Group Number: 7

Group Members: Manisha Deshpande, Miles Robson

Project Title: Shopping Cart System

Shopping Cart System

Functional Specification Document

Version <1.0>

02 February 2015

Document Management

Initiation Date	**02 February 2015.**
Team Members	**Manisha Deshpande, Miles Robson**
Approver	

Revision History

Version	Revision Date	Modified By	Description of Revisions
1.0	**02 February 2015**	**Manisha Deshpande, Miles Robson**	
2.0	**20 April 2015**	**Manisha Deshpande, Miles Robson**	**Final Submission**

Table of Contents

1. INTRODUCTION

1.1 Purpose of this Document:

The purpose of this document is to specify the high level Functional Specifications of all the functionality to be incorporated into the Shopping Cart System for Release 1.0. The functional specifications were based on the requirements. These requirements have been documented in the Shopping Cart Project Requirements Report – COP 4331- Project Shopping Cart.

1.2 Glossary:

Term/Acronym	Description
Costs	Sum of invoice price for all items brought in the inventory (bought)
Revenues	Sum of sell price for all sold items
Profit	Revenues - Costs
Buyer	Customer who buys products
Seller	A User who adds products for sale in the system

1.3 Reference Documents

Document	Version	Author	Location
Project requirement	1.0	Dr. Cardei	FAU

2. Scope Overview

2.1 Description

This Functional Specification specifies the business processes to be performed by the Shopping Card System. It includes a detailed statement of the information processing and data requirements.

Basic functionality needed by the shopping cart is when the application begins, it shows a login window. Depending who logs in, a customer or the seller, the application performs different functions. The application does not arrange for shipping. The details of the specifications is intended to be sufficient to eliminate ambiguity as to the nature and intent of the new functionality while at the same time giving the team members the necessary scope to implement the functions as appropriate. The detailed functional specifications are given in section 3.1

2.2 Assumptions

The assumption made in the system is all payments are successful.

3. List of Functions

3.1 Functional Specifications

1. Login

1.1 New Users

The new users can use the Shopping cart application when they register using a login page. The new user has to click either New Seller or New Buyer depending on if he is registering as a seller or customer. A user profile page is going to be displayed for them to enter their information to create an account. The new user need to provide first name, last name, address, DOB, email id. The new user will be asked to choose a user name and password.

1.2 Existing Users

The existing users can be either a customer or seller who are already registered. User has to enter their user name and password to login. Depending on whether the user is a customer or a seller, system will provide certain functionality. The available functionalities for a customer are he can review Products details and add it to shopping cart, review/update shopping cart, checkout by providing credit card information. The available functionalities for a seller are review or update inventory, add new product for sale. The application keeps track of all costs, revenues and profits for seller. The seller can access this information from the application UI.

2. Update Account Information

This functionality will be used by user to edit or modify the user profile information. To use this functionality of the shopping cart application a user must be logged in either as customer or as a seller, with his user name and password. User's profile information will be displayed when user proceeds to check out, there he can edit his profile information by by entering information and then by clicking Edit.

3. Review Products details

To use this functionality of the shopping cart application a user must be logged in as a customer/buyer with his user name and password. A window (frame) opens where he can select product type and browse through a list of available products that includes the product name, price, and available quantity. From this window the customer can select products and add them to the shopping cart.

4. Adding Items to Shopping Cart

To use this functionality of the shopping cart application a user must **Login** as a customer with his user name and password. After **Review Products details,** the customer can add the product to the shopping cart by selecting product from the list of products and then by entering the quantity. By default the quantity to be purchased will be 1. Depending on availability of the selected product the selected product gets added to the shopping cart.

5. Review/Update Products in Shopping Cart

To use this functionality of the shopping cart application a user must Login as a customer with his user name and password. After Adding Items to Shopping Cart customer can review or update the shopping cart before check out by clicking either review/update cart button. The new window for reviewing shopping cart is displayed. Here customer can review / update his cart. The shopping cart total amount is kept current on the main product browse window for the customer.

5.1 Review Products in Shopping Cart

If the customer wants to review his shopping cart then the system will display a window showing all purchase details. Product name, quantity, unit price, total price and total amount to pay. Once the shopping cart is reviewed, customer can select update or checkout button to proceed.

5.2 Update Products in Shopping Cart

If the customer wants to update his shopping cart then the system will display a window showing all purchase details. Product name, quantity, unit price, total price and total amount to pay. Here customer can change the quantity/item count for each product in the cart and thus update his cart. Once the shopping cart is updated, customer can select review, update or checkout button to proceed.

6. Customer Check out

The customer can proceed to checkout at any time. On the checkout window, the shopping cart can be updated by changing the item count for each product in the cart. At checkout the customer verifies the shopping cart content and pays for the goods by supplying the credit card information. Customer has to enter credit card number, expiry date.

7. Add new Product for sale

To use this functionality of the shopping cart application a user must Login as a customer with his user name and password. When the seller logs in, a window opens where the seller can select to add a new product for sale by clicking add new product for sale. The seller has to enter product name, type, quantity, invoice price, and selling price because the internal product representation includes ID, type, quantity, invoice price, and selling price. System gives functionality for adding bundles. Bundle is two or more products sold as a single product.

8. Review/Update Inventory of a product.

7.1 Review Inventory of a product

To use this functionality of the shopping cart application a user must Login as a Seller with his user name and password. When the seller logs in, a window opens where the seller can review products in a table. A list of all products with product details such as product ID, name, type, quantity, invoice price, and selling price will be displayed for review.

7.2 Update Inventory of a product

To use this functionality of the shopping cart application a user must Login as a customer with his user name and password. When the seller logs in, a window opens where the current state of the inventory is shown. The seller can update the inventory by adding product quantity - specifying product name, invoice price, sell price and by updating the available quantity.

9. Display Profit Summary Report to Seller

The application must keep track of all costs, revenues and profits.

The seller will able to access Profit, revenues and Costs information from the application UI.

Profit = Revenues - Costs,

Revenues = Sum of sell price for all sold items

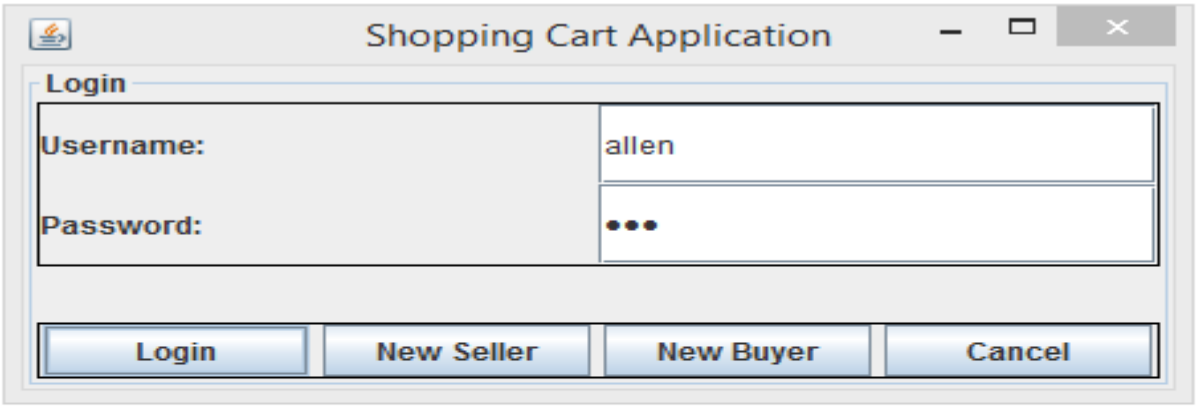
Costs = Sum of invoice price for all items brought in the inventory (bought)

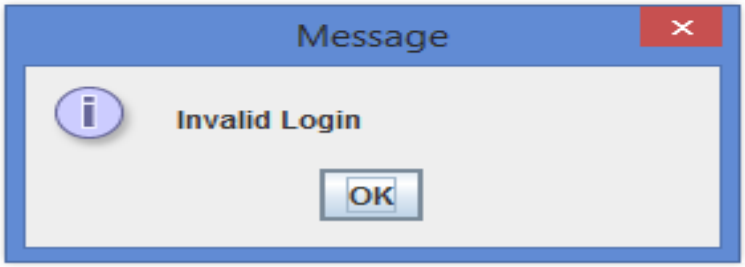
3.2 Application Architecture

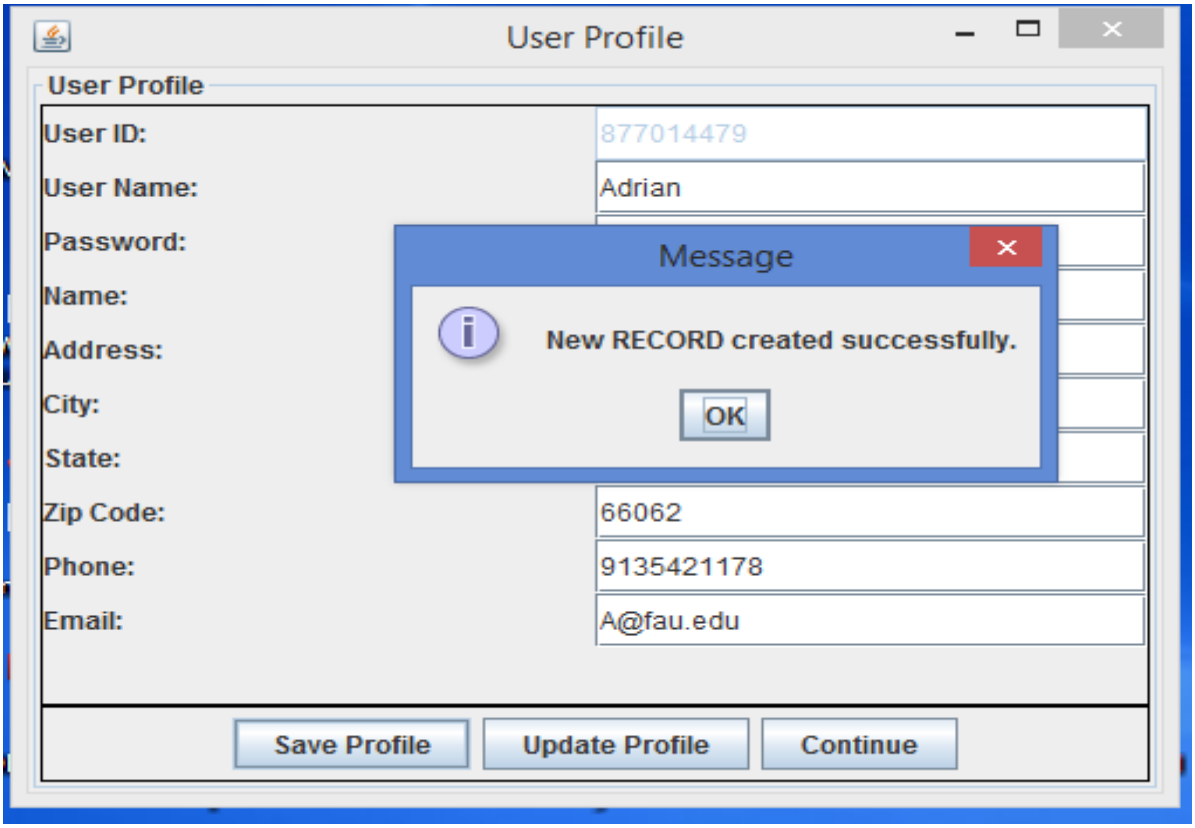
The Shopping Cart System is implemented as a Java application with Swing. The system uses the Model-View-Controllor architecture for handling GUI and changes to the models. The product/user databases are persistent. Oracle is the database used in the backend.

Use Cases For Shopping Cart System

Use Case ID	UC1
Use Case Name	**Login**
Primary Actors	**Customer, Seller**
Basic Flow(New User)	1. The User opens the shopping cart application. 2. The user clicks New User button 3. The system opens a window for entering the user information 4. The user enters his account information, username, password information, user type, billing/shipping address, payment information. 5. System validates the username is already in use. If not new account confirmation page will be displayed. User clicks ok. 6. The system displays screen for available functionality which user can select to proceed.
Basic Flow(Existing User)	1. The user opens the shopping cart application. 2. The user selects the Login/Existing user button 3. The system displays fields for entering user information. 4. The user enters information 5. The user logs in, and moves into the next screen, depending on user
Variation(New User) Info already in use	5a. A warning is shown to the user. 6a. Return to 4.
Variation(Existing User) Invalid UN/PW	5b. The user receives a warning. 6b. Return to 4.

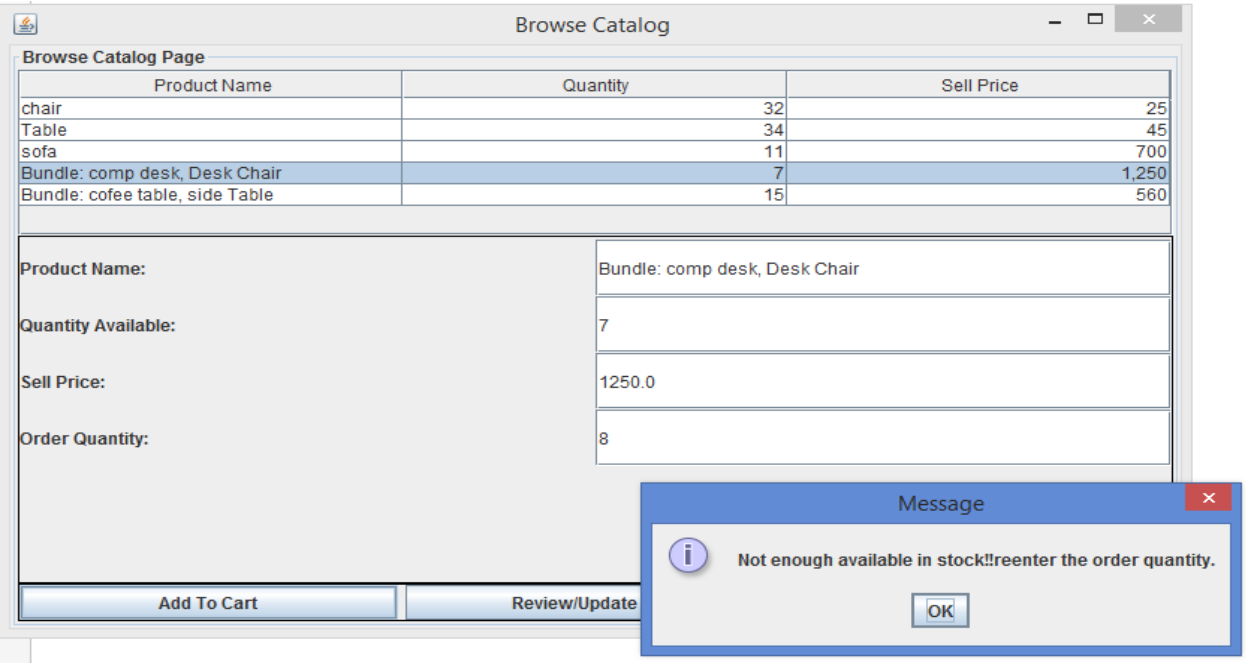


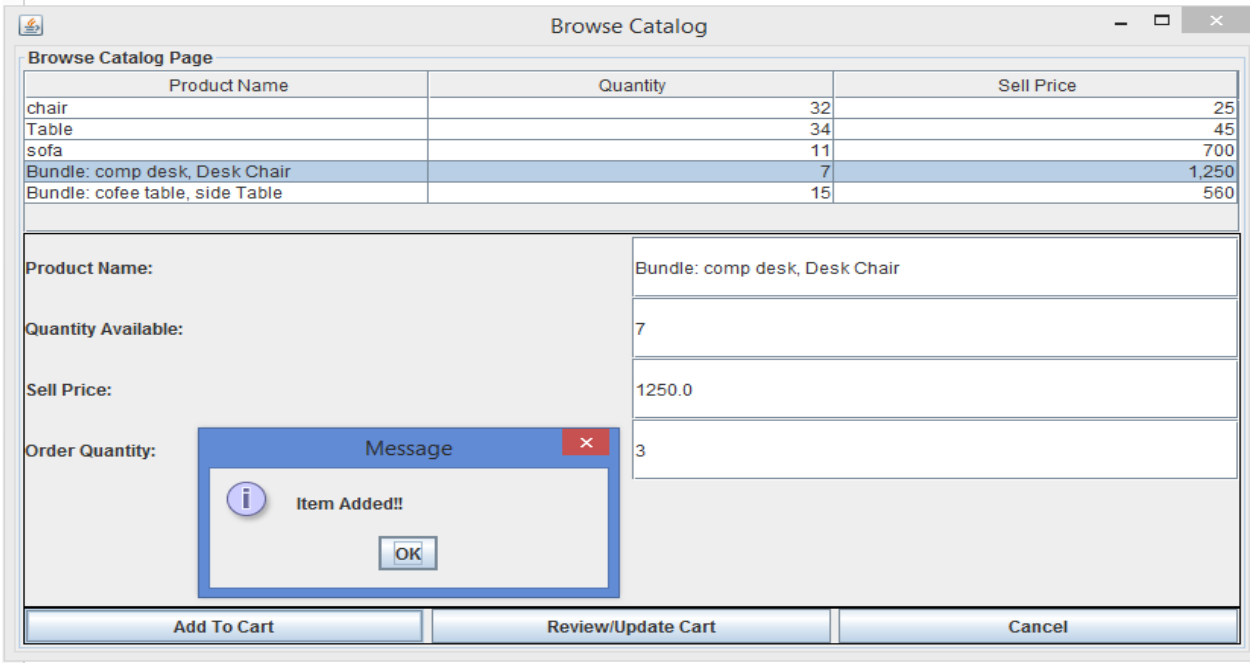




Use Case ID	**UC2**
Use Case Name	**Add product to the cart**
Primary Actors	**Customer**
Basic Flow	1. Customer ***Login*** the system 2. Customer is on the Product Catalog page - a list of available products that includes the product name, price, and available quantity. 3. User clicks on required product, indicates how many of an item they wish to Add to Cart. 4. User selects Add to Cart button. Item Added! Message displays. 5. The page refreshes. The shopping cart total amount is kept current on the main product browse window. 6. The items are added to the cart.
Variation # 1 Out of stock	2a.1 User attempts to indicate quantity greater than stock, and tries to add by clicking AddToCart. 2a.2. Error message is displayed as item/product is out of stock.

