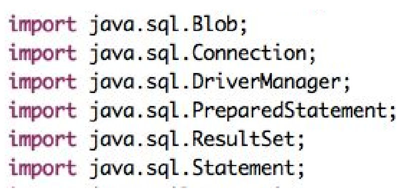
Criterion C: Development

List of Techniques Used

All my source code is presented in Appendix 3.

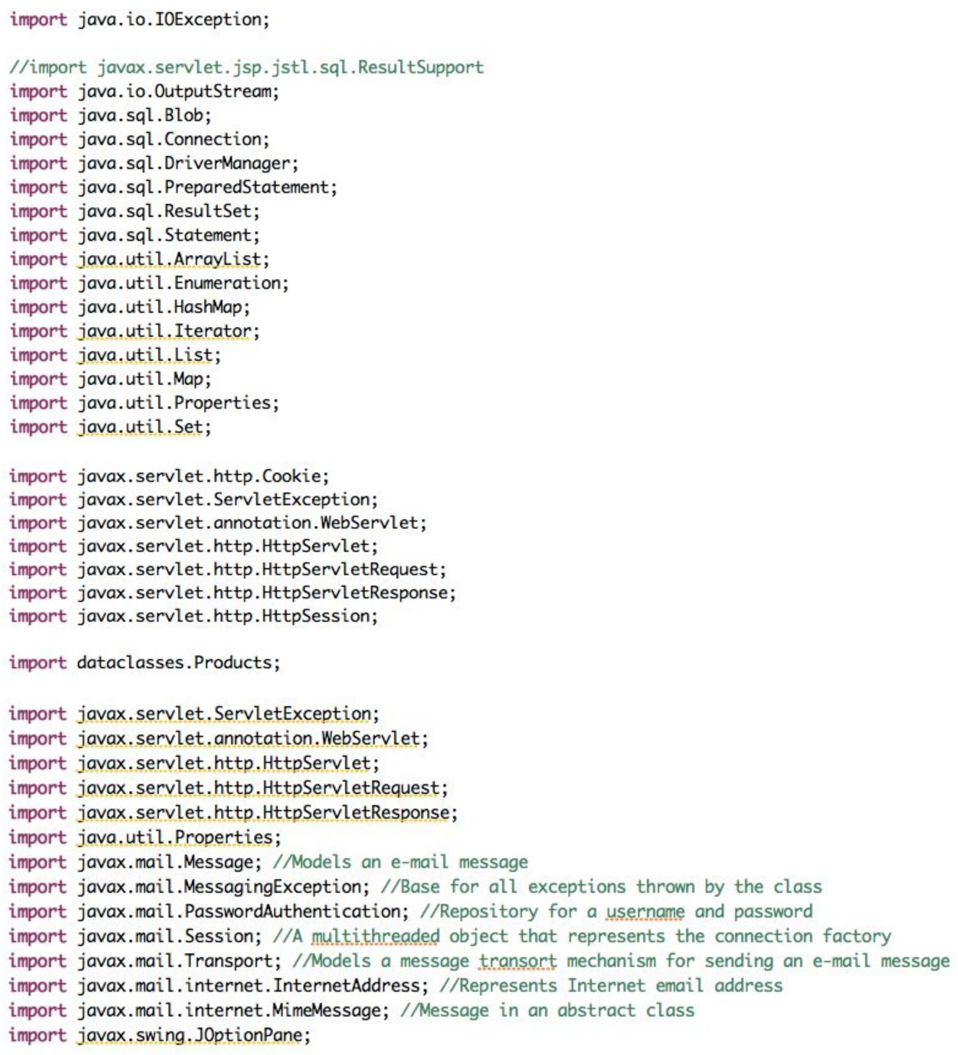
- Using Structured Query Language (SQL) through MySQL
  - Inserting into database
  - Deleting from database
  - Adding images to database
  - Sorting database via column headings
- Displaying of database items
  - Creating a table on ‘Database’ page of website that displays all the products in the Database
  - Accessing Information From the Database to print on the CartForm before the user confirms their order
- Using HTML5 (Hypertext Markup Language 5) for data validation of values entered in CartForm
- Using JSTL (JavaScript Template Language)
  - Creating a Form containing tables of information and Images
  - Printing the total price of selected products and their quantities
- Using CSS (Cascading Style Sheets) To Create a Parallax Effect
- Using Java to control the entire user interface through the Controller Servlet
  - ConfirmOrder (Returns parameters inputted into the CartForm and returns it in the form of an email draft)
  - shoppingCart (Adding additional values to the shopping cart on the CartForm after selecting items on the Products page previously)
  - totalPrice (Calculating the total price of all products selected)
  - getImage (Image Manipulation)
  - sendMail (Sending emails to users after they purchase products)

Libraries Imported



```
import java.sql.Blob;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
```

Figure 1: Contains the libraries imported by the main Controller servlet that provides for and controls the entire user interface through its connection to the Database class.



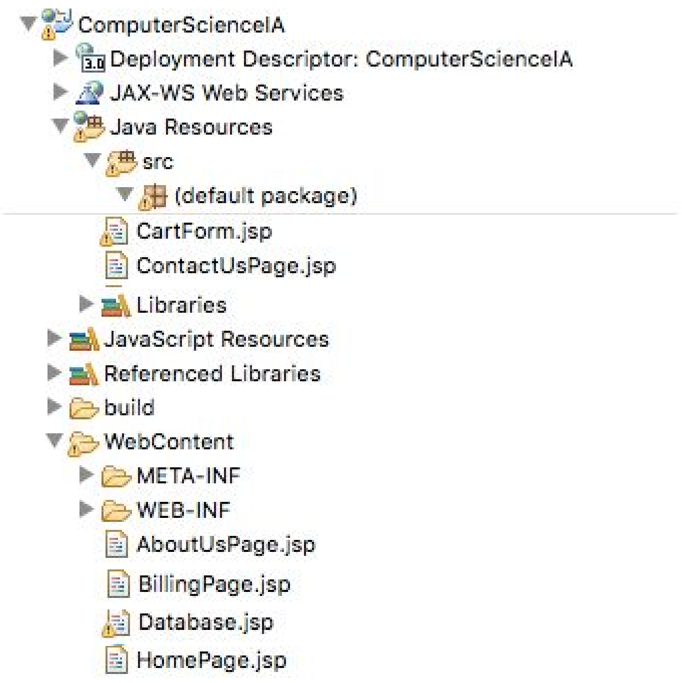
```
import java.io.IOException;

//import javax.servlet.jsp.jstl.sql.ResultSupport
import java.io.OutputStream;
import java.sql.Blob;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

import javax.servlet.http.Cookie;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import dataclasses.Products;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Properties;
import javax.mail.Message; //Models an e-mail message
import javax.mail.MessagingException; //Base for all exceptions thrown by the class
import javax.mail.PasswordAuthentication; //Repository for a username and password
import javax.mail.Session; //A multithreaded object that represents the connection factory
import javax.mail.Transport; //Models a message transort mechanism for sending an e-mail message
import javax.mail.internet.InternetAddress; //Represents Internet email address
import javax.mail.internet.MimeMessage; //Message in an abstract class
import javax.swing.JOptionPane;
```

Figure 2: The libraries imported in the *Controller* class. These libraries are also used in the other classes so the libraries imported for each class is not shown as they are the same with some not being used all the time.

