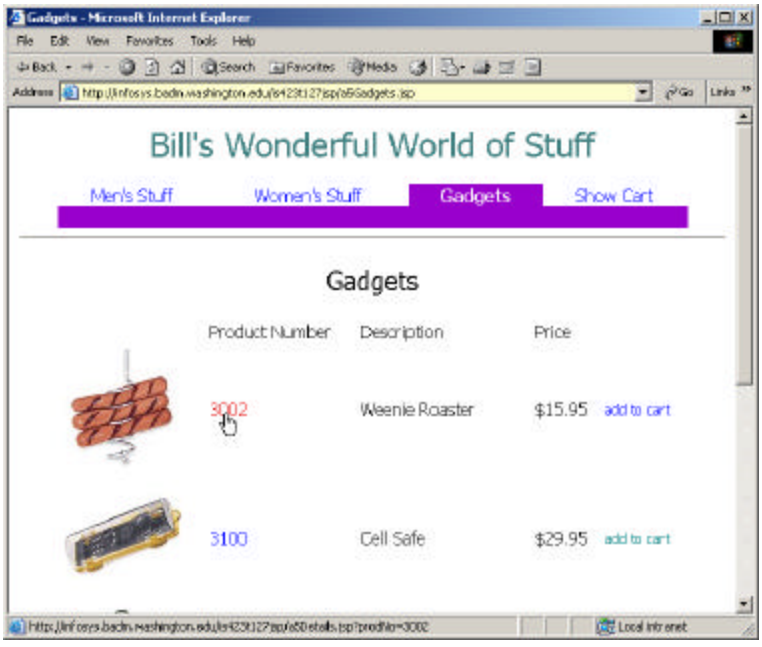
This assignment is designed to extend your understanding of web applications using java server pages and java beans. It also introduces you to the use of server-side database access.

Problem description

This problem simulates a company that sells items from its web site. The series of screen shots that follow show the major functionality of the system. The series begins with a start page that introduces the company and lists (at the top) the main categories that can be accessed.

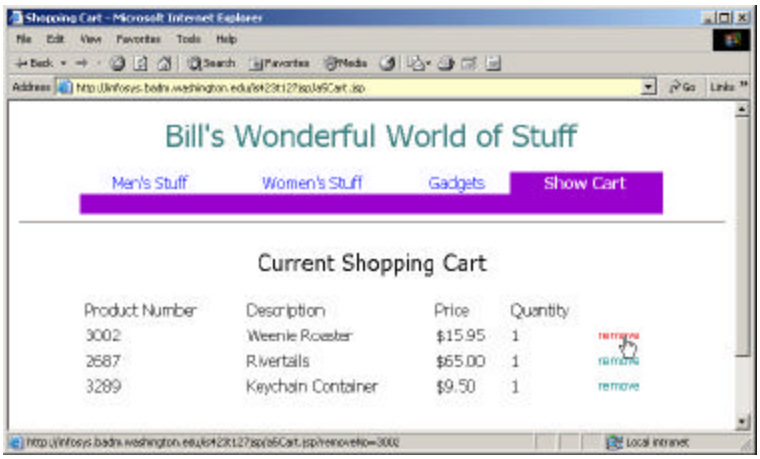


If the user clicks on a main category such as “Gadgets”, a screen will be displayed that shows a list and picture of all the gadgets in the database.



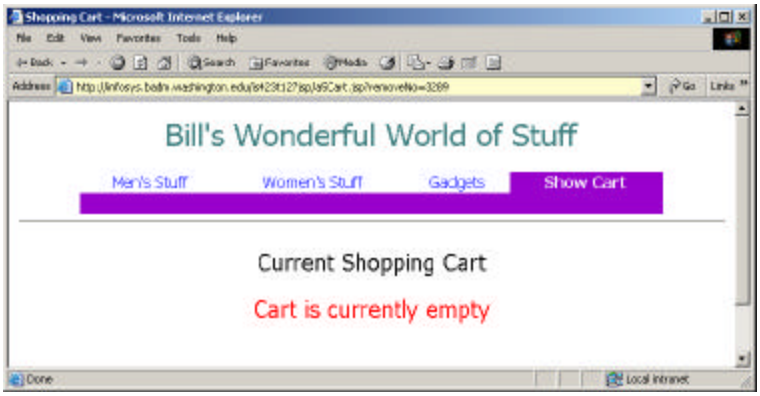
Note that all the information displayed is taken from the database and image files stored on the server. If the user clicks on the product number, additional information on the product will be shown.

Note that both the original page that lists all products as well as each product detail page includes the ability to add the item to the user's shopping cart.