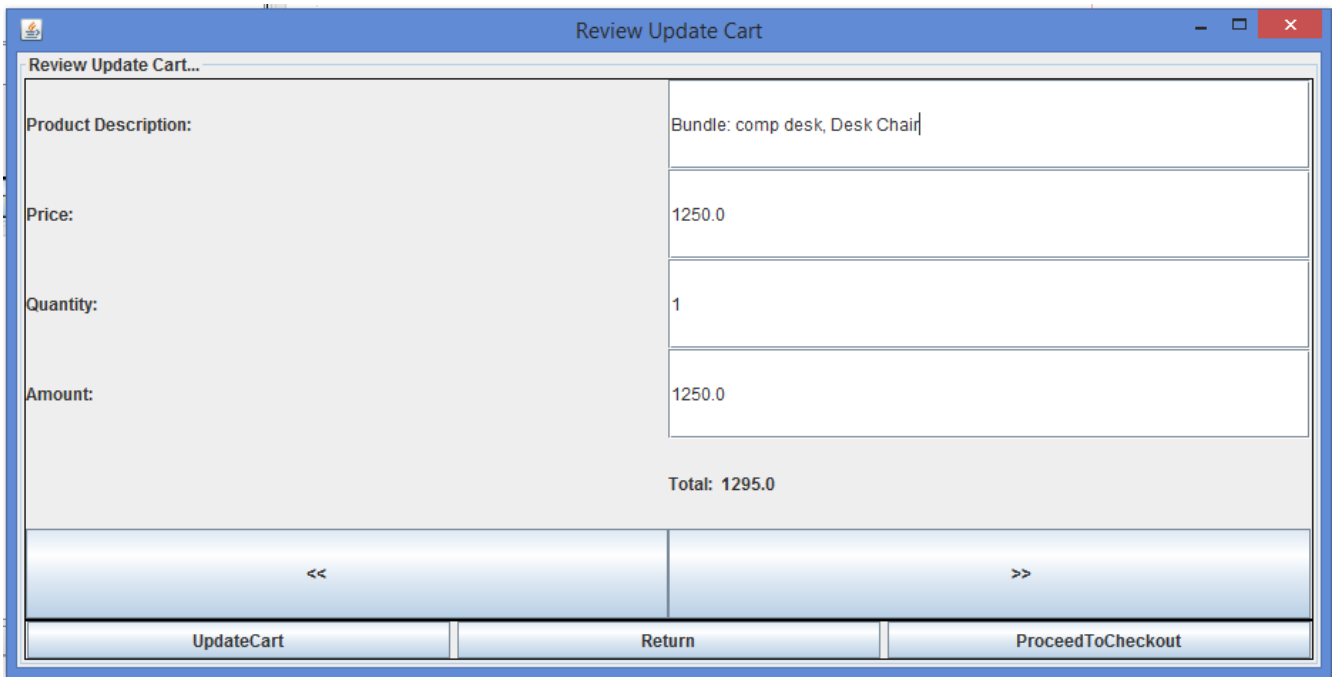
Use Case ID	UC3
Use Case Name	Reviewing product details
Primary Actors	Customer
Basic Flow	1. Customer ***Login*** the system 2. Customer is on display of items- Product Catalog page 3. Customer clicks an item. 4. The product information is displayed in the textboxes.



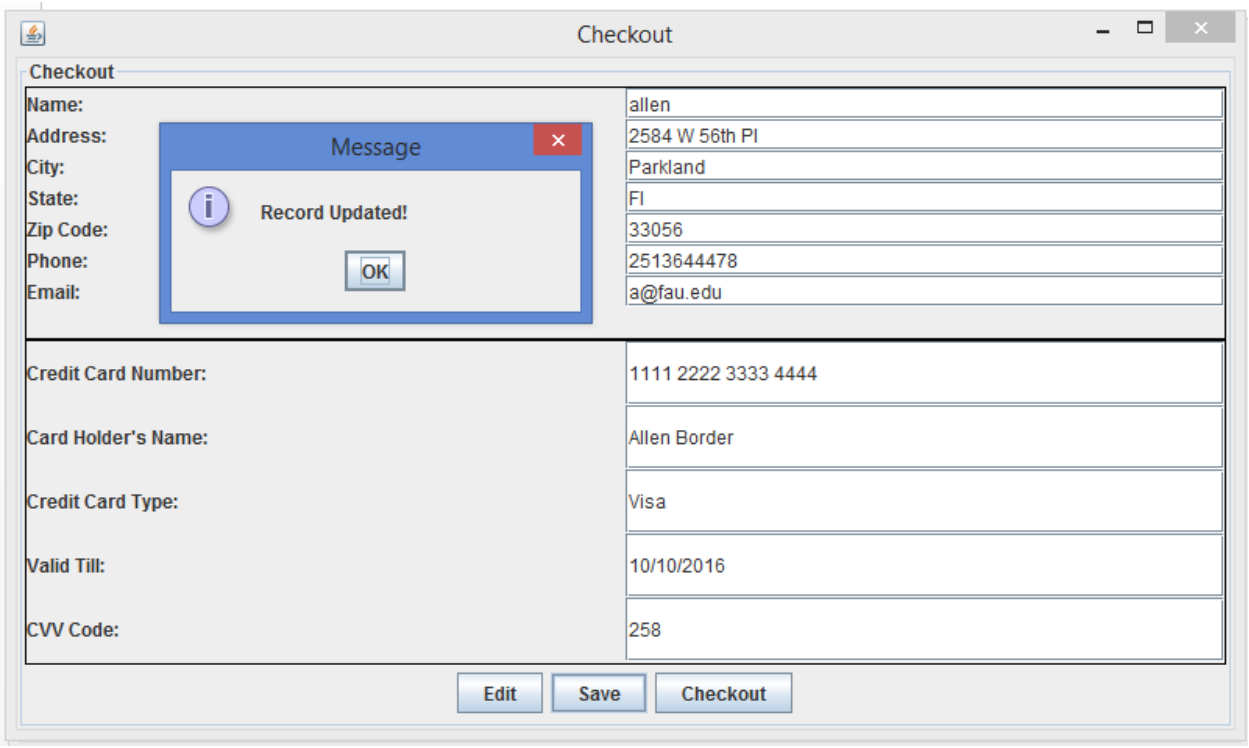


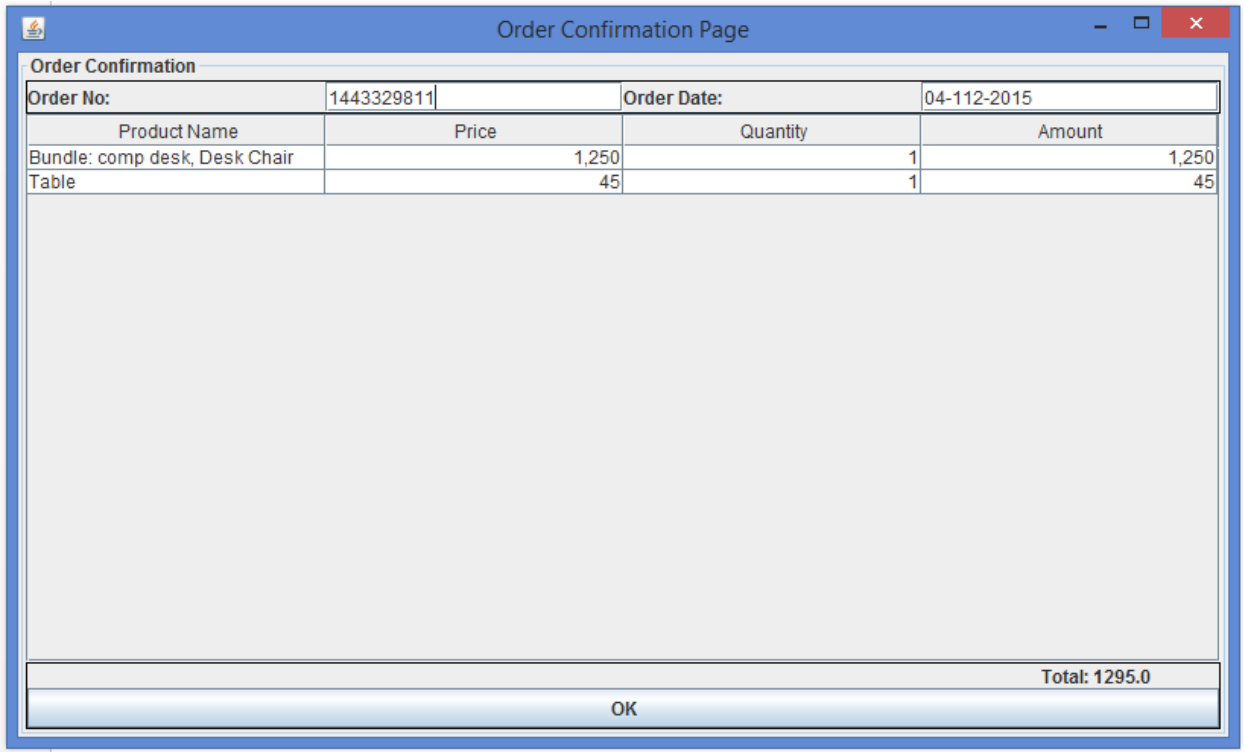
Use Case ID	UC4
Use Case Name	**Review/Update Shopping Cart**
Primary Actors	**Customer**
Basic Flow	1. Customer does **Add product to the cart** 2. Customer Clicks Review/Update Cart button, their cart is presented. 3. Total Price is displayed along with other purchased items information. 4. Customer clicks <<(previous) or >>(next button to see product one by one. 5. Customer changes quantity of an item if needed. 6. Customer presses the updateCart item. 7. Page refreshes, proper amount displayed, quantity updated message is given to user. 8. Customer is shown updated cart, with new Quantity and price.
Variation # 1 Invalid quantity.	6a. Customer is told that values are invalid due to inventory. 7a. Cart is presented to customer.
Variation #2	2b. Customer is notified if an item is no longer in stock if a vendor has removed/adjusted an item, cart functions as normal otherwise.

After adding two items I did a screen shot. It shows current cart amount in bold. For individual Item amount press << previous or next >> button.



Use Case ID	UC5
Use Case Name	**Checks out**
Primary Actors	**Customer**
Basic Flow	1. Customer does **Add an element to the cart** 2. Customer is at shopping cart. 3. Customer clicks check out. 4. Customer's profile of billing, shipping address and card info are displayed to verify. 5. User verifies, pays for the goods by supplying the credit card information on Customer Payment Page and order is complete. Inventory is adjusted
Variation # 1 Edit	4a. Start at step 4 5a Customer selects edit. 6a. Customer edits values. 7a. Customer selects update. 8a. Return to 4.

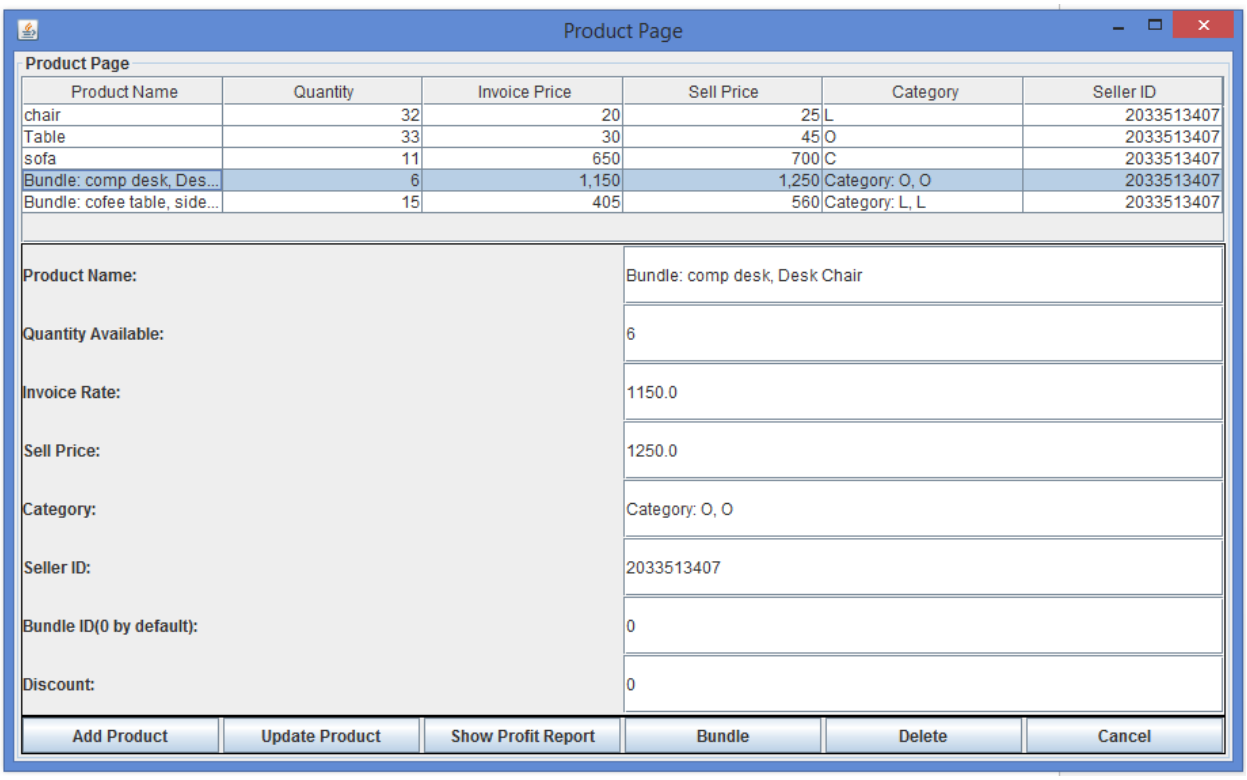




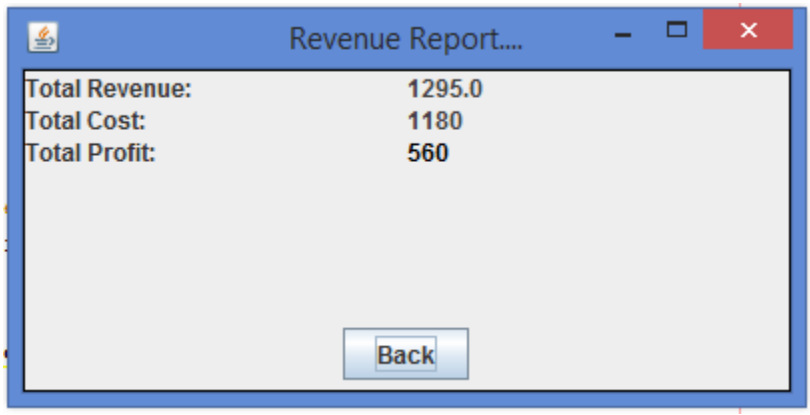
Use Case ID	**UC6**
Use Case Name	**Seller Review/Updates inventory**
Primary Actors	**Seller**
Basic Flow	1. Seller ***Login*** into the system. 2. Seller is on product review screen on which current inventory of product is displayed. 3. Seller changes the quantity of an item. 4. Seller presses update. 5. Page is refreshed, confirming results.
Variation # 1 Delete	2a. Seller selects to delete an item from the store. 3a. User is prompted to confirm deletion. 4a. User confirms 5a. Item is deleted.
Variation #2 Error invalid entry	4b. Seller receives error message. 5b. Return to 1.

Use Case ID	UC7
Use Case Name	**Seller Adds New Product**
Primary Actors	**Seller**
Basic Flow	1. Seller ***Login*** into the system 2. Seller is on product review screen on which current inventory of product is displayed. 3. Seller hits Add Item button 4. Seller is prompted to select product type and enter item information such as product/item name, invoice price, sell price and available quantity. 5. Seller presses Add Item button. 6. Item is added to inventory
Variation # 1 Invalid data.	5a. Item ID, Name, or other possible fields match current existing items, item is not added. 6a. Return to 1.

Use Case ID	UC8
Use Case Name	**Edit an item details**
Primary Actors	**Seller**
Basic Flow	1. Seller ***Login*** into the system. 2. Seller is on product/item review screen on which current inventory of product is displayed. 3. Seller has ability to edit information of item. 4. Seller selects update. 5. Item is updated.
Variation # 1 Invalid Data.	4a. Update fails, due to invalid invoice/price entry 5a. Return to 1.