Library	*Use*
java.sql	Link to the database created in MySQL
java.util	Create user interfaces, paint graphics, and provide interfaces and classes for transferring data between and within applications.
javax.servlet	Define objects that receives requests from a user and sends them to a servlet on the server.
javax.mail	Send emails to email addresses.
javax.swing	Allow code to work in similar ways on all java runtime programs.

Classes

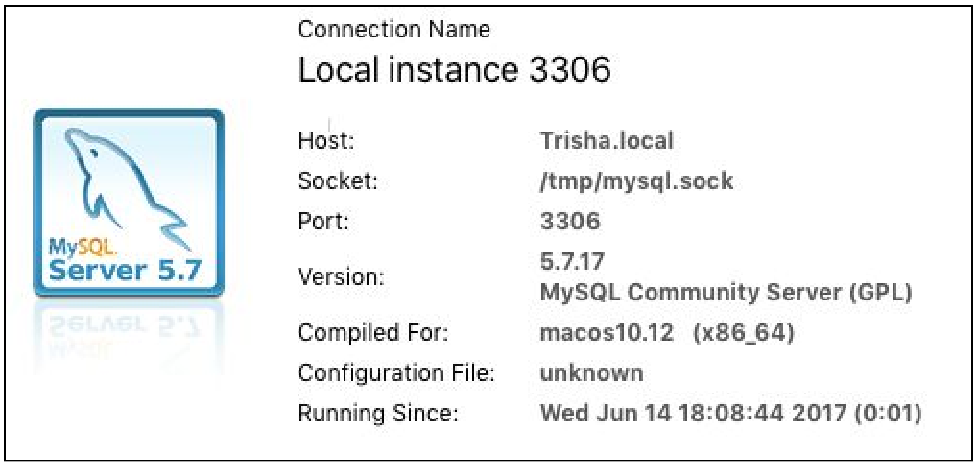
Figure 3: The classes in the project.



```
▼ ComputerScienceIA
  ▶ Deployment Descriptor: ComputerScienceIA
  ▶ JAX-WS Web Services
  ▼ Java Resources
    ▼ src
      ▼ (default package)
    CartForm.jsp
    ContactUsPage.jsp
    ▶ Libraries
  ▶ JavaScript Resources
  ▶ Referenced Libraries
  ▶ build
  ▼ WebContent
    ▶ META-INF
    ▶ WEB-INF
    AboutUsPage.jsp
    BillingPage.jsp
    Database.jsp
    HomePage.jsp
```

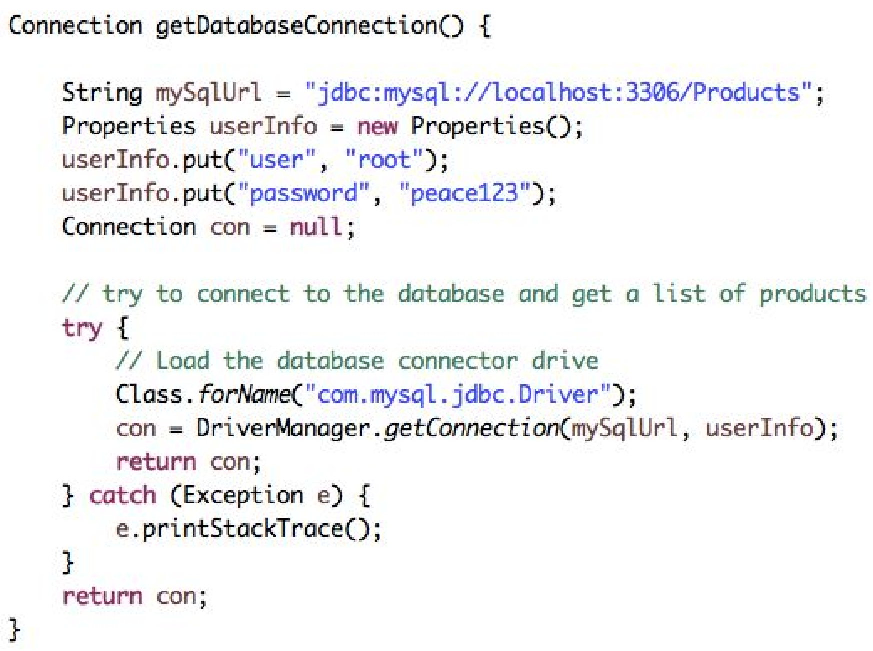
Using Structured Query Language (SQL) through MySQL

SQLConnection



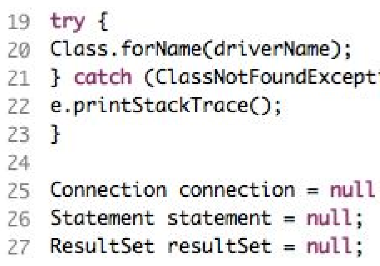
Connection Name	Local instance 3306
Host:	Trisha.local
Socket:	/tmp/mysql.sock
Port:	3306
Version:	5.7.17 MySQL Community Server (GPL)
Compiled For:	macos10.12 (x86_64)
Configuration File:	unknown
Running Since:	Wed Jun 14 18:08:44 2017 (0:01)

Figure 4: The database connection name and other details taken from MySQL Workbench 6.3CE.



```
Connection getDatabaseConnection() {

    String mySqlUrl = "jdbc:mysql://localhost:3306/Products";
    Properties userInfo = new Properties();
    userInfo.put("user", "root");
    userInfo.put("password", "peace123");
    Connection con = null;

    // try to connect to the database and get a list of products
    try {
        // Load the database connector drive
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection(mySqlUrl, userInfo);
        return con;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}
```

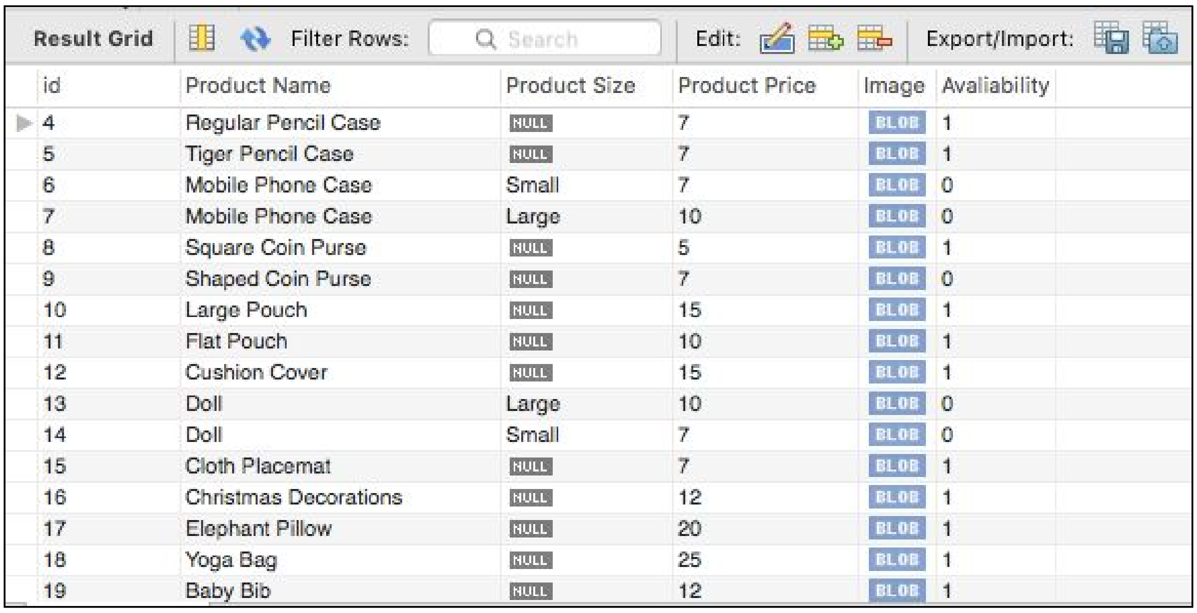
Figure 5: The code used to connect the database ‘products’ to the controller servlet. The try statement is executed if no exceptions are thrown, and loads the database connector drive.



```
try {
Class.forName(driverName);
} catch (ClassNotFoundException e) {
e.printStackTrace();
}

Connection connection = null;
Statement statement = null;
ResultSet resultSet = null;
```