The shopping cart page shows all the items and quantities of items the user has added to the cart. If an item is added multiple times, then the "Quantity" field represents this information.

Note also that the user has the option of removing a product from the shopping cart by clicking on the "remove" button. This should remove the associated item even if the quantity field is greater than one. If the cart become empty, this should be indicated as shown in the next screen shot.



You should be able to use the sample solution available on the net to verify the functionality of the problem.

Local versus Server Testing

Your final solution should be placed on the "infosys" server as you did in assignment 4. You can create and test the application within JBuilder if you prefer. However, since the solution involves a database and a folder of images, testing locally requires that you perform a more complex setup than in assignment 4.

To locally develop and test, you need all your .jsp files stored in the project's "defaultroot" directory (just like all JBuilder JSP projects). You need to store the images folder in the "defaultroot" directory also.

You can place the products.mdb in any location (within the project or the "defaultroot" directory would make sense). However, you must register the database on the local machine with a System DSN. You need to use the Data Sources (ODBC) control panel tool (found in the Administrative Tools folder in Windows 2000). Instructions for setting up an ODBC data source are available on the course web site

To make sure that my local machine was set up correctly, I wrote a simple jsp page as shown below that displayed selected fields from the database.

```
<%@ page import = "java.sql.*" %>
<html>
<head>
<title>
SimpleResultSet
</title>
</head>
<body>
<h2>
Viewing Records in the Product Table
</h2>
<%
```

```
Connection databaseConnection = null;
ResultSet rsStudents = null;
try {
  // Load the driver class
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  // This defines the data source for the driver
  String sourceURL = new String("jdbc:odbc:products");
  // Create connection through the DriverManager
  databaseConnection =
    DriverManager.getConnection(sourceURL);
  Statement myStatement =
    databaseConnection.createStatement();
  rsStudents =
    myStatement.executeQuery("select * from prodList");
  // Place results into table
  %>
  <table border="1" bgcolor="silver" cellpadding="2">
    <tr>  <!-- create the table heading -->
      <td><b>prodNo</b></td>
      <td><b>cat</b></td>
      <td><b>desc</b></td>
      <td><b>smURL</b></td>
      <td><b>price</b></td>
      <td><b>story</b></td>
    </tr>
  <%
  while(rsStudents.next()){
%>
    <tr>
      <td align="center"> <%=rsStudents.getString("prodNo")%> </td>
      <td align="center"> <%=rsStudents.getString("cat")%> </td>
      <td align="center"> <%=rsStudents.getString("desc")%> </td>
      <td align="center"> <%=rsStudents.getString("smURL")%> </td>
      <td align="center"> <%=rsStudents.getString("price")%> </td>
      <td align="center"> <%=rsStudents.getString("story")%> </td>
    </tr>
<%
  }
%> </table> <%
}
  catch(ClassNotFoundException cnfe){
    //JOptionPane.showMessageDialog(this, cnfe.toString());
  }
  catch(SQLException sqle) {
    //JOptionPane.showMessageDialog(this, sqle.toString());
  }
  finally{
    rsStudents.close();
    databaseConnection.close();
  }
%>
</body>
</html>
```

A JBuilder project with this code in it is available on the course web site. I also wrote a test program to run on my server account to make sure that things where set up correctly there. The only difference in the two text programs is the name of the driver class and its URL on the server. The appropriate lines are:

Code for local test:

```
try {
  // Load the driver class
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  // This defines the data source for the driver
  String sourceURL = new String("jdbc:odbc:products");
```

Code for the remote (server) test:

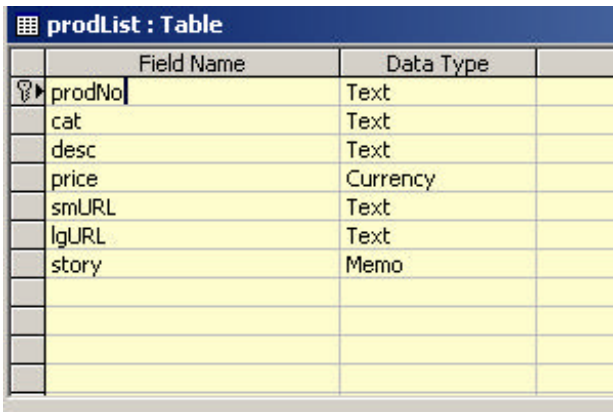
```
try {
  // Load the driver class
  Class.forName("easysoft.sql.jobDriver");
  // This defines the data source for the driver
  String sourceURL = new String("jdbc:easysoft://infosys.badm.washington.edu:8831/products");
```

A JBuilder project with this code in it is also available on the course web site. For the server solution, you do not need to worry about a copy of either the database or the folder of product images. These are already set up on the server. So the only difference between your local solution and the solution you post to the web site will be the two statements shown above.