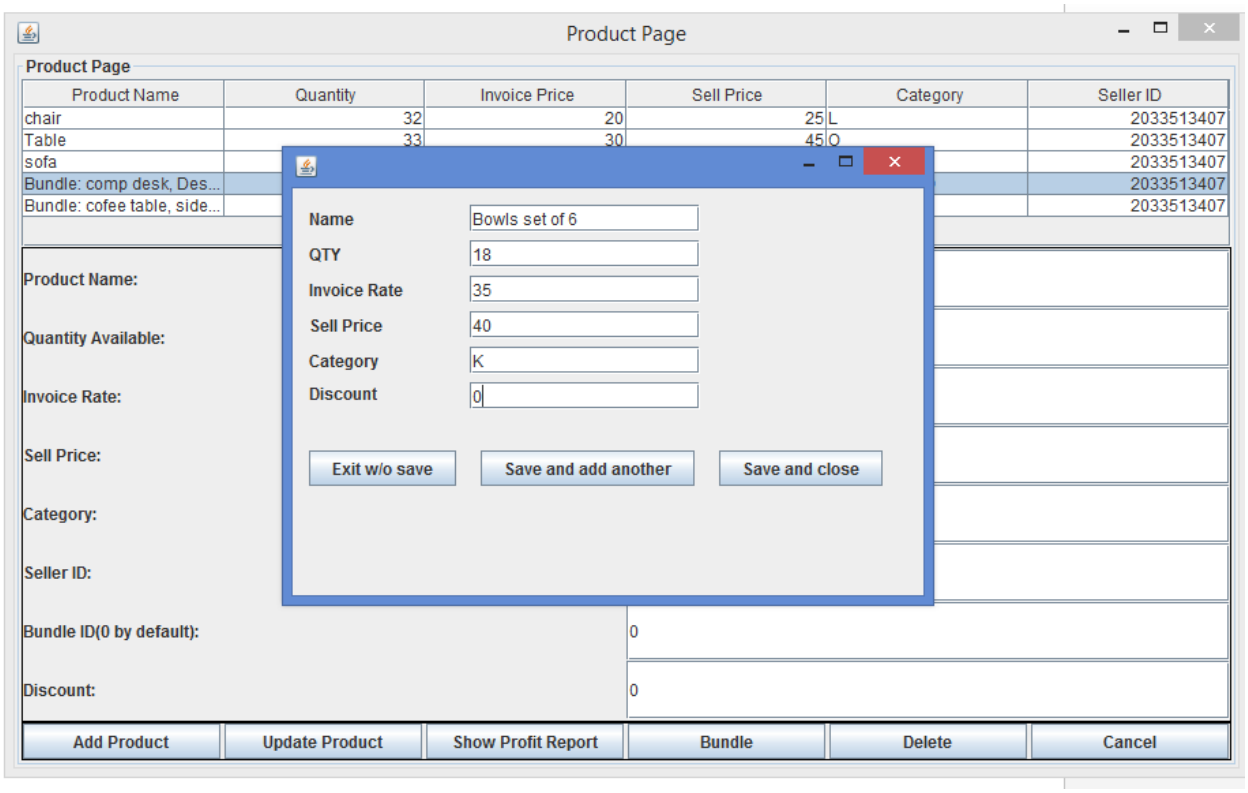


Use Case ID	UC9
Use Case Name	**Display Profit Report For Seller**
Primary Actors	**Seller**
Basic Flow	1. Seller ***Login*** into the system. 2. Seller is on product/item review screen on which current inventory of product is displayed. 3. Seller choose display profit report. 4. Window with profit details such as profits, revenues, cost (for that seller) will pop up.

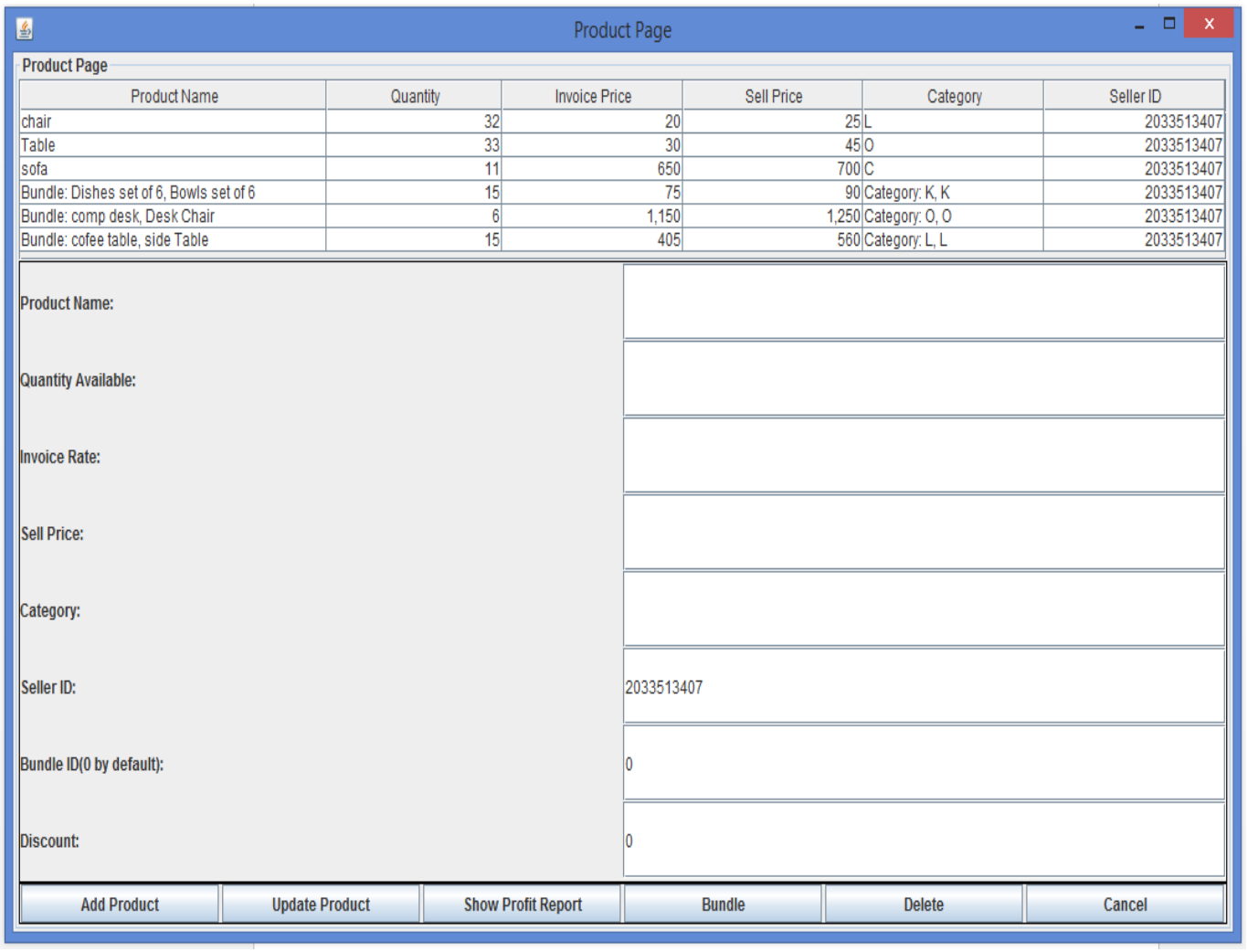


Use Case ID	UC10
Use Case Name	**Seller adding a bundle**
Primary Actors	**Seller**
Basic Flow	1. Seller presses **Bundle** button, BundleUI page gets displayed. 2. Seller inputs items into field, pressing **Save and Add Another** button 3. Seller presses **Save and Exit** when complete. 4. Seller is returned to sellerProductPage.

Screen shot how to add bundle



Screen that shows that whatever bundle we added is on the sellerProductPage in the table.


Product Page
Product Page
Product Name
Quantity
Invoice Price
Sell Price
Category
Seller ID
chair 32 20 25 L 2033513407
Table 33 30 45 O 2033513407
sofa 11 650 700 C 2033513407
Bundle: Dishes set of 6, Bowls set of 6 15 75 90 Category: K, K 2033513407
Bundle: comp desk, Desk Chair 6 1,150 1,250 Category: O, O 2033513407
Bundle: cofee table, side Table 15 405 560 Category: L, L 2033513407
Product Name:
Quantity Available:
Invoice Rate:
Sell Price:
Category:
Seller ID:
2033513407
Bundle ID(0 by default):
0
Discount:
0
Add Product
Update Product
Show Profit Report
Bundle
Delete
Cancel

CRC Cards

Class: ShoppingCartSystem	
Responsibility *Creates Buyer Seller and supports ShoppingCart objects for ShoppingCartSystem Functionality* *Validates login* *Gives only one instance wherever required.* *Initiates the database connection.*	**Collaboration** *Buyer* *Product* *Bundle* *DiscountedItem* *LineItem* *Seller* *ShoppingCart*

Class: ShoppingCart	
Responsibility *Calculating Total Price* *Displaying Total Price* *Maintains ShoppingCart Information*	**Collaboration** *Buyer* *Product* *Bundle* *DiscountedItem* *LineItem*

Class: UsersOfSystem	
Responsibility *Saves Basic User data (f/l name, acc type, etc)* *Used as basis for buyer/seller*	**Collaboration** *ShoppingCartSystem* *Buyer* *Seller*

Class: Buyer	
Responsibility *Add LineItem To Cart* *Remove LineItem From Cart* *Begin Checkout* *Interact With Catalog*	**Collaboration** *ShoppingCartSystem* *LineItem* *ShoppingCart* *Product* *Bundle* *DiscountedItem*

Class: Seller	
Responsibility *Manage Personal Inventory* *Allows review of sales data*	**Collaboration** *ShoppingCartSystem* *Inventory* *SellerProductPage* *RevenueReportUI*

Class: Inventory	
Responsibility *Maintain Seller LineItems* *Manage add, edit, delete Inventory Line items*	**Collaboration** *Seller* *Product* *Bundle* *DiscountedItem* *LineItem*

Class: LineItem is an interface	
Responsibility *Keeps information of Items-description, price*	**Collaboration** *Product* *Bundle* *DiscountedItem* *LineItem*

Class: Product	
Responsibility *Contains details of item added by seller(price descript, etc)* *acting as a global inventory for buyer and cart.*	**Collaboration** *Inventory* *LineItem*

Class: Bundle	
Responsibility *Displaying data for interface and acting as a global inventory for buyer and cart.*	**Collaboration** *LineItem* *Product* *DiscountedItem*

Class: BundleUI	
Responsibility *Provides UI for inputting a bundle.*	**Collaboration** *LineItem* *Inventory*

Class: BundleUIListener	
Responsibility *Understands user interaction with bundle UI*	**Collaboration** *BundleUI* *Inventory* *LineItem*

Class: OrderConfirmationUI	
Responsibility *Confirms an order to the buyer once checkout is done*	**Collaboration** *LineItem* *Buyer* *Inventory*

Class: Discounted Item	
Responsibility *Stores discount information* *Applies discount to line item*	**Collaboration** *LineItem*

Class: LoginUI	
Responsibility *Provides user interface for username password entry. Also provides buttons for passing user events to ShoppingCartSystem*	**Collaboration** *ShoppingCartSystem* *LoginListener* *NewBuyerListener* *NewSellerListener* *LoginCancelListener*

Class: LoginListener- anonymous class	
Responsibility *Passes user interactions to the ShoppingCartSystem for checking login validations*	**Collaboration** *LoginUI* *ShoppingCartSystem*

Class: NewSellerListener- anonymous class	
Responsibility *Displays the User Profile window for seller to enter*	**Collaboration** *UseProfileUI*

Class: UserProfileUI	
Responsibility *Gives UI for user to enter his information*	**Collaboration** *saveProfileListener* *UpdateProfileListener* *ContinueListener* *UsersofSystem*

Class: NewBuyerListener- anonymous class	
Responsibility *Displays the User Profile window for buyer to enter his info.*	**Collaboration** *UseProfileUI*

Class: LoginCancelListener- anonymous class	
Responsibility *To cancel user interaction with the ShoppingCartSystem*	**Collaboration**

Class: saveProfileListener - anonymous class	
Responsibility *Passes user interactions to the ShoppingCartSystem to display browse catalog screen*	**Collaboration** *ShoppingCartSystem* *UsersofSystem* *UserProfileUI*

Class: UpdateProfileListener- anonymous class	
Responsibility *Passes user interactions to the ShoppingCartSystem for updating the user information*	**Collaboration** *ShoppingCartSystem* *UsersofSystem* *UserProfileUI*

Class: ProductCatalogUI	
Responsibility *Display available products to customer to shop*	**Collaboration** *LineItem* *BrowsePanelClickListener*

Class: ProductCatalogUI	
Responsibility *UI that provides buyer to buy products*	**Collaboration** *addToCartListener* *ReviewUpdateCardListener*

Class: ContinueListener- anonymous class	
Responsibility *Passes user interactions to the ShoppingCartSystem to once the user information is added or updated*	**Collaboration**

Class: *BrowsePanelClickListener*- anonymous class	
Responsibility Display the right product page to user according to userclick action on panel	**Collaboration** *LineItem* *ProductPageUI*

Class: *addToCartListener*- anonymous class	
Responsibility handles user interaction of adding items to cart	**Collaboration** *ShoppingCart* *LineItem* *ProductCatalogUI*

Class: *ReviewUpdateCartListener*- anonymous class	
Responsibility *Review/UpdateCart* UI is displayed for user	**Collaboration** *ReviewUpdateCartUI* *ShoppingCart*

Class: *ReviewUpdateCartUI*	
Responsibility Provides user UI to update cart	**Collaboration** *updateCartListener* *returnListener* *UsersofSystem* *Buyer* *PlaceOrderListener*

Class: *updateCartListener* - anonymous class	
Responsibility Updates qty to be purchased	**Collaboration** *ShoppingCart* *ReviewUpdateCartUI*

Class: *returnListener*- anonymous class	
Responsibility *Displays ProductCatalogUI i.e Productpage for user to continue shopping*	**Collaboration** *LineItem* *ProductCatalogUI*

Class: *PlaceOrderListener* - anonymous class	
Responsibility *Displays checkout screen for buyer to enter credit card information and verifying his profile information* *Buyer can edit his profile information here as well.*	**Collaboration** *LineItem* *UsersofSystem* *Buyer* *checkOutUI*

Class: editProfilebtnListener- anonymous class	
Responsibility User interaction of editing profile is handled	**Collaboration** *UsersofSystem* *Seller* *Buyer* *checkOutUI*

Class: saveProfilebtnListener- anonymous class	
Responsibility User interaction saving profile information such as credit card or any updated user information is saved	**Collaboration** *UsersofSystem* *Seller* *Buyer* *checkOutUI*

Class: CheckOutUI	
Responsibility *Polls the user for updated profile information before continuing transaction.*	**Collaboration** *ShoppingCart* *Buyer*

Class: ProductsForSaleUI -sellerUI	
Responsibility Provides seller UI to add, update, delete products for sale. Provides feature of show profit report Adjusts inventory according to seller transaction.	**Collaboration** *addProductListener* *UpdateProductListener* *DeleteProductListener* *ShowProfitListener* *LineItem*

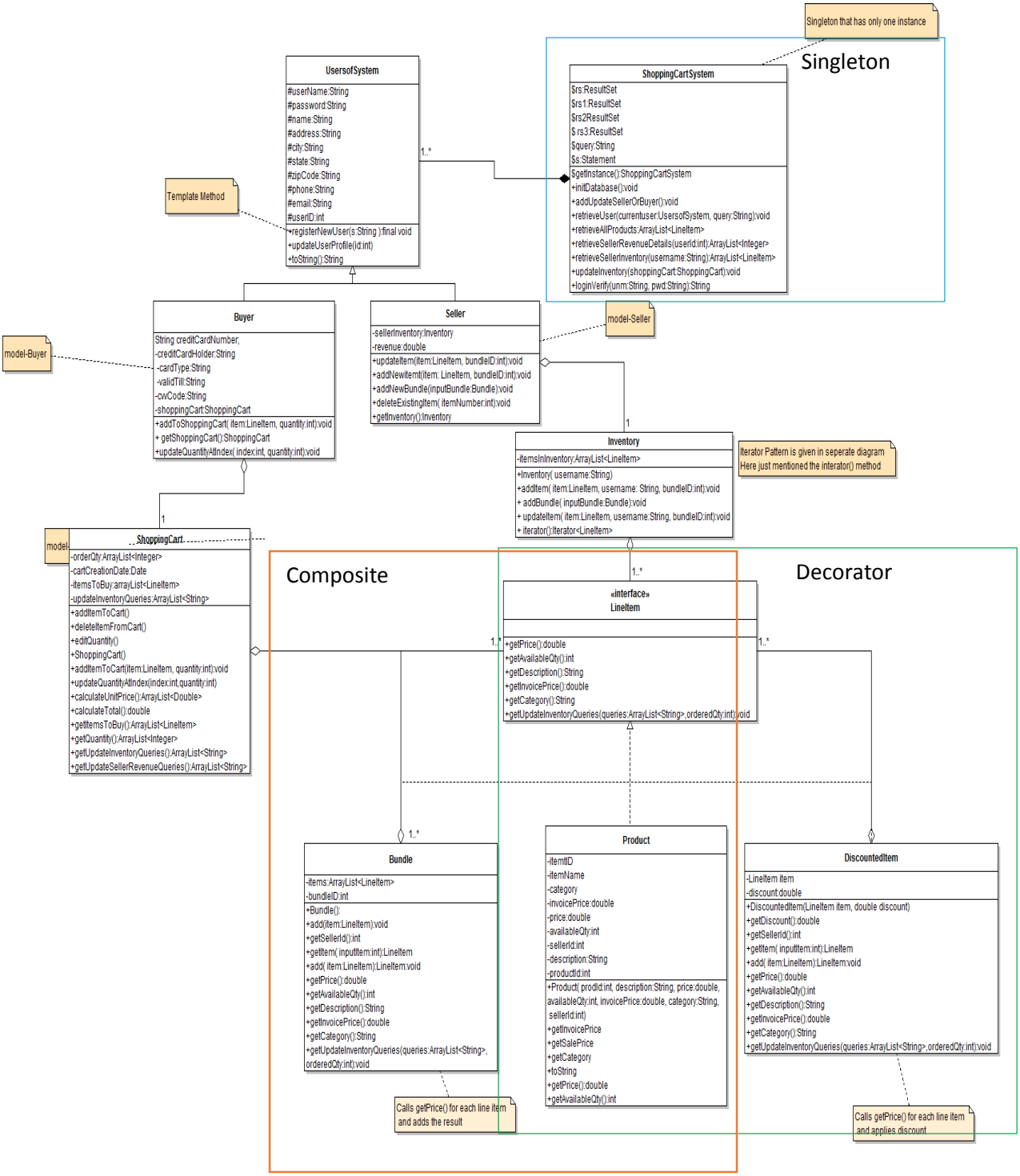
Class: CheckOutListener - anonymous class	
Responsibility *Displays Order confirmation* *Adjusts Inventory*	**Collaboration** *LineItem* *checkOutUI*