Figure 6: This code allows the database to be connected to the *Database* class, which is a JavaServer Page that displays the database table on an HTML page.

Inserting/Deleting from Database



Result Grid | Filter Rows: Search | Edit: | Export/Import:

id	Product Name	Product Size	Product Price	Image	Avaliability
4	Regular Pencil Case	NULL	7	BLOB	1
5	Tiger Pencil Case	NULL	7	BLOB	1
6	Mobile Phone Case	Small	7	BLOB	0
7	Mobile Phone Case	Large	10	BLOB	0
8	Square Coin Purse	NULL	5	BLOB	1
9	Shaped Coin Purse	NULL	7	BLOB	0
10	Large Pouch	NULL	15	BLOB	1
11	Flat Pouch	NULL	10	BLOB	1
12	Cushion Cover	NULL	15	BLOB	1
13	Doll	Large	10	BLOB	0
14	Doll	Small	7	BLOB	0
15	Cloth Placemat	NULL	7	BLOB	1
16	Christmas Decorations	NULL	12	BLOB	1
17	Elephant Pillow	NULL	20	BLOB	1
18	Yoga Bag	NULL	25	BLOB	1
19	Baby Bib	NULL	12	BLOB	1

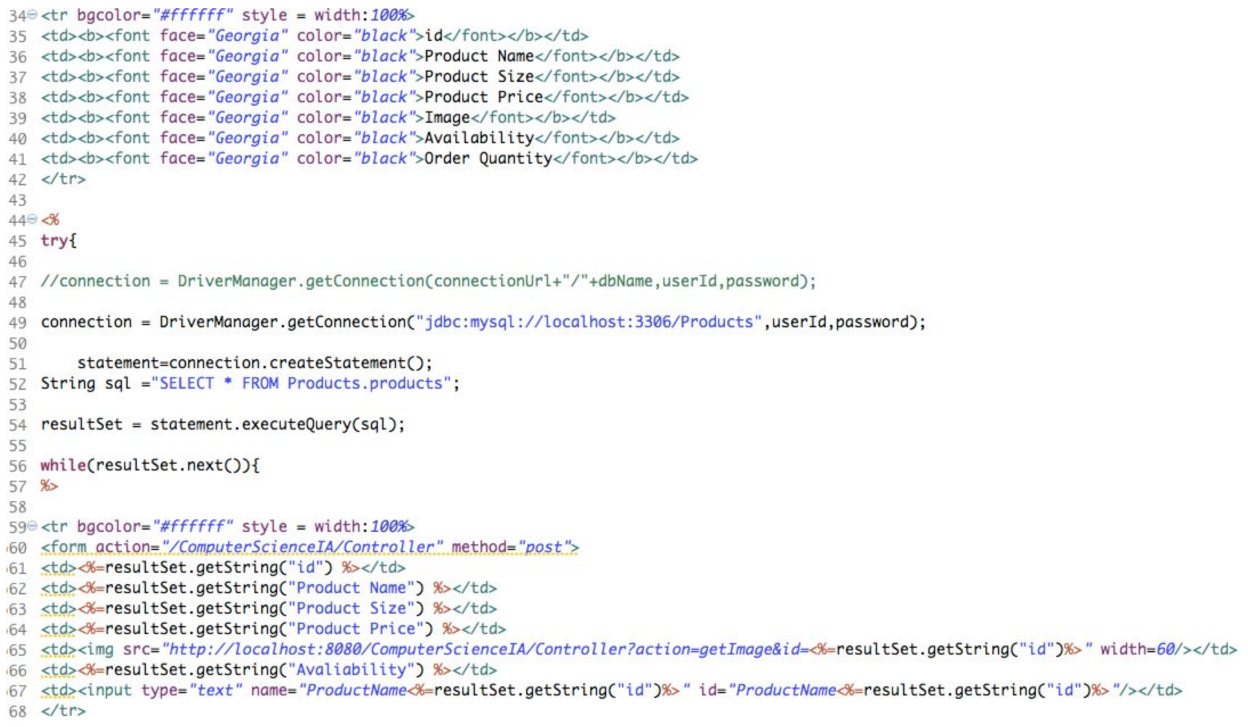
Figure 7: This shows a part of the *products* database. The top of the database shows the columns' names, which are referred to throughout my code. The images are of type 'mediumblob', and are converted into files that can be viewed on the website, which can be seen in the getImage function below. Some of the products do not have specific sizes, for example 'Cloth Placemat'. Therefore, the elements are shown as null.

```
        try {
            connection = getDatabaseConnection();
            PreparedStatement ps = connection.prepareStatement("insert into Orders values (?,?,?,?,?,?,?)");
            ps.setString(1, firstname);
            ps.setString(2, lastname);
            ps.setString(6, studentmentor);
            ps.setString(7, email);

            ps.executeUpdate();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Figure 8: This function adds the order description (which was inputted into the *CartForm* by the user) into the *Orders* database, by getting a connection with the database.

Displaying Database Items

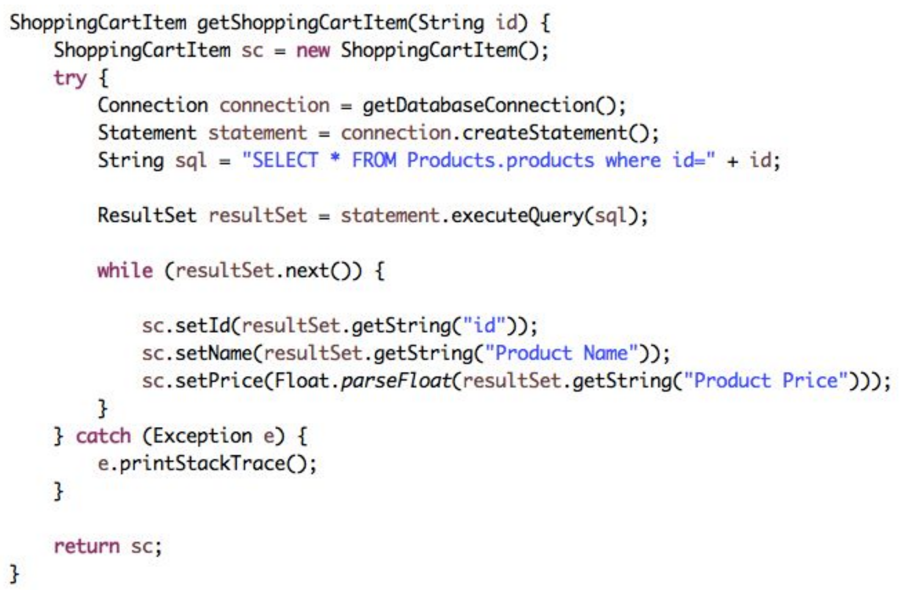
Creating a table on ‘Database’ page of website that displays all the products in the Database