If you plan to develop and test locally, I recommend you get the project fully debugged locally, make a copy of the project folder that will be become the basis for the server solution, and then finally edit the copy in the server solution to reflect the correct driver class name and URL.

The Database

The database, named products.mdb consists of a single table named prodList and seven fields as shown next:



A sample few rows from the table follow:

prodList : Table

prodNo	cat	desc	price	smURL	lgURL	
1212	Men's Apparel	Windy Point Shell	$165.00	images/shell.jpg	images/shellLg.jpg	You can't find better value in all-a
1342	Men's Apparel	Fleece jacket	$89.50	images/fleece.jpg	images/fleeceLg.jpg	Reversible jacket gives you all-we
1486	Men's Apparel	Odor Free Tee	$29.50	images/tee.jpg	images/teeLg.jpg	Our CoolMax® Odor-Free Tee eli

Note several important things. First, each product includes the name of a small and a large image in fields named “smURL” and “lgURL”. These references assume that the pictures are stored in a folder named “images” that is in the same directory as the jsp page resides (as discussed previously). For the database on the server, these URL fields are different and instead of pointing to a relative URL, they point to a physical URL. This means that for your local testing, you need to have the “images” folder in your “defaultroot” folder but for the server solution, you do not need to worry about where the image files are actually located.

Second, although you cannot see this from the image above, there are three category values for the “cat” field. These are shown next:

prodNo	cat	desc	price	smURL
2932	Women's Apparel	Bubble Skirt	$64.00	images/bubble.jpg
1840	Men's Apparel	Cargo Shorts	$35.00	images/cargo.jpg
3100	Gadgets	Cell Safe	$29.95	images/cell.jpg

You will need to query the database to find records that match each category. Because two of the category values include an apostrophe, you should use the SQL “like” operator to find matches (because you cannot put apostrophes in an SQL string literal. For example, you might say:

```
select * from prodList where cat like 'Men%'
```

This selects all records that start with the three letters “Men”. The “%” means the remaining characters do not matter.

Finally, the “price” field can be accessed within java using the getDecimal() method and the “story” field can be accessed using getString() method. In the later case, some of the special characters (such as ‘ and ®) will not display correctly. Do not worry about this small problem.

The Cart

You are free to define your cart using any of the technologies we have discussed. My solution used a java bean to solve the problem. I created a class named Product that included the following public members:

```
public Product(String prodNo)
public String getProdNo() {
public int getQty() {
public void addOneToQty()
```

The bean itself keeps track of products in the cart using a Hashtable. The bean includes the following public members:

public Cart()
: Instantiates the Hashtable

public void addProduct(String prodNo)
: Creates a new Product object and either stores it in the Hashtable (assuming it is not already there) or adds one to the quantity (assuming the product is already in the hashtable).

public void removeProduct(String prodNo)
: Removes the product object from the Hashtable.

public Hashtable getAllProducts()
: Returns a clone of the Hashtable.

Helpful Hints

When you refer to a field in a resultSet for a particular record, that field cannot be accessed a second time unless you move to another record. If you need the value of a field more than once for the same record, store it first in a variable and then use the variable multiple times.

If you expect that your select statement will only retrieve a single record, you still need to execute the resultSet's next() method to get to the single record.

Be sure to close your resultSet as soon as you finish using it. You cannot create a new resultSet from the same connection if another resultSet is already open.

Use the out.println() method in your catch blocks to help you see what is going wrong if an exception is thrown. For example:

```
catch(SQLException sqle) {
  out.println(sqle.toString());
}
```

As you run the test solution, watch the URL carefully both as you go to a new page as well as moving the cursor over a link (such as a cart "add" or "remove" link). This will help you see how the pages are communicating.

Finally you will need to emulate the trick we have seen in class where a page calls itself and can react differently depending on the URL parameters that it is being passed.

Everything you need to solve this problem has been discussed in class and seen in the numerous examples that have been handed out. Look them over carefully for ideas.

Directory Setup

You need to FTP your java server pages and your JavaBean class package to the server for final testing and grading. The location of these files is the same as you used for assignment 4. In addition, if your solution includes any other class files (such as the Product class my solution used), these should go into the same classes folder that the beans go into on the server.

To Turn In

Send an email to is423wb@u.washington.edu. In the subject heading put your team account number:

Subject: is423t127

In the body of the email include all your team member names and include the URL to the first page of your solution. Provide the entire URL including the http:// prefix.