Class: *ReviewUpdateCartUI*	
Responsibility *Provides user UI to review andupdate cart*	**Collaboration** *updateCartListener* *returnListener* *UsersofSystem* *Buyer* *PlaceOrderListener*

Class: RevenueReportUI	
Responsibility *Calculating Total cost, revenue, profit* *Displaying Total cost, revenue, profit* *Maintains ShoppingCart Information*	**Collaboration** *Seller* *Inventory* *LineItem* *ShoppingCart*

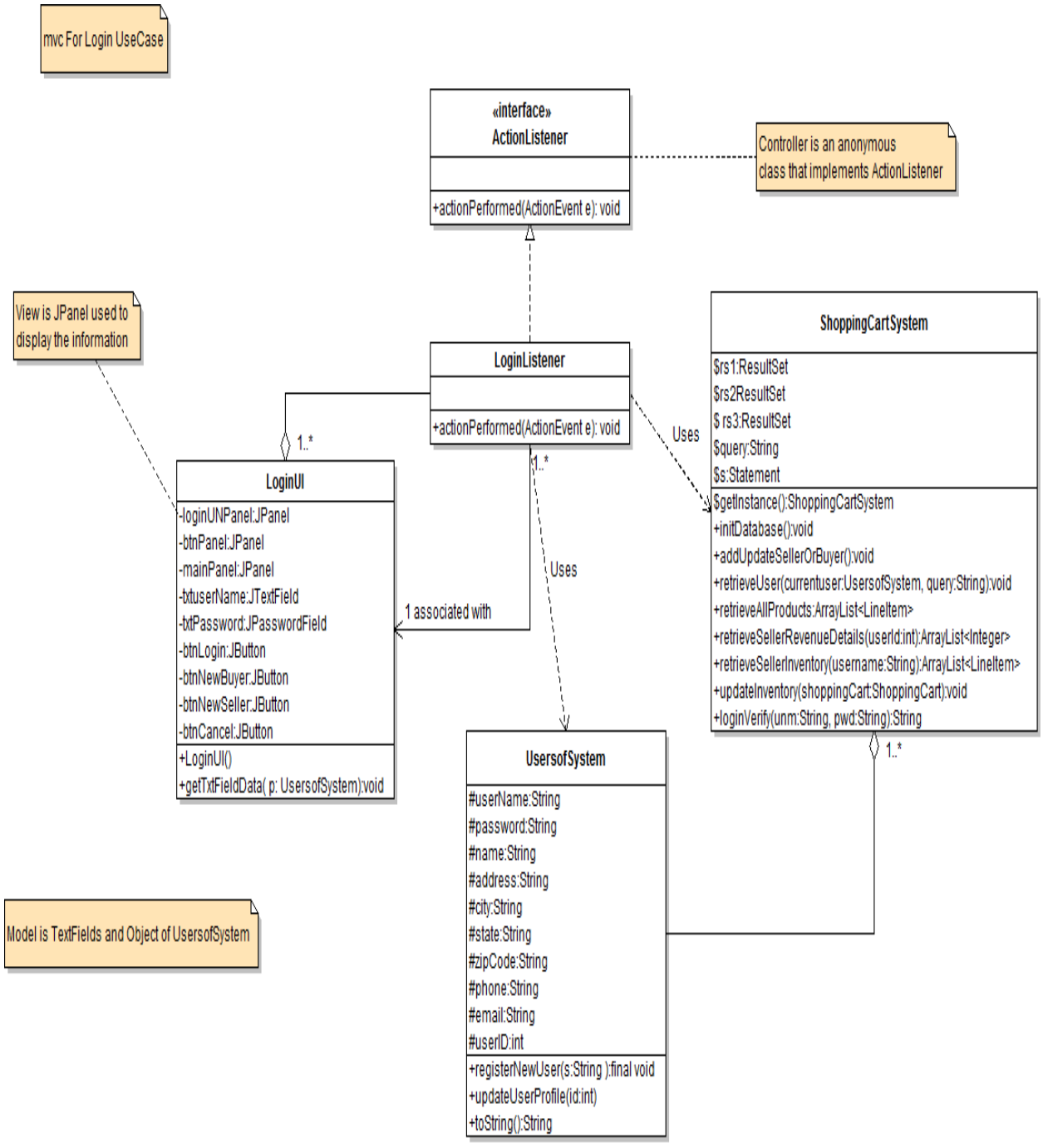
UML Diagrams:

Class Diagram For Shopping Cart System

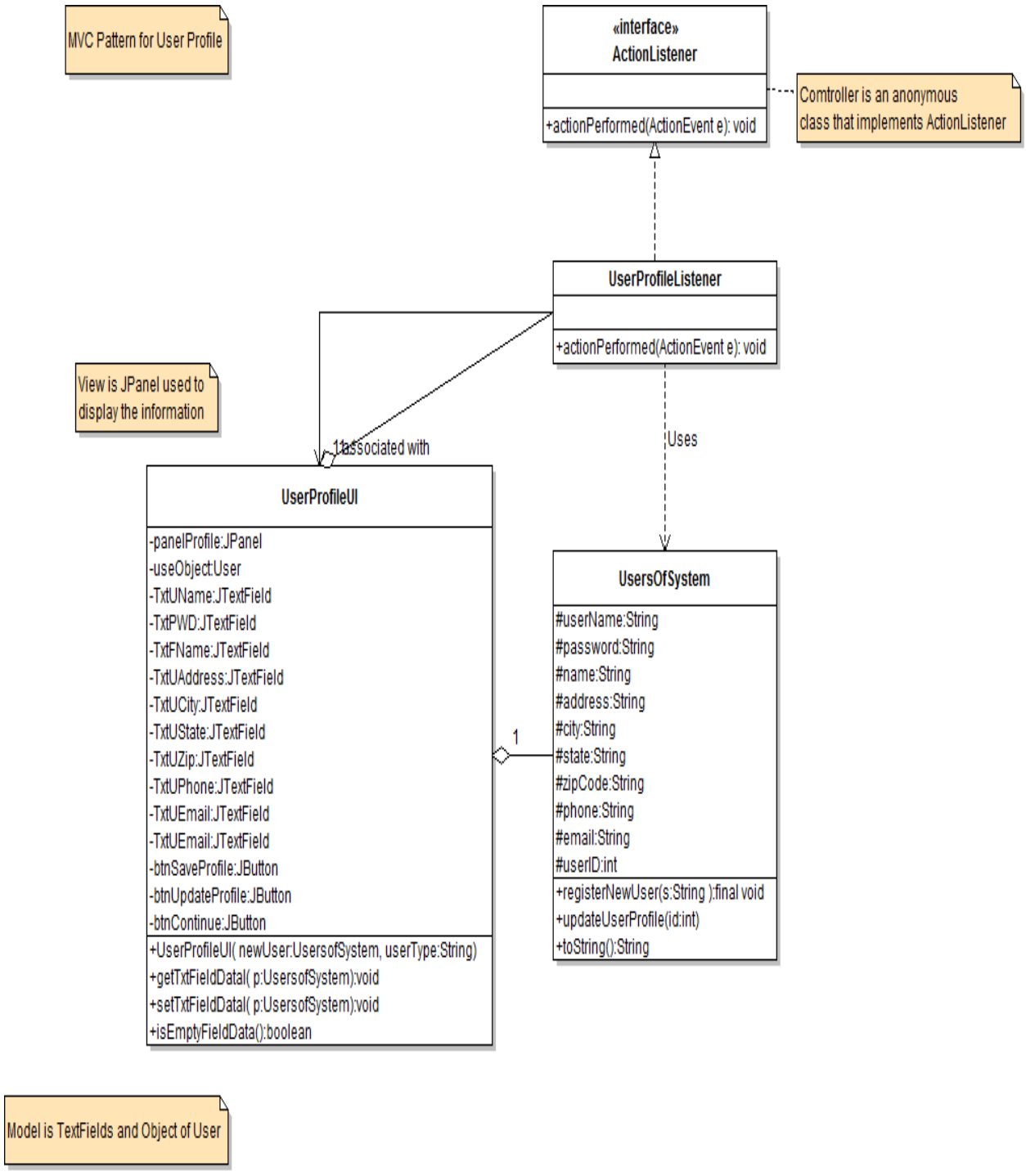


MVC Pattern UML Diagrams

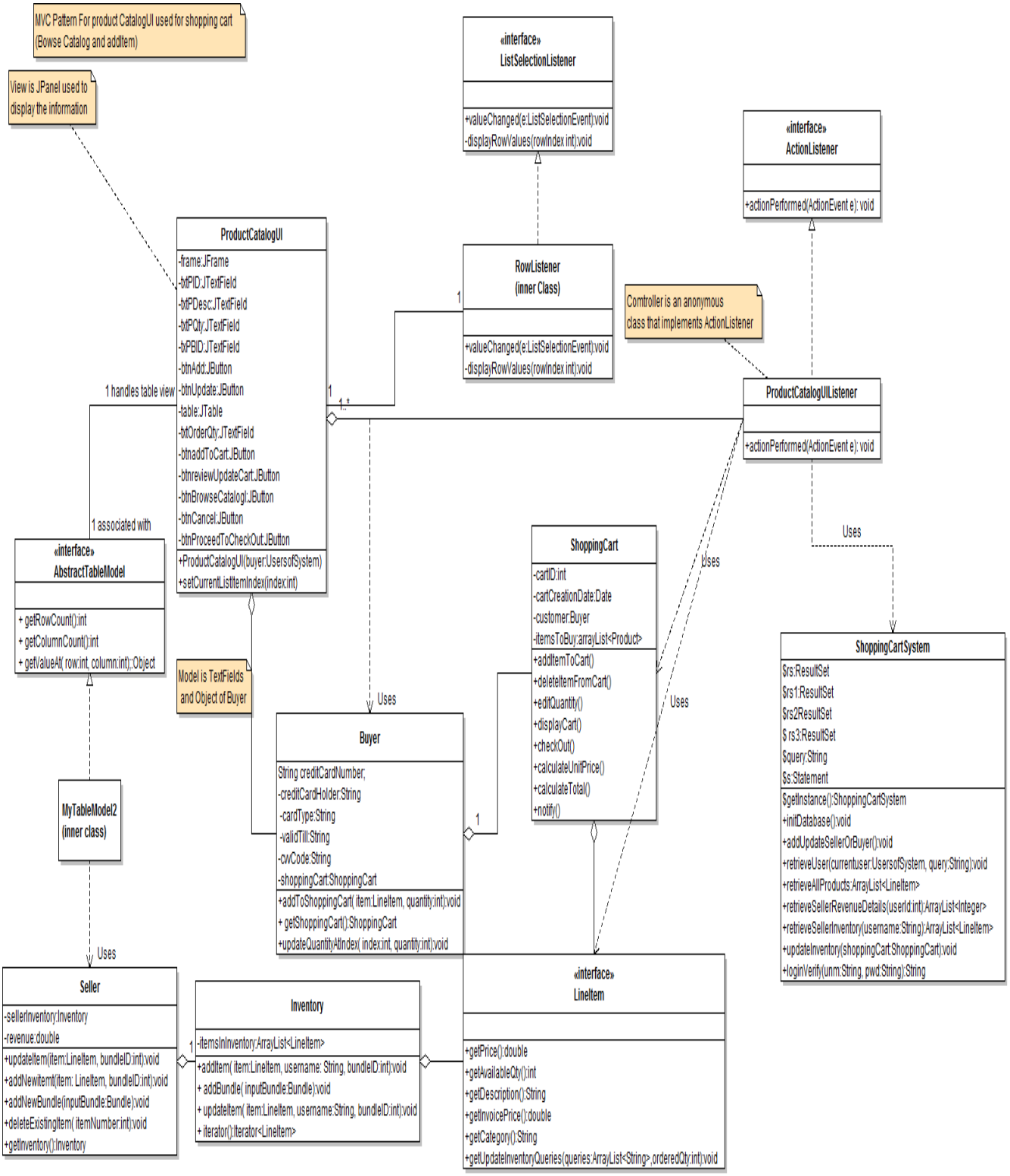
Mvc for login usecase:



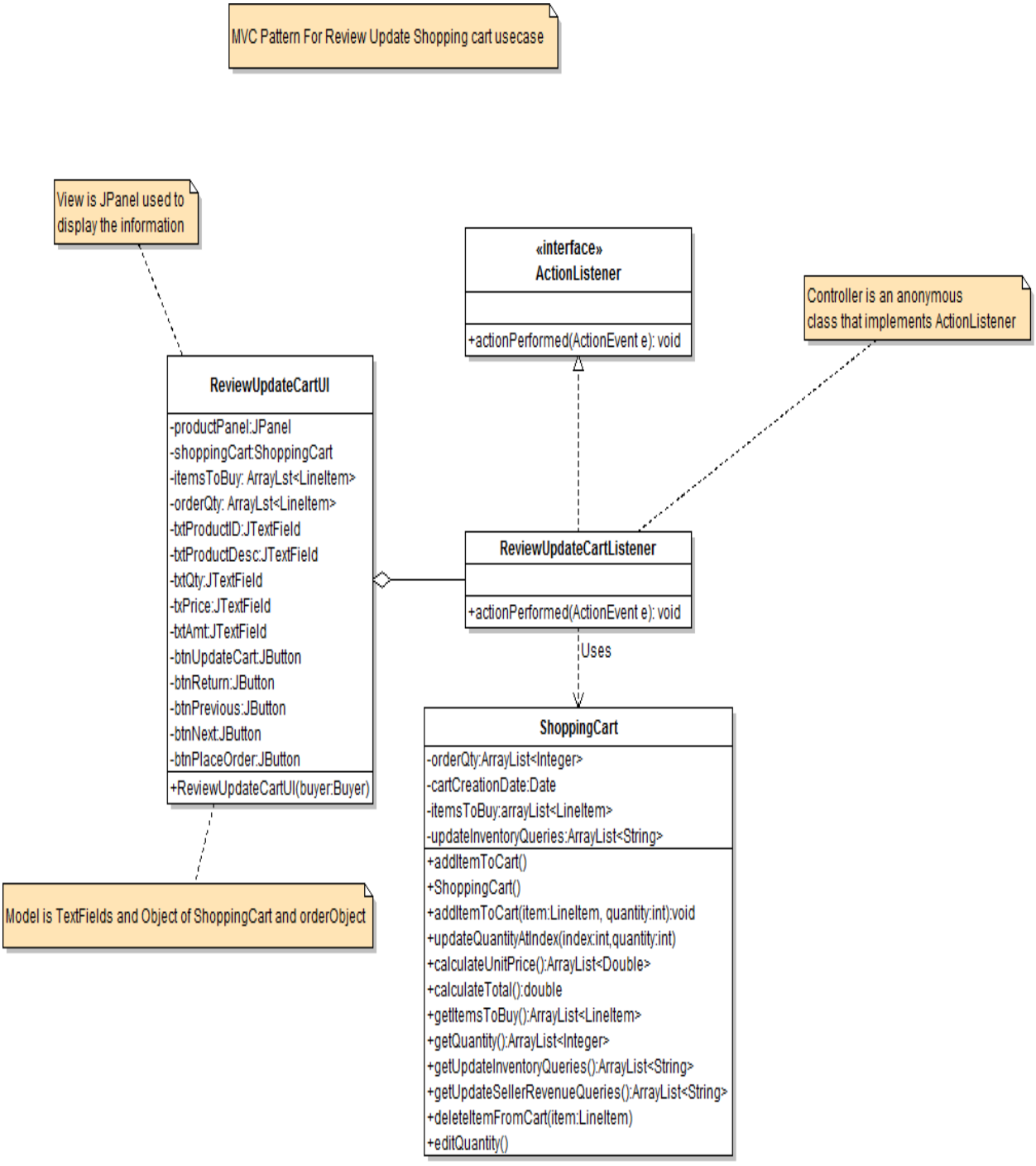
Mvc for User Profile:



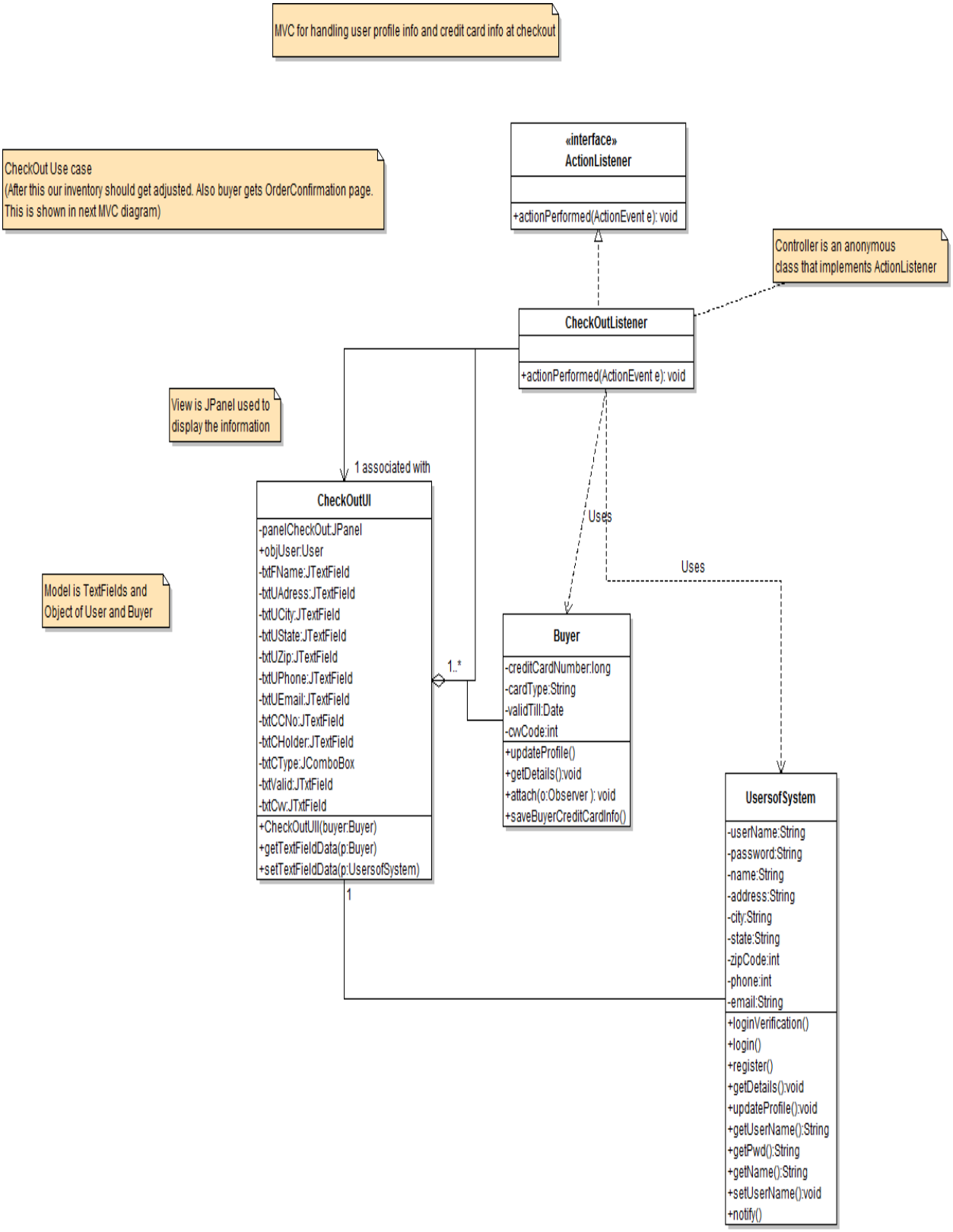
MVC Browse Catalog and add item



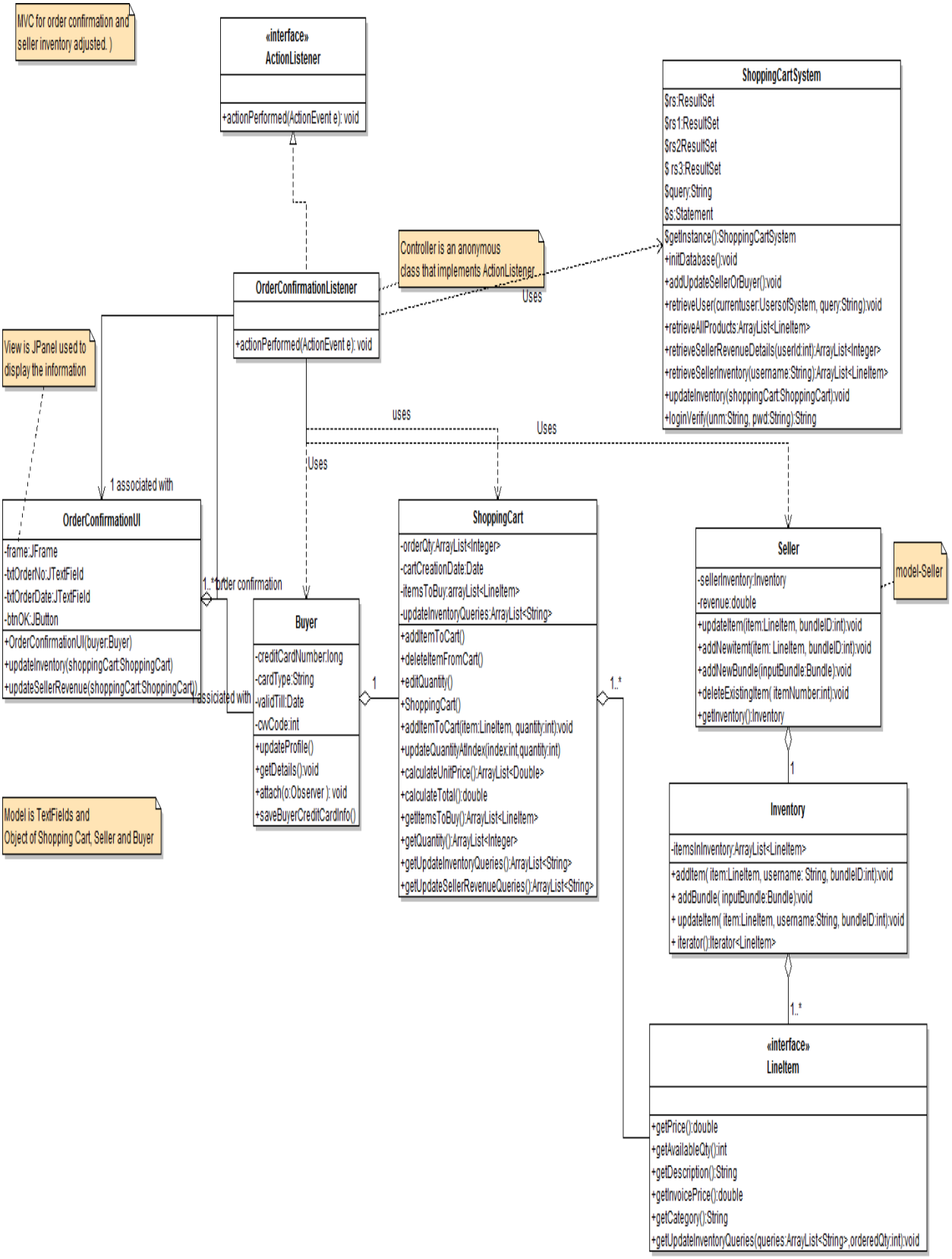
MVC Review Update Shopping Cart



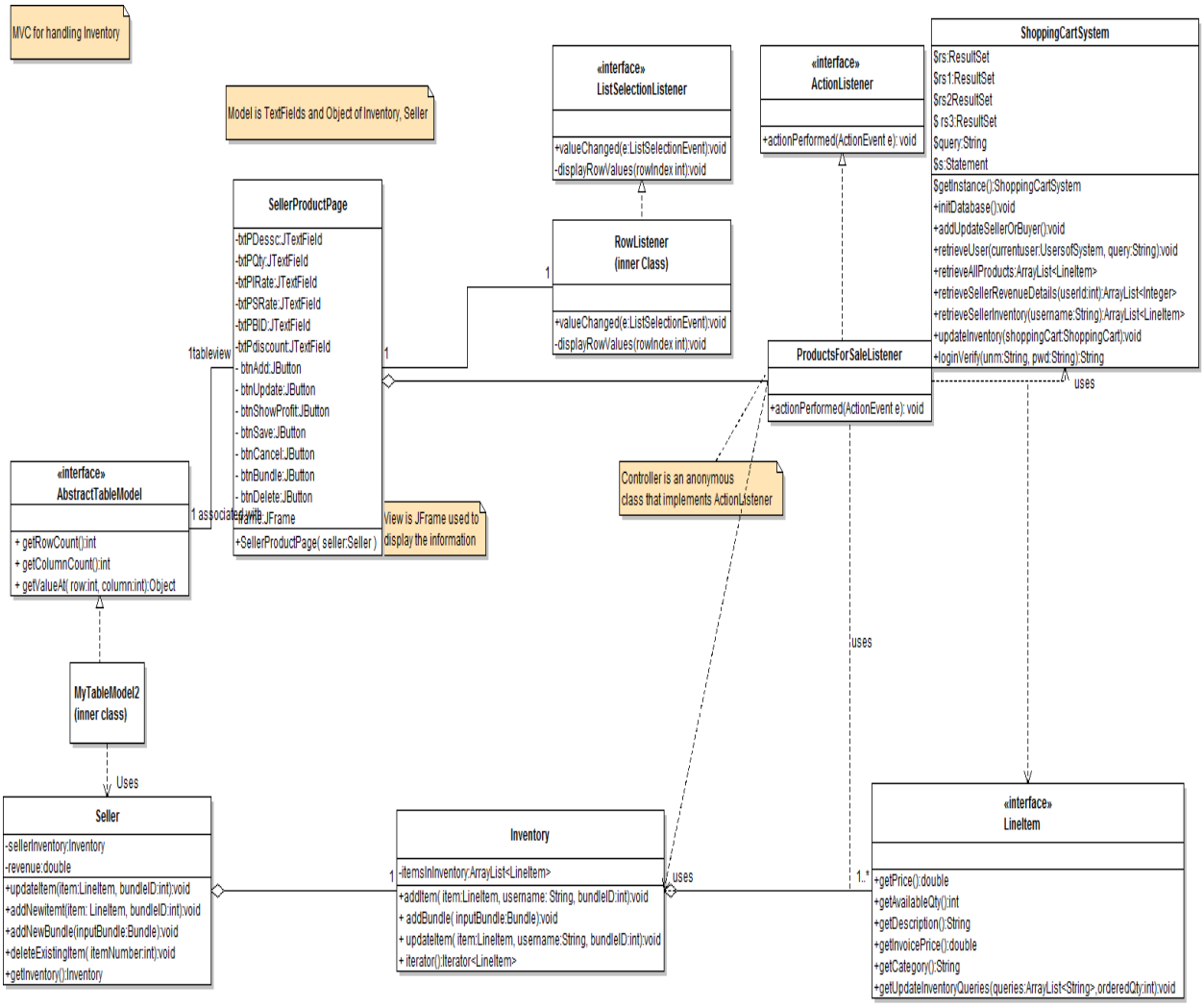
MVC for handling user profile info and credit card info at checkout



MVC order Confirmation

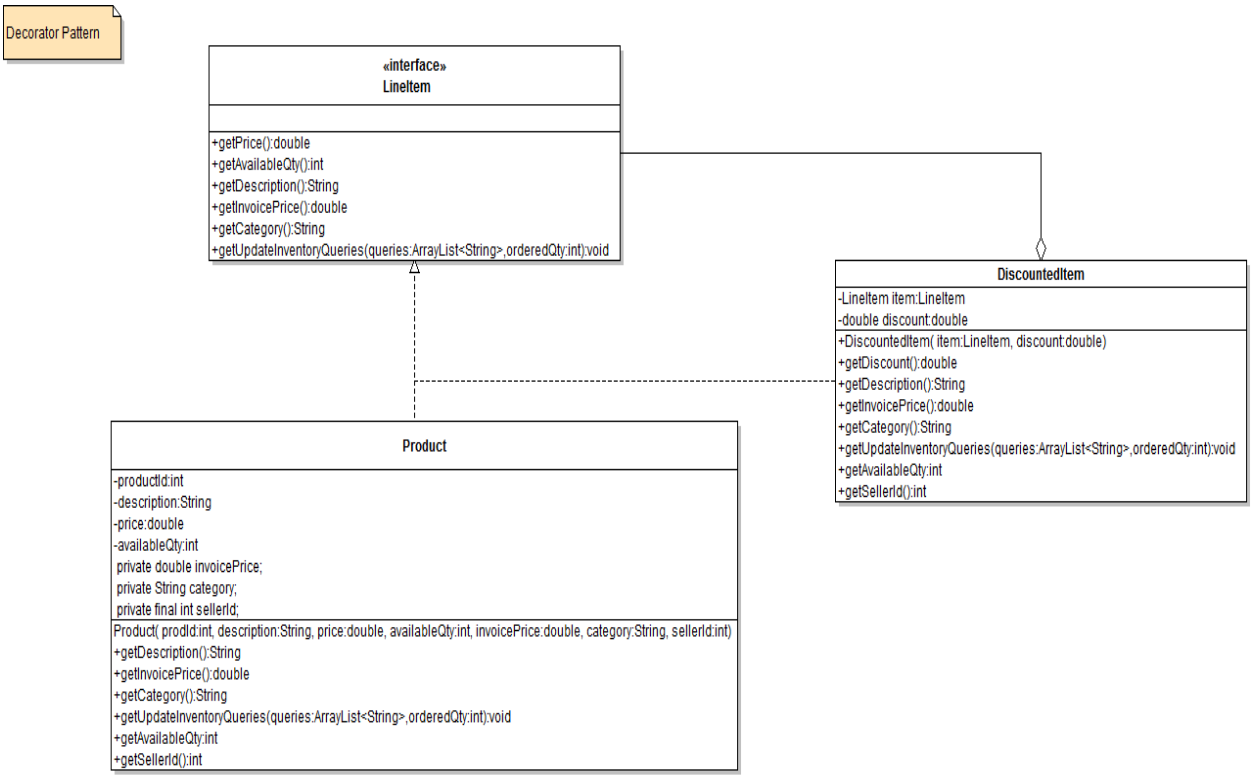


MVC For handling inventory

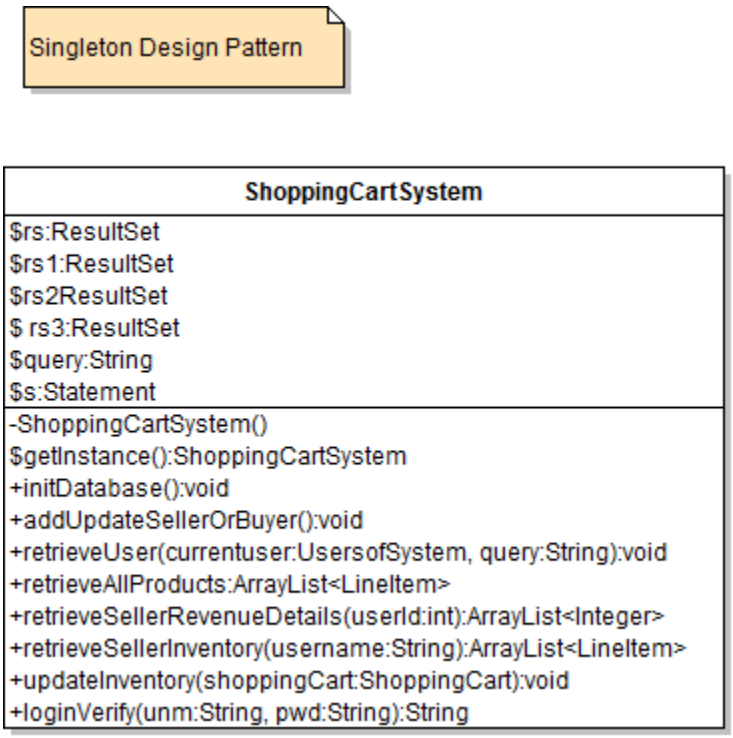


Design Pattern Class Diagrams:

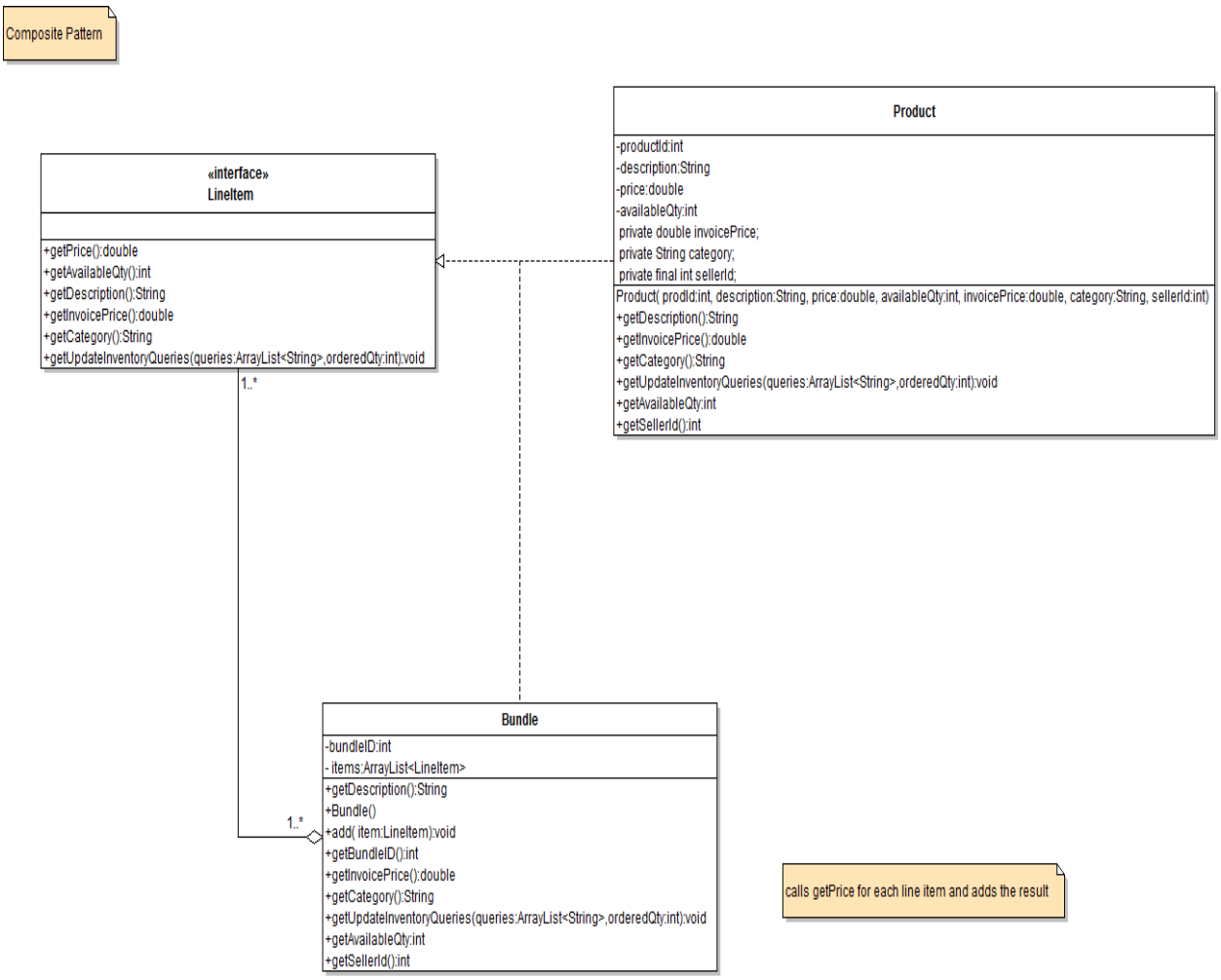
Decorator Pattern



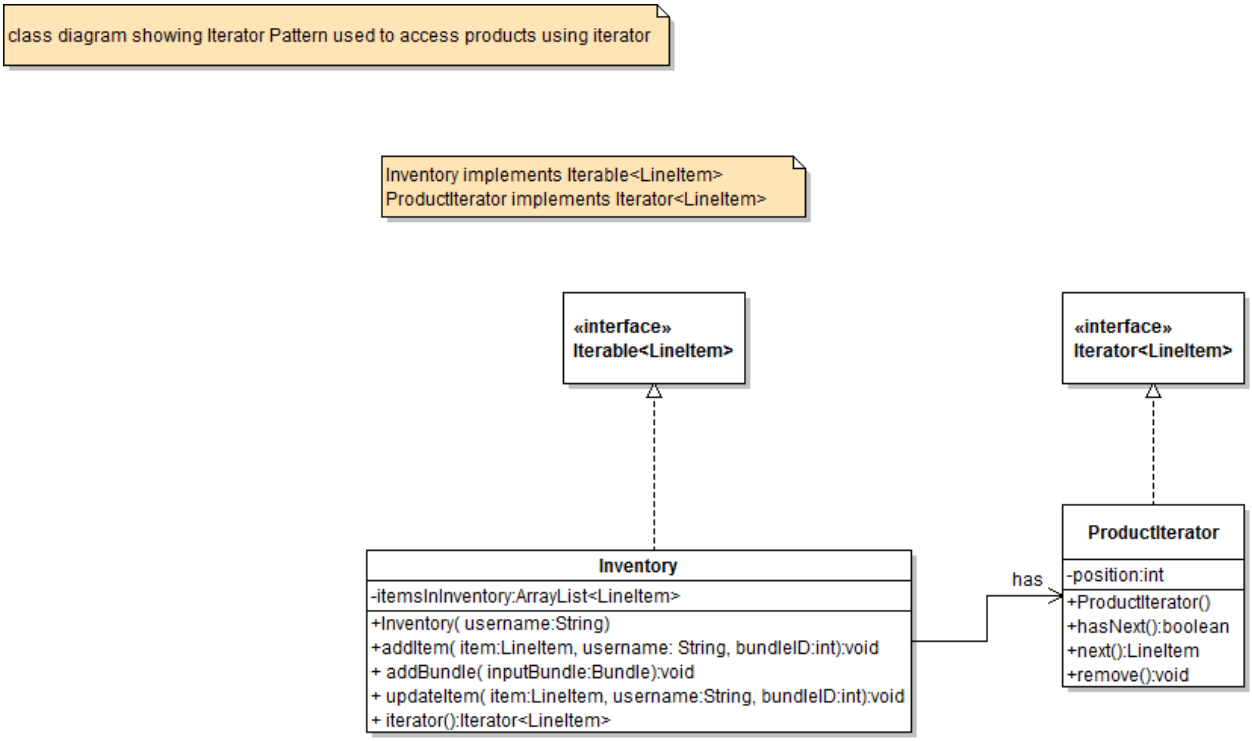
Singleton Pattern



Composite Pattern



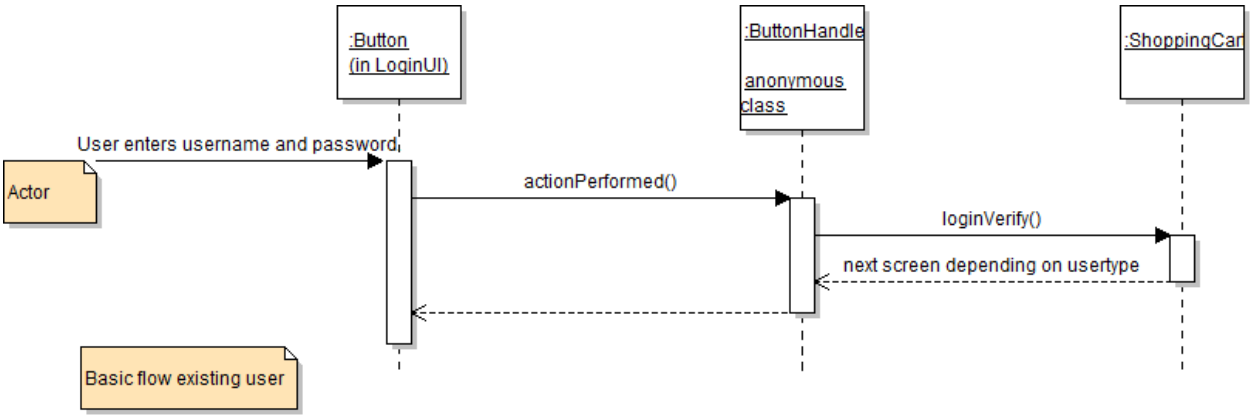
Iterator Pattern



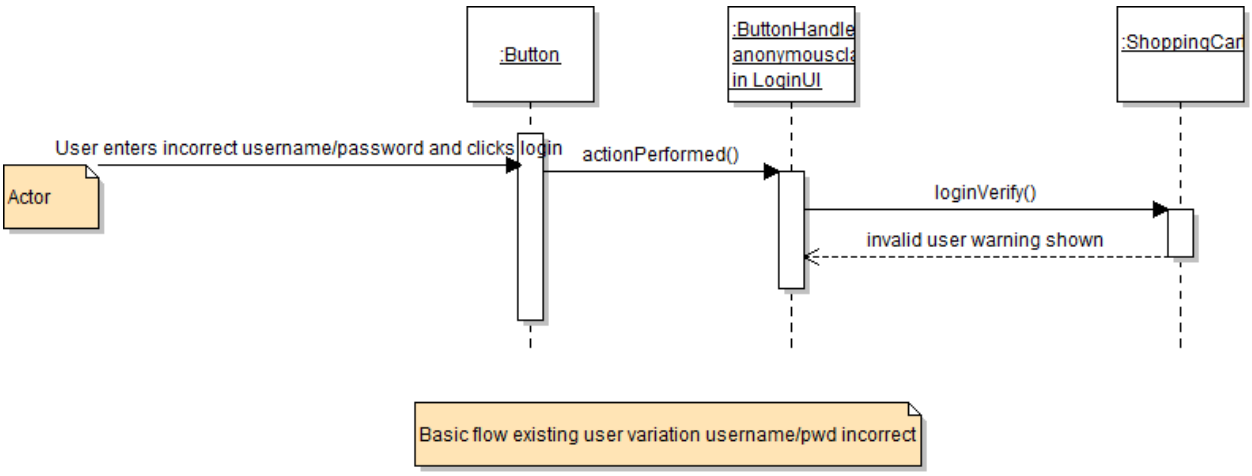
Sequence Diagrams:

Use Case Login

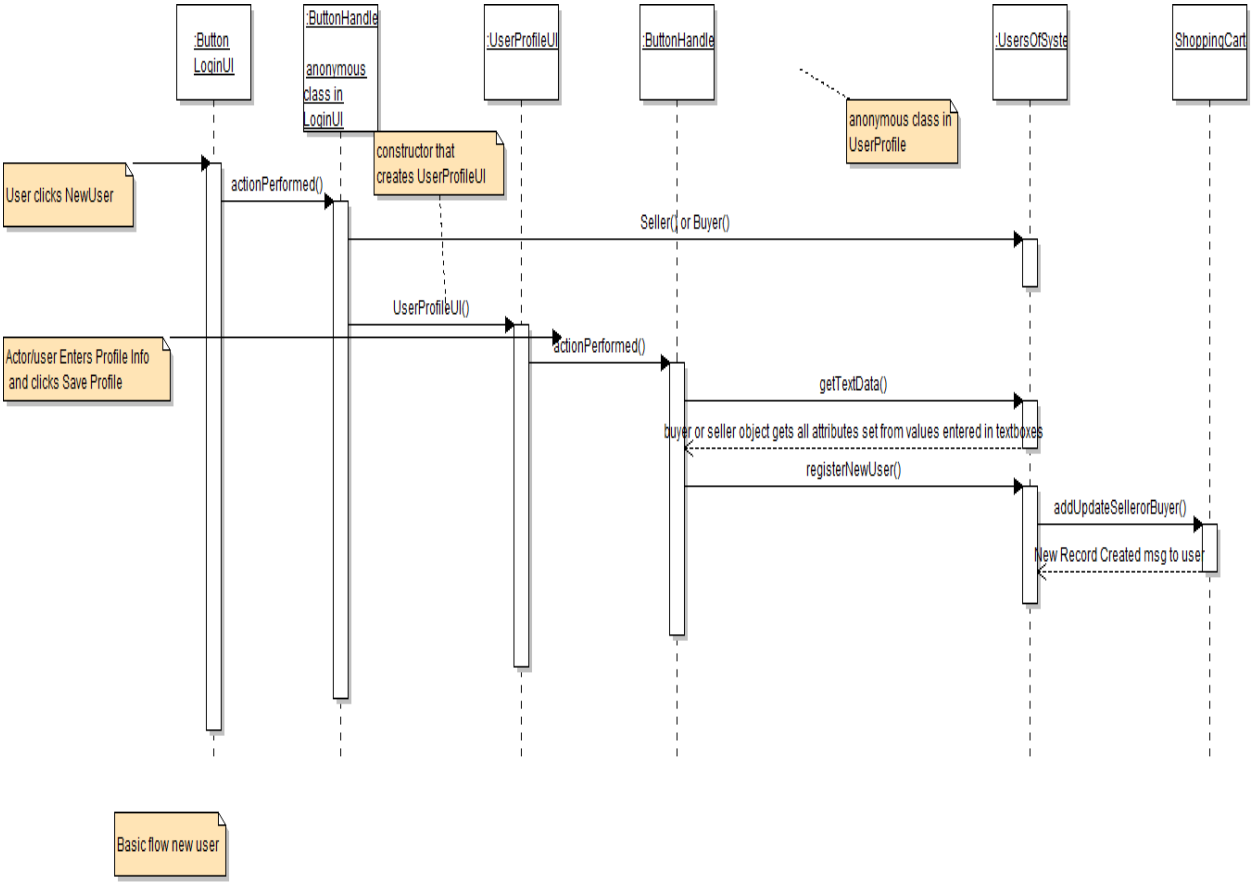
Basic flow existing user



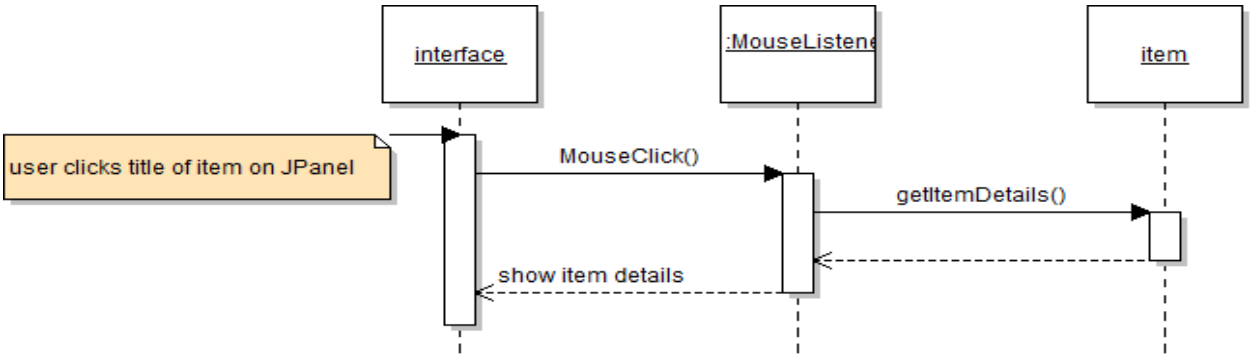
Basic flow existing user variation username/pwd incorrect