```
34 <tr bgcolor="#ffffff" style = width:100%>
35 <td><b><font face="Georgia" color="black">id</font></b></td>
36 <td><b><font face="Georgia" color="black">Product Name</font></b></td>
37 <td><b><font face="Georgia" color="black">Product Size</font></b></td>
38 <td><b><font face="Georgia" color="black">Product Price</font></b></td>
39 <td><b><font face="Georgia" color="black">Image</font></b></td>
40 <td><b><font face="Georgia" color="black">Availability</font></b></td>
41 <td><b><font face="Georgia" color="black">Order Quantity</font></b></td>
42 </tr>
43
44 <%
45 try{
46
47 //connection = DriverManager.getConnection(connectionUrl+"/"+dbName,userId,password);
48
49 connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/Products",userId,password);
50
51     statement=connection.createStatement();
52 String sql ="SELECT * FROM Products.products";
53
54 resultSet = statement.executeQuery(sql);
55
56 while(resultSet.next()){
57 %>
58
59 <tr bgcolor="#ffffff" style = width:100%>
60 <form action="/ComputerScienceIA/Controller" method="post">
61 <td><%=resultSet.getString("id") %></td>
62 <td><%=resultSet.getString("Product Name") %></td>
63 <td><%=resultSet.getString("Product Size") %></td>
64 <td><%=resultSet.getString("Product Price") %></td>
65 <td><img src="http://localhost:8080/ComputerScienceIA/Controller?action=getImage&id=<%=resultSet.getString("id")%>" width=60/></td>
66 <td><%=resultSet.getString("Avaliability") %></td>
67 <td><input type="text" name="ProductName<%=resultSet.getString("id")%>" id="ProductName<%=resultSet.getString("id")%>"/></td>
68 </tr>
```

Figure 9: This shows the code used to create a table of products in the *CartForm* by using the Products table in the database, allowing components of the database (the products, their names, images etc.) to be displayed on the *Products* page of the website. The code from lines 34-42 is used to create the products table. The code from lines 59-68 allow transferring of information under each column from the database.

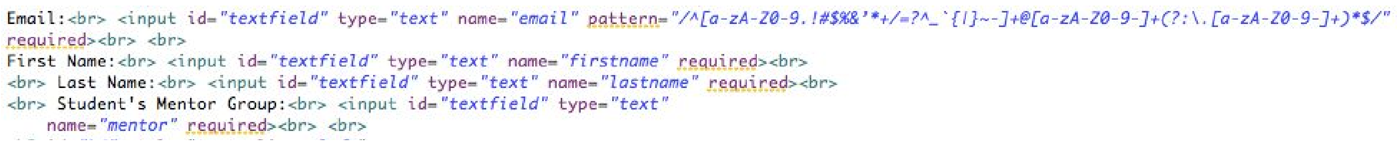
Accessing Information From the Database to print on the *CartForm* before the user confirms their Order



```
ShoppingCartItem getShoppingCartItem(String id) {
    ShoppingCartItem sc = new ShoppingCartItem();
    try {
        Connection connection = getDatabaseConnection();
        Statement statement = connection.createStatement();
        String sql = "SELECT * FROM Products.products where id=" + id;

        ResultSet resultSet = statement.executeQuery(sql);

        while (resultSet.next()) {

            sc.setId(resultSet.getString("id"));
            sc.setName(resultSet.getString("Product Name"));
            sc.setPrice(Float.parseFloat(resultSet.getString("Product Price")));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return sc;
}
```

Figure 10: This function receives the selected product’s id, name and price from the *products* database to be placed into the shopping cart, which is later displayed on the *CartForm* before the user confirms their order.

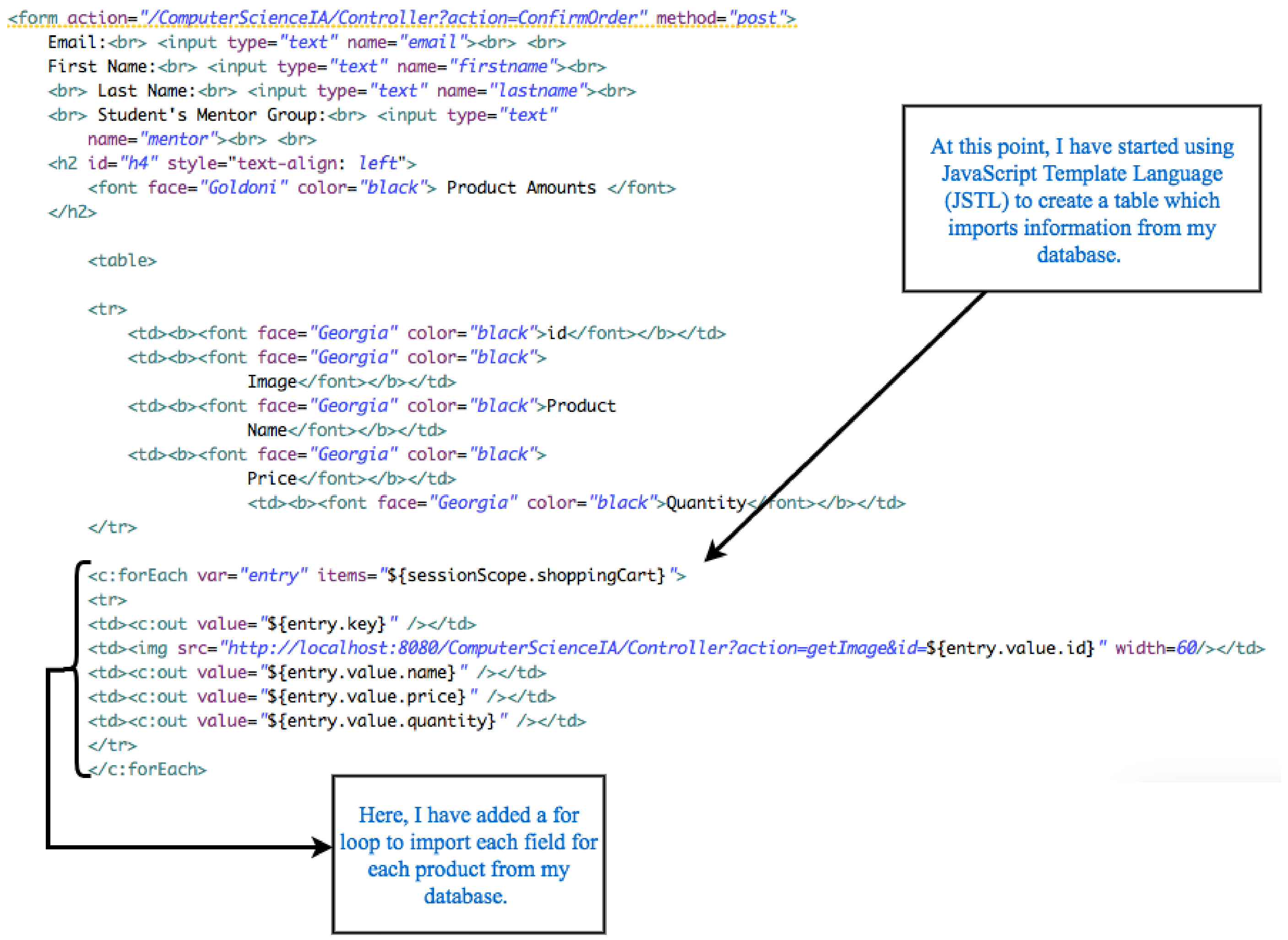
Using HTML5 (Hypertext Markup Language 5) for data validation of values entered in CartForm



```
Email:<br> <input id="textfield" type="text" name="email" pattern="/^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/"
required><br> <br>
First Name:<br> <input id="textfield" type="text" name="firstname" required><br>
<br> Last Name:<br> <input id="textfield" type="text" name="lastname" required><br>
<br> Student's Mentor Group:<br> <input id="textfield" type="text"
    name="mentor" required><br> <br>
```

Figure 11: The email entered into the email text field in the CartForm by the user is validated based on whether it is the correct format (format check). Emails are not allowed to have certain characters such as '&', which are included in the set of unused characters above. In addition, the email, first name, last name and mentor group are checked from presence (presence check) by using the keyword 'required' at the end of each line of code in this section. If the 'ConfirmOrder' button is clicked without completing any of the fields, an alert will be displayed on the screen directing the user to type in their information where it is empty.

Using JSTL (JavaScript Template Language) to create a Form containing tables of information and Images



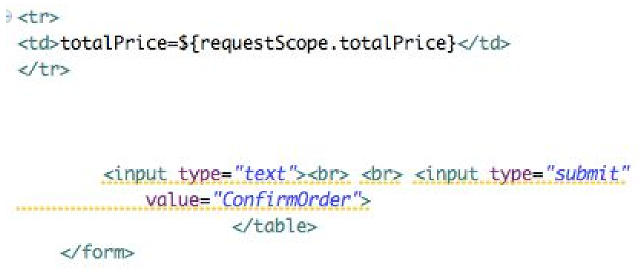
```
<form action="/ComputerScienceIA/Controller?action=ConfirmOrder" method="post">
    Email:<br> <input type="text" name="email"><br> <br>
    First Name:<br> <input type="text" name="firstname"><br>
    <br> Last Name:<br> <input type="text" name="lastname"><br>
    <br> Student's Mentor Group:<br> <input type="text"
        name="mentor"><br> <br>
    <h2 id="h4" style="text-align: left">
        <font face="Goldoni" color="black"> Product Amounts </font>
    </h2>

        <table>

        <tr>
            <td><b><font face="Georgia" color="black">id</font></b></td>
            <td><b><font face="Georgia" color="black">
                        Image</font></b></td>
            <td><b><font face="Georgia" color="black">Product
                        Name</font></b></td>
            <td><b><font face="Georgia" color="black">
                        Price</font></b></td>
                        <td><b><font face="Georgia" color="black">Quantity</font></b></td>
        </tr>

        <c:forEach var="entry" items="${sessionScope.shoppingCart}">
        <tr>
        <td><c:out value="${entry.key}" /></td>
        <td><img src="http://localhost:8080/ComputerScienceIA/Controller?action=getImage&id=${entry.value.id}" width=60/></td>
        <td><c:out value="${entry.value.name}" /></td>
        <td><c:out value="${entry.value.price}" /></td>
        <td><c:out value="${entry.value.quantity}" /></td>
        </tr>
        </c:forEach>
```

At this point, I have started using JavaScript Template Language (JSTL) to create a table which imports information from my database.

Here, I have added a for loop to import each field for each product from my database.

Figure 12: This shows the *CartForm*, which is a page that is opened once the purchase button on the *Products* website page is clicked. The products that were selected by the user on the *Products* page of the website appear once again at the bottom of the *CartForm* to confirm the user's order before they click submit. These products are printed using a JSTL for loop, which is shown in the image above. **Justification:** By using JSTL language, I have completed the code for this loop in a way that is much shorter and neater when compared to using HTML5, therefore increasing the extensibility of my code.

Using JSTL (JavaScript Template Language) To Print the Total Price of Selected Products and their Quantities

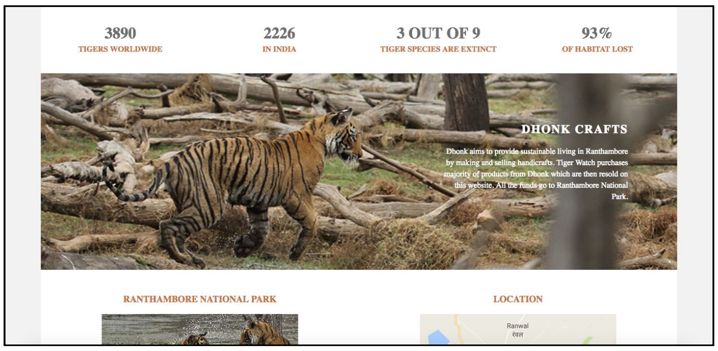