Basic flow new user

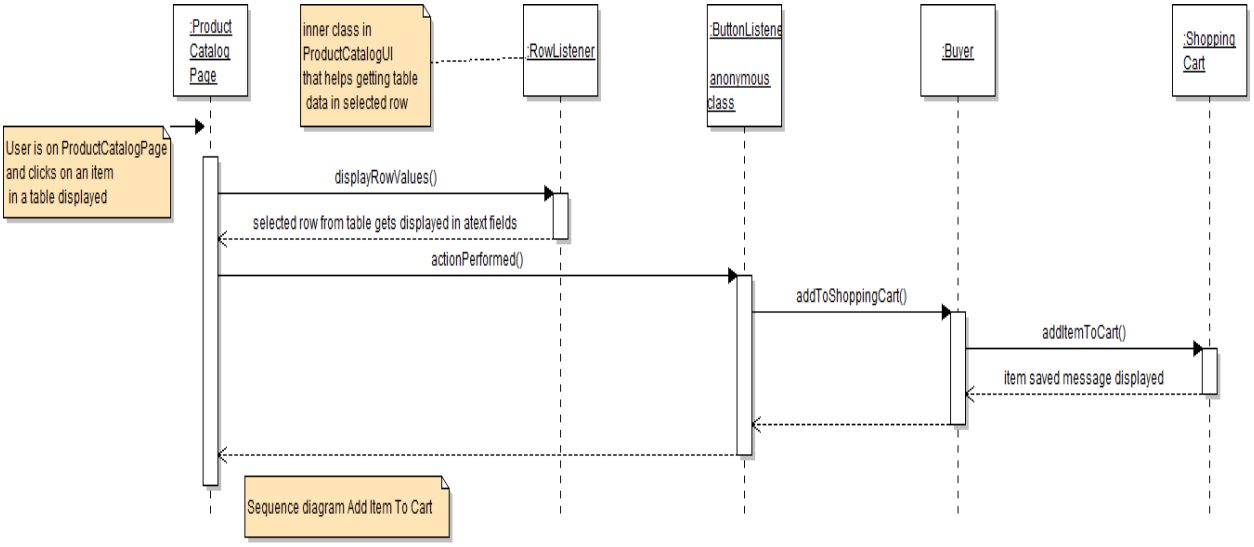


Review product details

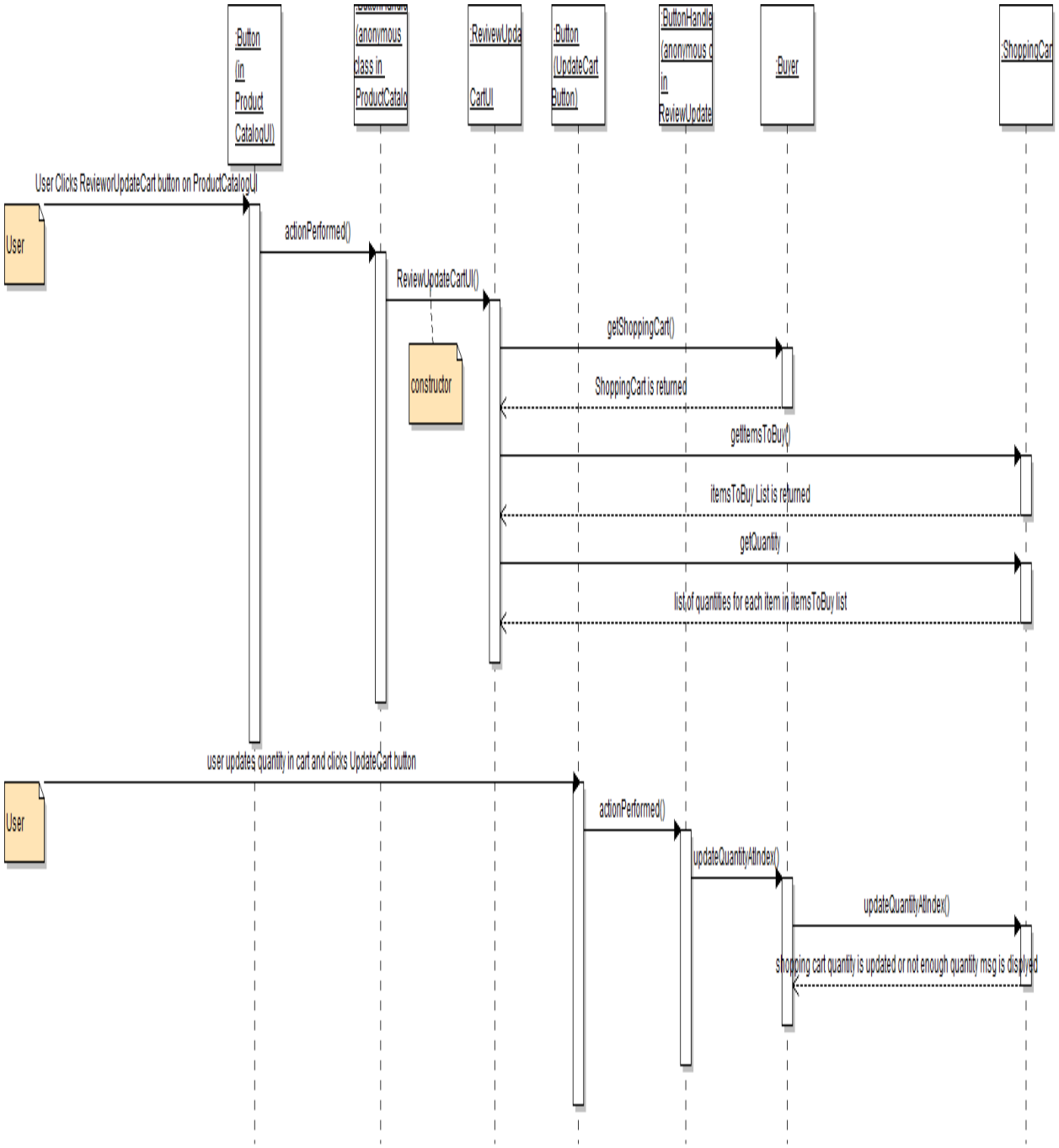


Product Page

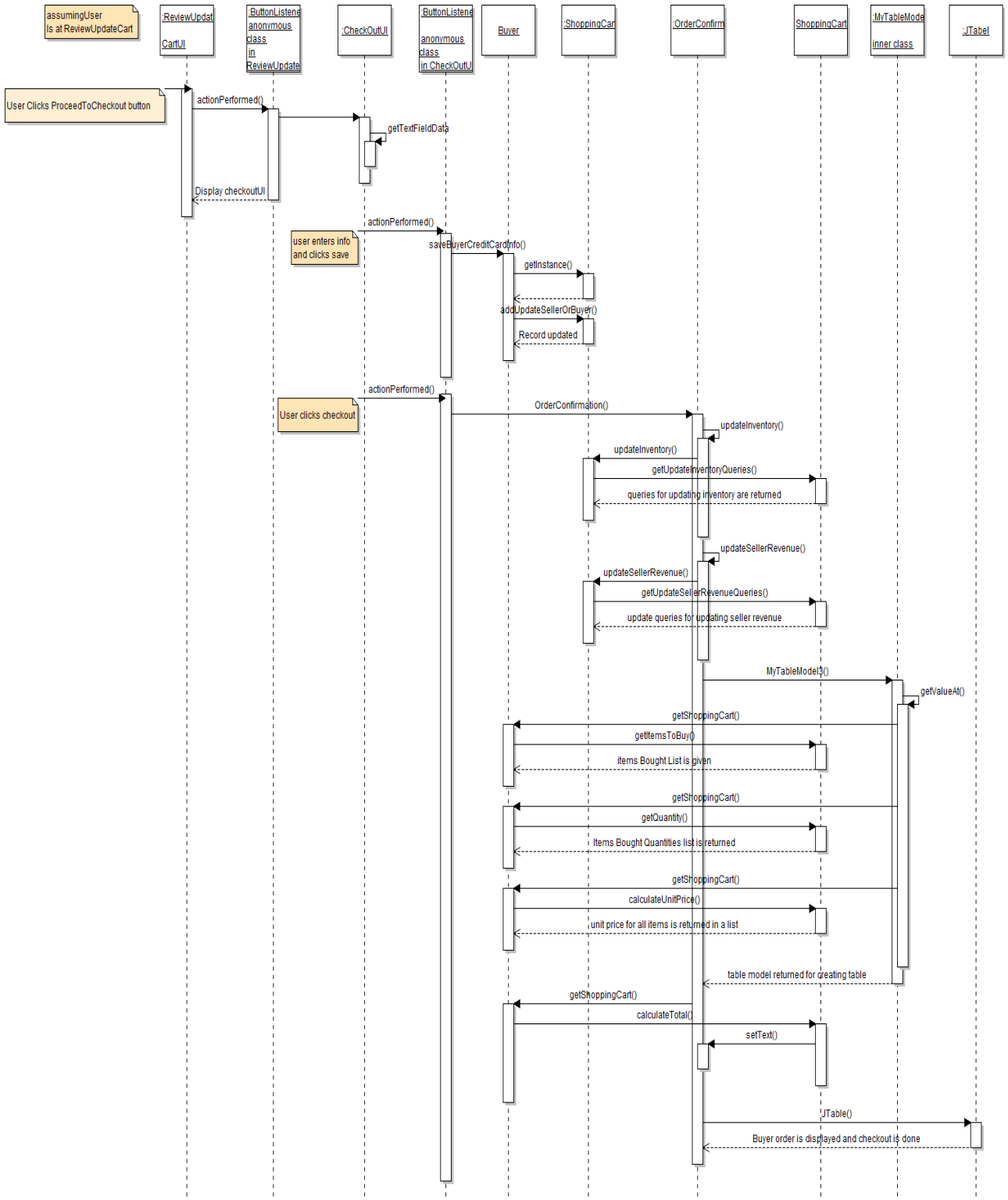
Add Item to Cart



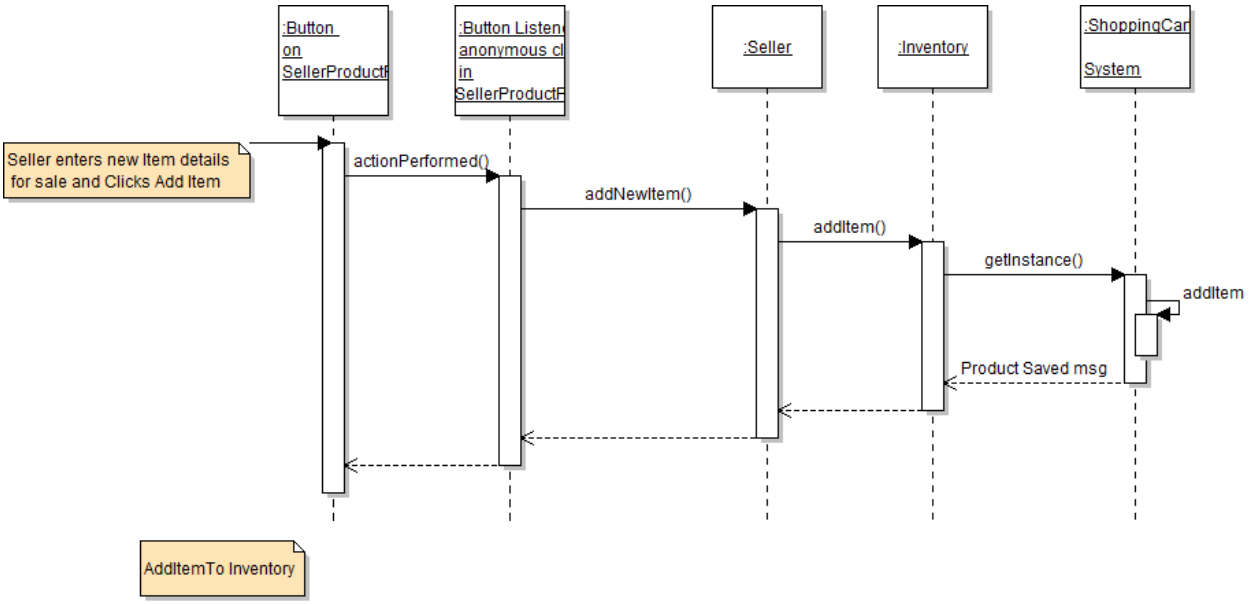
Review/update cart



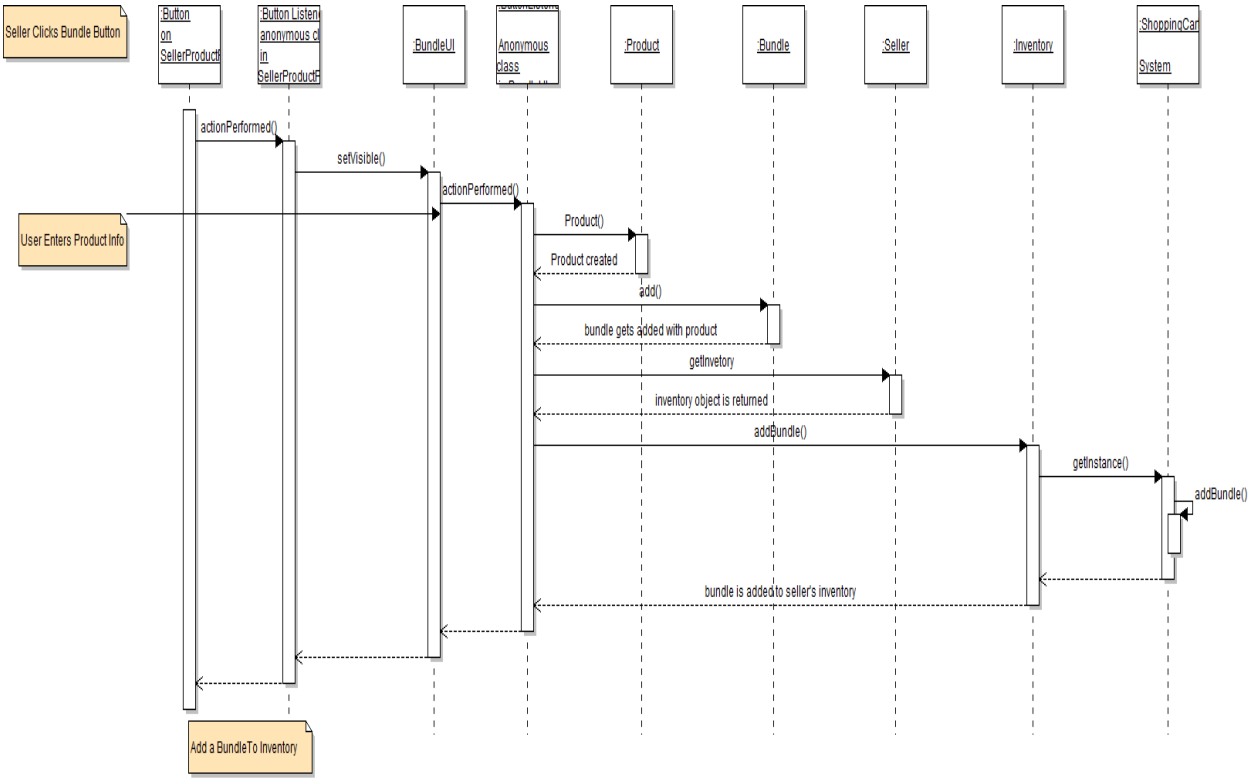
Checkout


assumingUser
Is at ReviewUpdateCart
:ReviewUpdat
CartUI
:ButtonListene
anonymous
class
in
ReviewUpdate
:CheckOutUI
:ButtonListene
anonymous
class
in CheckOutU
Buyer
:ShoppingCar
:OrderConfirm
ShoppingCart
:MyTableMode
inner class
:JTabel
User Clicks ProceedToCheckout button
actionPerformed()
getTextFieldData
Display checkoutUI
actionPerformed()
user enters info
and clicks save
saveBuyerCreditCardInfo()
getInstance()
addUpdateSellerOrBuyer()
Record updated
actionPerformed()
User clicks checkout
OrderConfirmation()
updateInventory()
updateInventory()
getUpdateInventoryQueries()
queries for updating inventory are returned
updateSellerRevenue()
updateSellerRevenue()
getUpdateSellerRevenueQueries()
update queries for updating seller revenue
MyTableModel3()
getValueAt()
getShoppingCart()
getItemsToBuy()
items Bought List is given
getShoppingCart()
getQuantity()
Items Bought Quantities list is returned
getShoppingCart()
calculateUnitPrice()
unit price for all items is returned in a list
table model returned for creating table
getShoppingCart()
calculateTotal()
setText()
JTable()
Buyer order is displayed and checkout is done