```
<tr>
<td>totalPrice=${requestScope.totalPrice}</td>
</tr>


        <input type="text"><br> <br> <input type="submit"
            value="ConfirmOrder">
                    </table>
    </form>
```

Figure 13: The products that were selected by the user on the *Products* page of the website appear once again at the bottom of the *CartForm* to confirm the user's order before they click submit. The total price of all products is requested from the Controller Servlet by using JSTL code, allowing the total price to be returned on the screen at the bottom of the CartForm, just above the *ConfirmOrder* button.

Using CSS (Cascading Style Sheets) To Create a Parallax Effect

The parallax effect involves an image (onetiger.png) remaining stationary as the user scrolls up or down the page. Text is also displayed in this area. The parallax effect on the *Home* page of my website can be seen below.

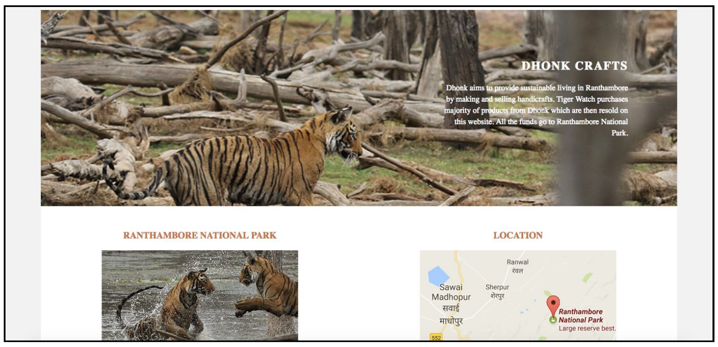


3890 TIGERS WORLDWIDE | 2226 IN INDIA | 3 OUT OF 9 TIGER SPECIES ARE EXTINCT | 93% OF HABITAT LOST

DHONK CRAFTS

Dhonk aims to provide sustainable living in Ranthambore by making and selling handicrafts. Tiger Watch purchases majority of products from Dhonk which are then resold on this website. All the funds go to Ranthambore National Park.

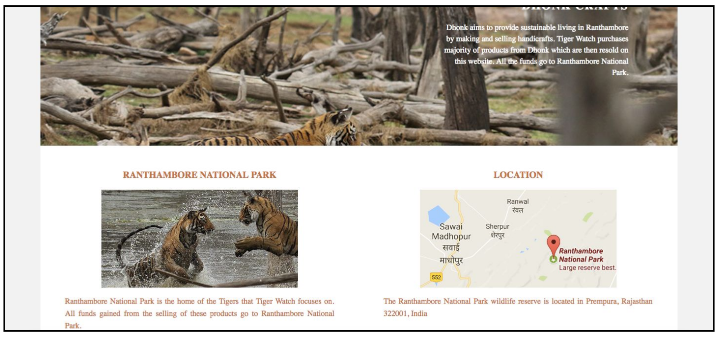
RANTHAMBORE NATIONAL PARK | LOCATION



DHONK CRAFTS

Dhonk aims to provide sustainable living in Ranthambore by making and selling handicrafts. Tiger Watch purchases majority of products from Dhonk which are then resold on this website. All the funds go to Ranthambore National Park.

RANTHAMBORE NATIONAL PARK | LOCATION

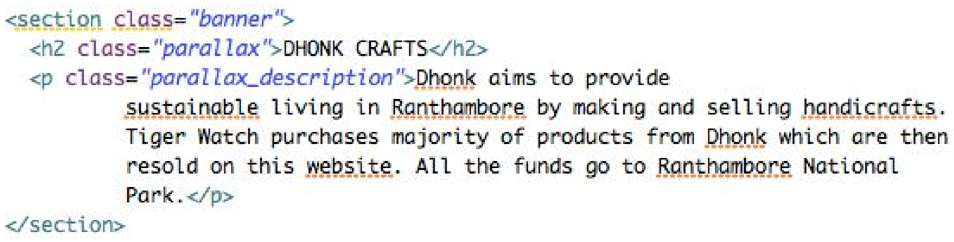


Dhonk aims to provide sustainable living in Ranthambore by making and selling handicrafts. Tiger Watch purchases majority of products from Dhonk which are then resold on this website. All the funds go to Ranthambore National Park.

RANTHAMBORE NATIONAL PARK | LOCATION

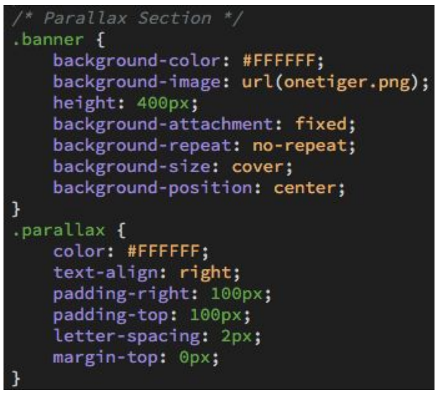
Ranthambore National Park is the home of the Tigers that Tiger Watch focuses on. All funds gained from the selling of these products go to Ranthambore National Park.

The Ranthambore National Park wildlife reserve is located in Prempura, Rajasthan 322001, India

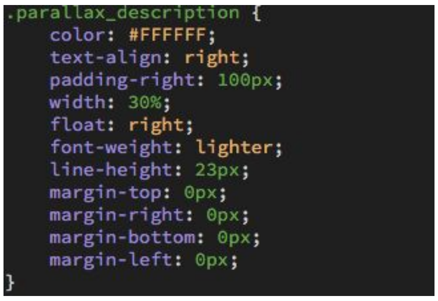


```html
<section class="banner">
  <h2 class="parallax">DHONK CRAFTS</h2>
  <p class="parallax_description">Dhonk aims to provide
          sustainable living in Ranthambore by making and selling handicrafts.
          Tiger Watch purchases majority of products from Dhonk which are then
          resold on this website. All the funds go to Ranthambore National
          Park.</p>
</section>
```

Figure 14: The code above shows the HTML5 code involved when creating a Parallax effect. the HTML code is minimal here, however, the class names declared, eg. banner and parallax, are used in the connected CSS page below.



```css
/* Parallax Section */
.banner {
    background-color: #FFFFFF;
    background-image: url(onetiger.png);
    height: 400px;
    background-attachment: fixed;
    background-repeat: no-repeat;
    background-size: cover;
    background-position: center;
}
.parallax {
    color: #FFFFFF;
    text-align: right;
    padding-right: 100px;
    padding-top: 100px;
    letter-spacing: 2px;
    margin-top: 0px;
}
```



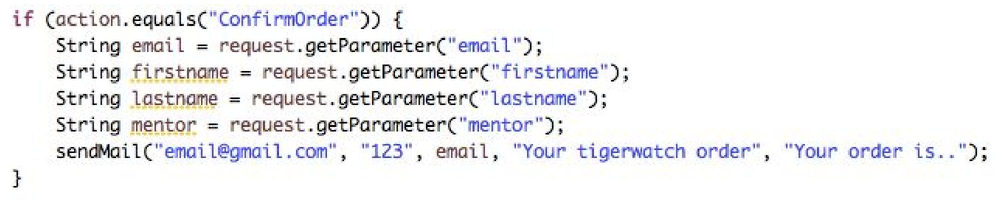
```css
.parallax_description {
    color: #FFFFFF;
    text-align: right;
    padding-right: 100px;
    width: 30%;
    float: right;
    font-weight: lighter;
    line-height: 23px;
    margin-top: 0px;
    margin-right: 0px;
    margin-bottom: 0px;
    margin-left: 0px;
}
```

Figure 15: The CSS code on the page ‘singlePageTemplate.css’ is shown above. Here, the banner class keeps the image fixed in place as the user scrolls upwards or downwards. The parallax class pads the image and keeps it in place. The parallax_description class allows the font of the text displayed on the parallax to be designed.

Using Java to control the entire user interface through the *Controller* Servlet

The controller contains the main servlet that provides for and controls the user interface through its connection to the *Database* class. This allows my project to be quite extensible overall because of my good Object-Oriented design. This means that duplication of code is not required as each class aggregates from the *Controller* itself.

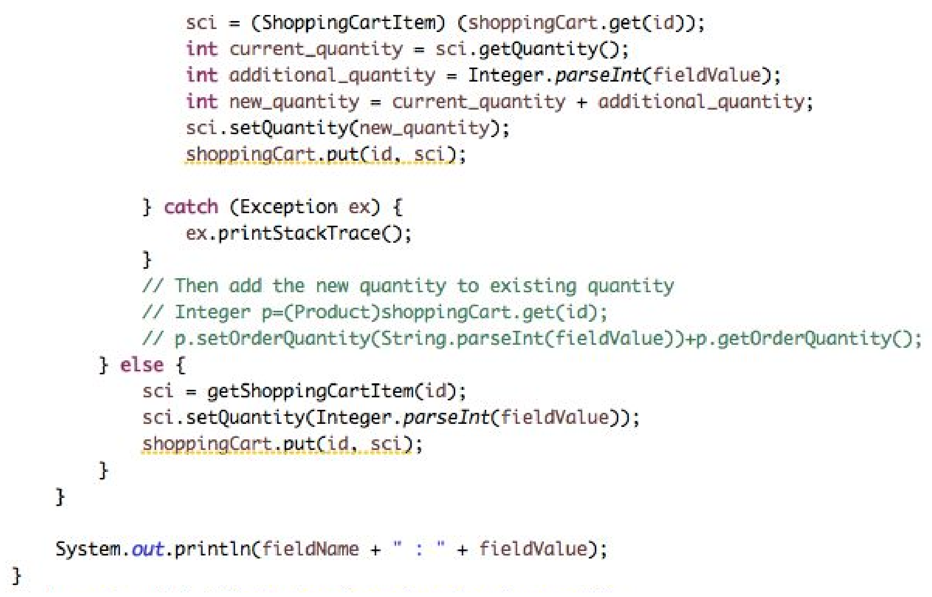
ConfirmOrder (Returns parameters inputted into the CartForm and returns it in the form of an email draft)



```java
if (action.equals("ConfirmOrder")) {
    String email = request.getParameter("email");
    String firstname = request.getParameter("firstname");
    String lastname = request.getParameter("lastname");
    String mentor = request.getParameter("mentor");
    sendMail("email@gmail.com", "123", email, "Your tigerwatch order", "Your order is..");
}
```

Figure 16: This function receives the parameters inputted into the *CartForm* by the user (as explained above on page 7) and returns it in the format of an email draft which is sent to the purchaser/user through the email function after the user confirms their order. *ConfirmOrder* here refers to the final confirmation button at the bottom of the *CartForm.*

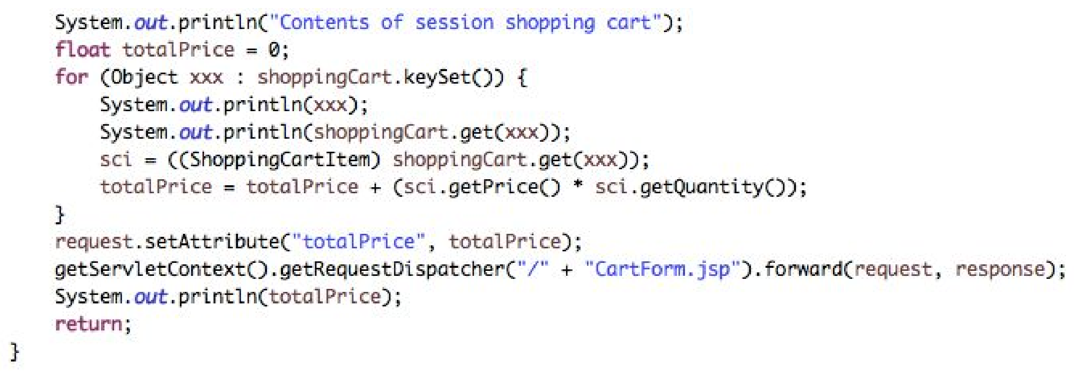
shoppingCart (Adding additional values to the shopping cart on the *CartForm* after selecting items on the *Products* page previously)



```java
                sci = (ShoppingCartItem) (shoppingCart.get(id));
                int current_quantity = sci.getQuantity();
                int additional_quantity = Integer.parseInt(fieldValue);
                int new_quantity = current_quantity + additional_quantity;
                sci.setQuantity(new_quantity);
                shoppingCart.put(id, sci);

            } catch (Exception ex) {
                ex.printStackTrace();
            }
            // Then add the new quantity to existing quantity
            // Integer p=(Product)shoppingCart.get(id);
            // p.setOrderQuantity(String.parseInt(fieldValue))+p.getOrderQuantity();
        } else {
            sci = getShoppingCartItem(id);
            sci.setQuantity(Integer.parseInt(fieldValue));
            shoppingCart.put(id, sci);
        }
    }

    System.out.println(fieldName + " : " + fieldValue);
}
```

Figure 17: The *ConfirmOrder* function is activated once the user clicks the submit button at the bottom of the *CartForm*. If the user had added an amount of products on the products page itself and then added a new amount on the *CartForm* page for the same product, their existing quantities can be increased or decreased if necessary. The function does this by adding new order quantities (additional_quantity) entered by the user to the existing order quantity (current_quantity) to display a total order quantity on the *CartForm*.

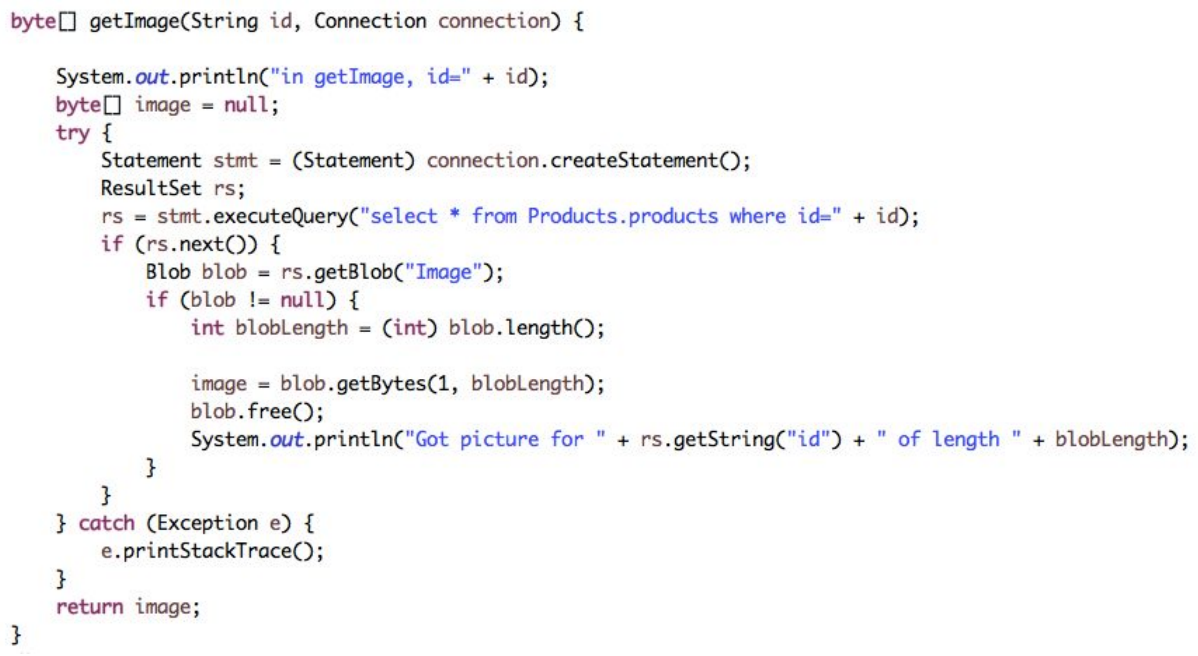
totalPrice (Calculating the total price of all products selected)



```
        System.out.println("Contents of session shopping cart");
        float totalPrice = 0;
        for (Object xxx : shoppingCart.keySet()) {
            System.out.println(xxx);
            System.out.println(shoppingCart.get(xxx));
            sci = ((ShoppingCartItem) shoppingCart.get(xxx));
            totalPrice = totalPrice + (sci.getPrice() * sci.getQuantity());
        }
        request.setAttribute("totalPrice", totalPrice);
        getServletContext().getRequestDispatcher("/" + "CartForm.jsp").forward(request, response);
        System.out.println(totalPrice);
        return;
    }
```

Figure 18: This function calculates the total price of the products and their respective quantities as selected by the user. the total price is the price multiplied by the quantity inputted, as shown above. The information about the prices of each product is returned from the *products* database. The total price is then displayed at the bottom of the *CartForm* through the use of JSTL, which can be seen on page 9. Additionally, if the user decides to add a new quantity, the value is also added to the overall total price.

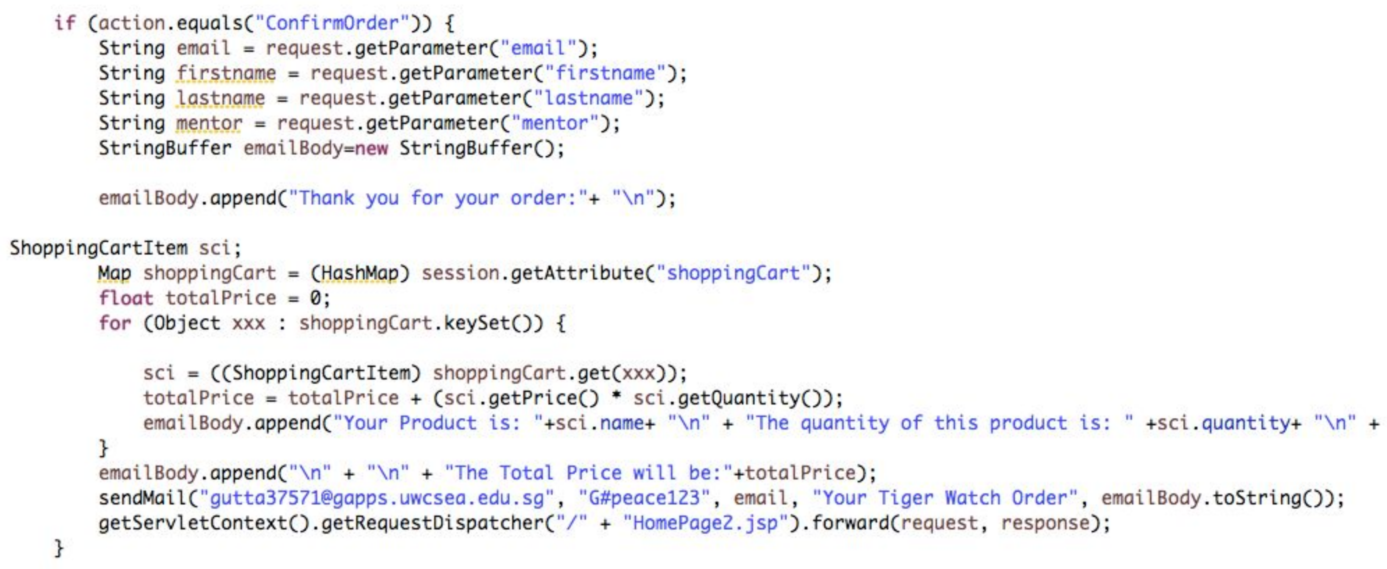
getImage Function (Image Manipulation when displaying on website page)



```
byte[] getImage(String id, Connection connection) {

    System.out.println("in getImage, id=" + id);
    byte[] image = null;
    try {
        Statement stmt = (Statement) connection.createStatement();
        ResultSet rs;
        rs = stmt.executeQuery("select * from Products.products where id=" + id);
        if (rs.next()) {
            Blob blob = rs.getBlob("Image");
            if (blob != null) {
                int blobLength = (int) blob.length();

                image = blob.getBytes(1, blobLength);
                blob.free();
                System.out.println("Got picture for " + rs.getString("id") + " of length " + blobLength);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return image;
}
```

Figure 19: This function inserts the image in the form ‘mediumblob’ from the *products* database, so that it can be displayed on the *Products* JavaServer web page. In order to do this, it calculates the length of the blob in the ‘Products’ database, and converts it by getting the image’s id value from the *products* database.

sendMail (Sending emails to users after they purchase products)



```
    if (action.equals("ConfirmOrder")) {
        String email = request.getParameter("email");
        String firstname = request.getParameter("firstname");
        String lastname = request.getParameter("lastname");
        String mentor = request.getParameter("mentor");
        StringBuffer emailBody=new StringBuffer();

        emailBody.append("Thank you for your order:"+ "\n");

 ShoppingCartItem sci;
        Map shoppingCart = (HashMap) session.getAttribute("shoppingCart");
        float totalPrice = 0;
        for (Object xxx : shoppingCart.keySet()) {

            sci = ((ShoppingCartItem) shoppingCart.get(xxx));
            totalPrice = totalPrice + (sci.getPrice() * sci.getQuantity());
            emailBody.append("Your Product is: "+sci.name+ "\n" + "The quantity of this product is: " +sci.quantity+ "\n" +
        }
        emailBody.append("\n" + "\n" + "The Total Price will be:"+totalPrice);
        sendMail("gutta37571@gapps.uwcsea.edu.sg", "G#peace123", email, "Your Tiger Watch Order", emailBody.toString());
        getServletContext().getRequestDispatcher("/" + "HomePage2.jsp").forward(request, response);
    }
```

Figure 20: This function allows an email to be sent to the user when their email is inputted into the *CartForm* text field. The Msg variable includes the message to be sent to the users who purchased items. Their email is received from the email they had entered in the *CartForm* field.

1184 Words

Extensibility of Code

Examples	*Action Taken to Make Code Extensible*
ConfirmOrder, shoppingCart	I have ensured that my code exhibits good Object-Oriented design by having most functions in the Controller class only, so that there is a central class that has all the methods (aggregation). This eliminates the process of duplication of code and keeps code as simple as possible.
getShoppingCartItem	This function keeps my code simplistic as it gets information from a database and easily prints it into an email, which is more efficient than making the user physically input their order, or making a GC member physically type an email to all purchasers/users.
ConfirmOrder, shoppingCart	I have used reasonable variable names when coding for my project.
For Loop used to print products on the *CartForm*	I have used JSTL here, which makes my code much neater and shorter when compared to using HTML5.

Resources That I Utilised

- Oracle Java Guide: https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html.
- MySQL Guide: https://dev.mysql.com/doc/relnotes/mysql/5.7/en/.
- Stack Overflow: https://stackoverflow.com/.
- Java help Page: https://java.com/en/download/help/.