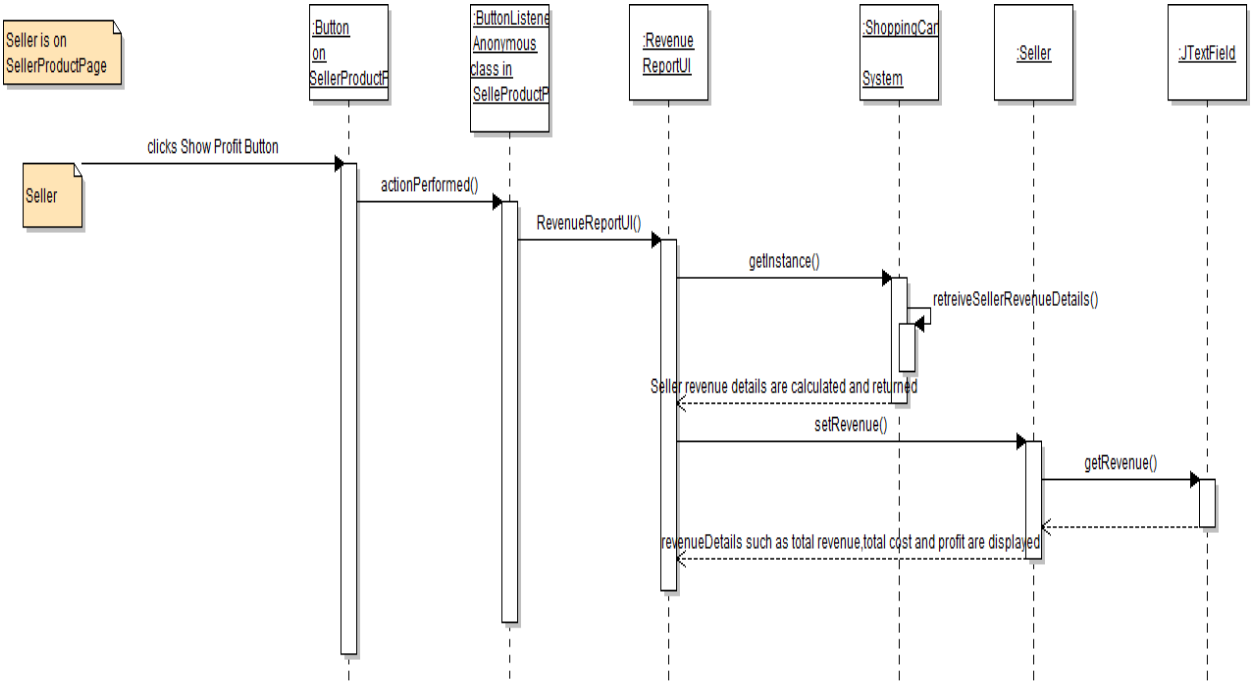
Add item to inventory



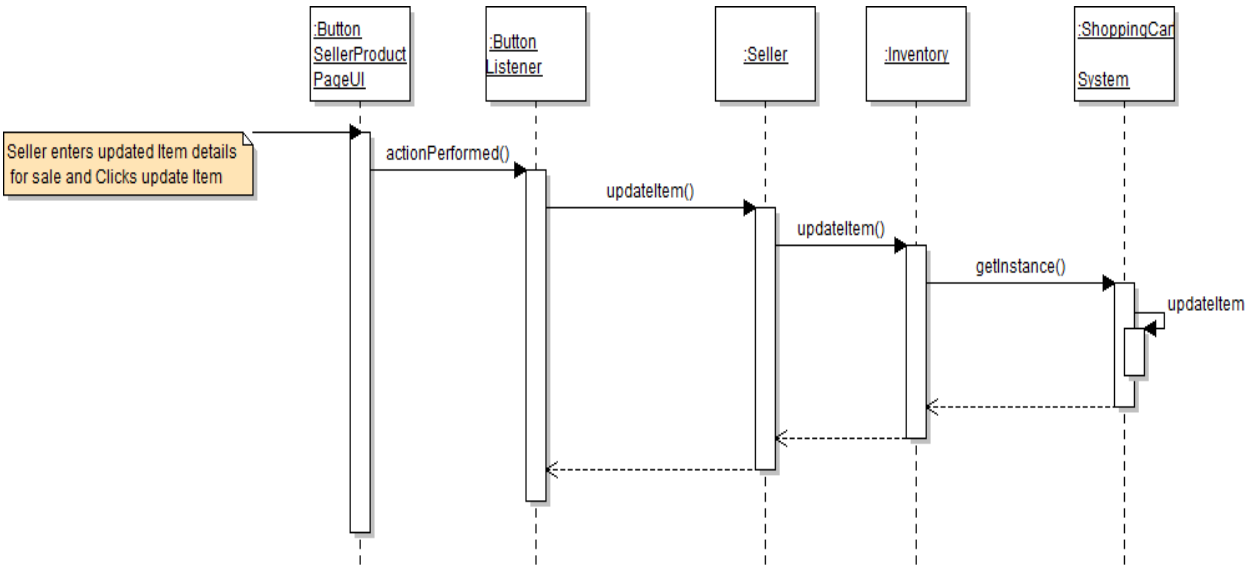
Add Bundle To Inventory



Display Revenue

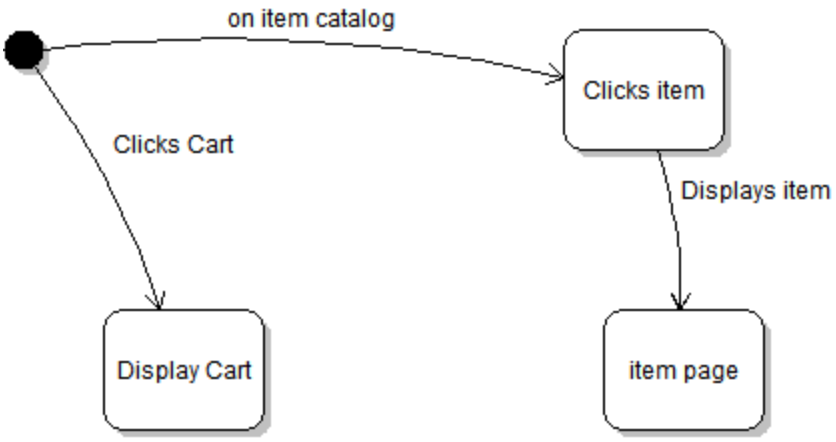


Update inventory by seller

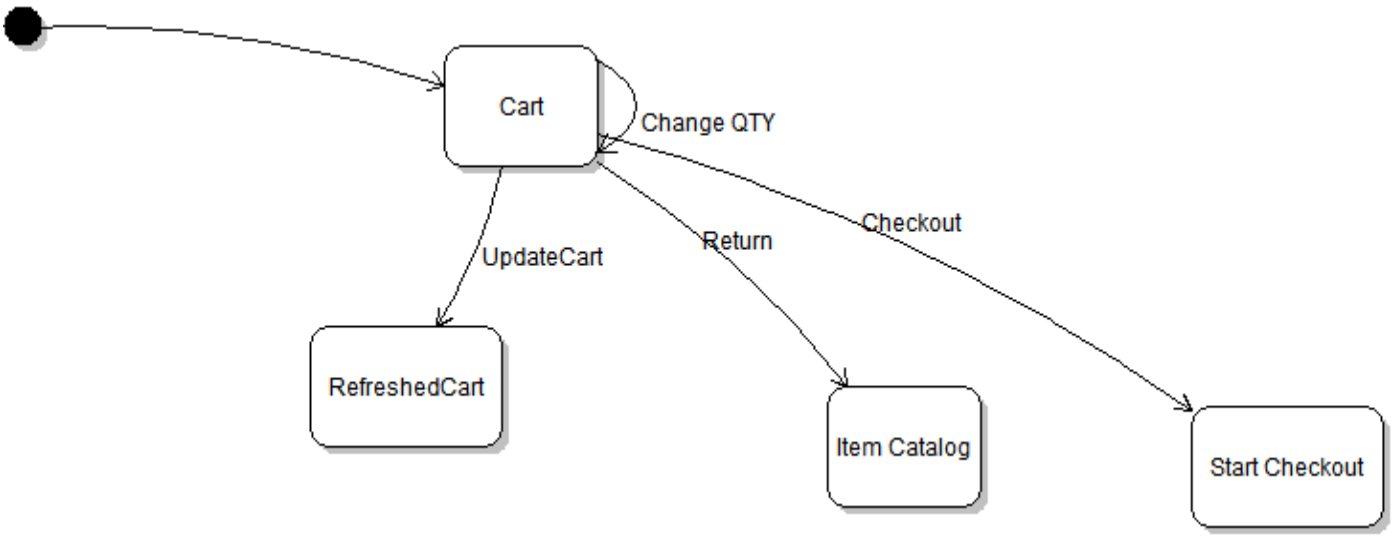


State Diagrams:

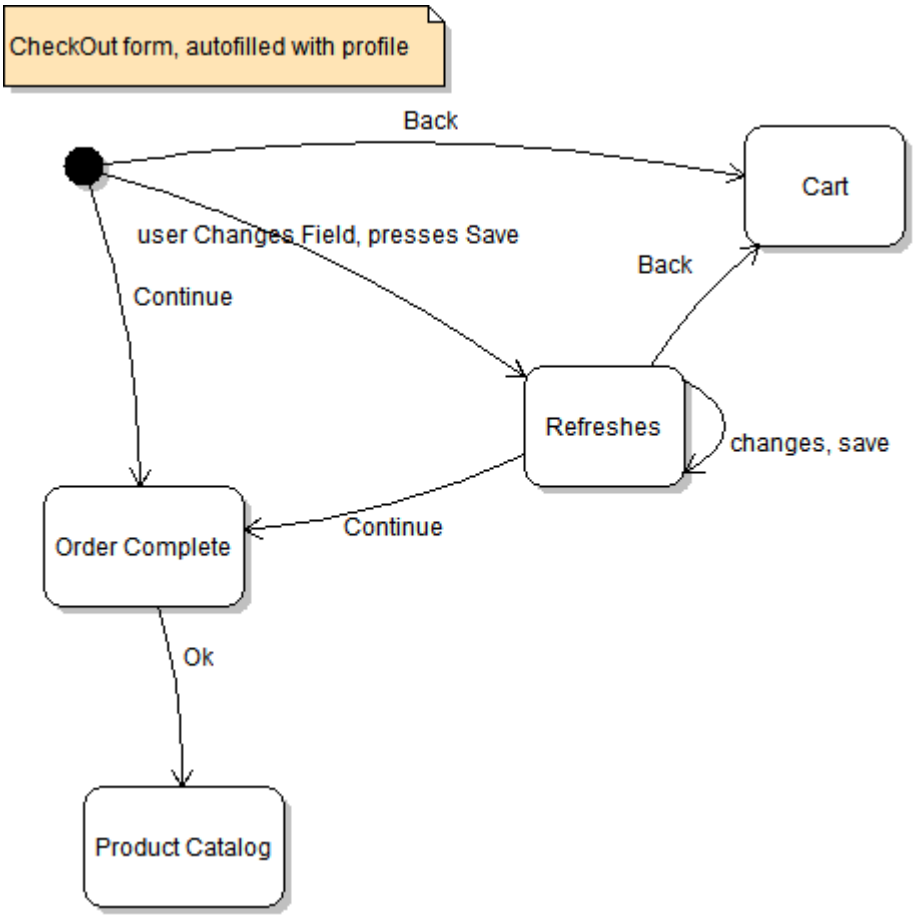
Buyer Catalog State



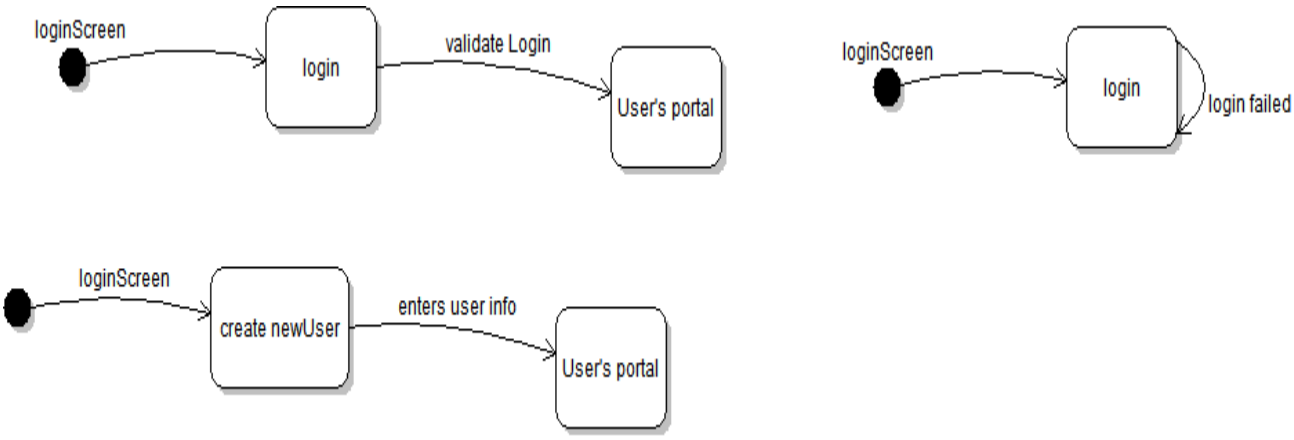
Cart state diagram



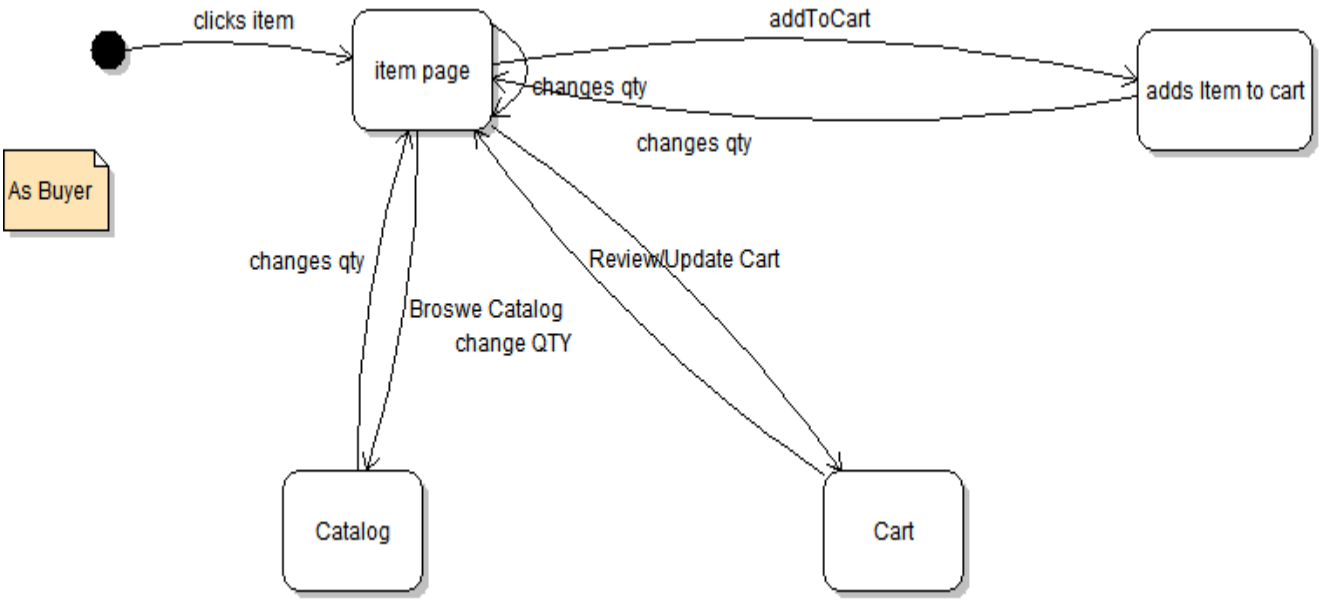
Checkout state



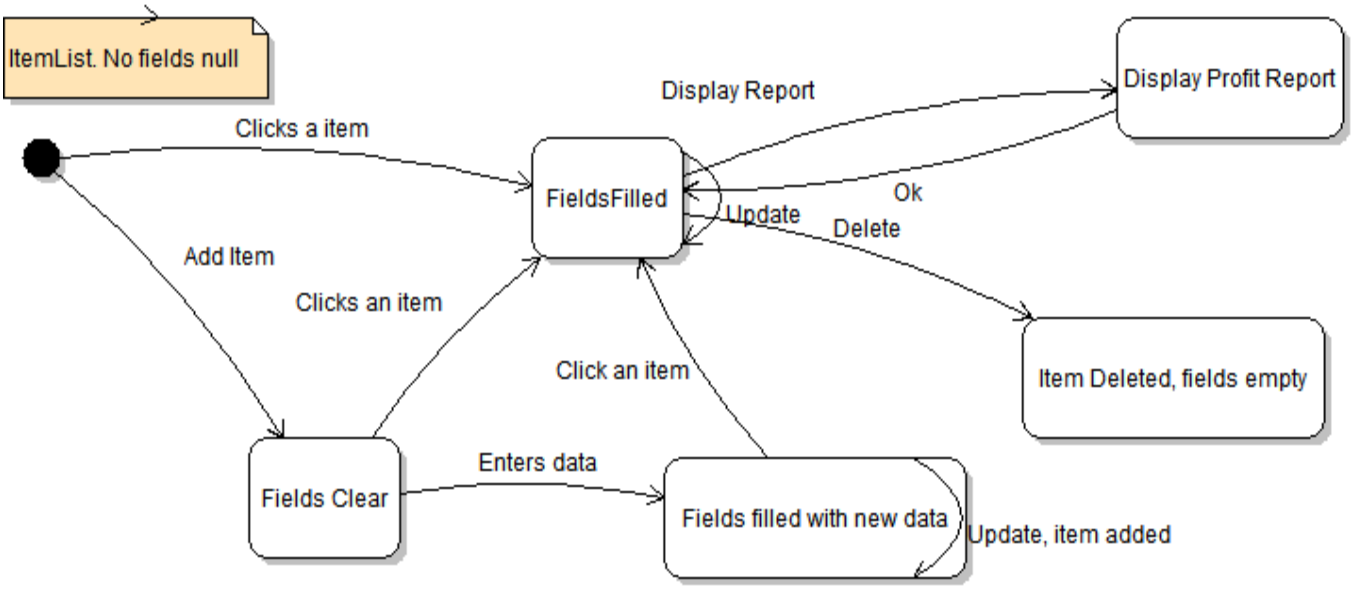
Login State



Product Page State



Seller Catalog State



Source code For the project: Shopping Cart System

//cop4331\prj\src\cop4331\gui\BundleUI.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import cop4331.model.*;
/**
 * A class that gives interface to add bundle
 * @author Miles Robson
 */
public class BundleUI extends JFrame
{

    private JFrame revFrame;

    private JLabel lblTotalRevenue;
    private JLabel lblTotalCost;
    private JLabel lblTotalProfit;

    private JLabel txtTotalRevenue;
    private JLabel txtTotalCost;
    private JLabel txtTotalProfit;

    private JButton btnBack;
    private JPanel revPanel;
    private JPanel mainPanel;
    private JPanel btnPanel;
    Random rn = new Random();

    /**
     * constructor for BundleUI
     * @param seller the Seller object seller who is logged in
     */

    public BundleUI(Seller seller)
    {

        Bundle ourBundle = new Bundle();
        ourBundle.setBundleID(rn.nextInt(Integer.MAX_VALUE) + 1);

        nameLabel = new javax.swing.JLabel();
```

```
        qtyLabel = new javax.swing.JLabel();
        invoiceLabel = new javax.swing.JLabel();
        sellLabel = new javax.swing.JLabel();
        categoryLabel = new javax.swing.JLabel();
        discountLabel = new javax.swing.JLabel();
        nameTextField = new javax.swing.JTextField("", 15);
        qtyTextField = new javax.swing.JTextField("", 15);
        invoiceTextField = new javax.swing.JTextField("", 15);
        sellPriceTextField = new javax.swing.JTextField("", 15);
        cateTextField = new javax.swing.JTextField("", 15);
        disTextField = new javax.swing.JTextField("", 15);
        cancelButton = new javax.swing.JButton();
        saveAdd = new javax.swing.JButton();
        saveCloseButton = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        nameLabel.setText("Name");

        qtyLabel.setText("QTY");

        invoiceLabel.setText("Invoice Rate");

        sellLabel.setText("Sell Price");

        categoryLabel.setText("Category");

        discountLabel.setText("Discount");

        nameTextField.setText("");

        qtyTextField.setText("");

        invoiceTextField.setText("");

        sellPriceTextField.setText("");

        cateTextField.setText("");

        disTextField.setText("");

        cancelButton.setText("Exit w/o save");
        cancelButton.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                //  cancelButtonActionPerformed(evt);
                dispose();
            }
        });

        saveAdd.setText("Save and add another");
        saveAdd.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                Product ourProduct;
                ourProduct = new Product(rn.nextInt(Integer.MAX_VALUE) + 1, nameTextField.getText(),
Double.parseDouble(sellPriceTextField.getText()),
                        Integer.parseInt(qtyTextField.getText()), Double.parseDouble(invoiceTextField.getText()),
                        cateTextField.getText(), seller.getUserID());
                ourBundle.add(ourProduct);
```

```
            nameTextField.setText("");

            qtyTextField.setText("");

            invoiceTextField.setText("");

            sellPriceTextField.setText("");

            cateTextField.setText("");

            disTextField.setText("");

        }
    });

    saveCloseButton.setText("Save and close");
    saveCloseButton.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            Product ourProduct;
            ourProduct = new Product(rn.nextInt(Integer.MAX_VALUE) + 1, nameTextField.getText(),
Double.parseDouble(sellPriceTextField.getText()),
                Integer.parseInt(qtyTextField.getText()), Double.parseDouble(invoiceTextField.getText()),
                cateTextField.getText(), seller.getUserID());
            ourBundle.add(ourProduct);
            nameTextField.setText("");

            qtyTextField.setText("");

            invoiceTextField.setText("");

            sellPriceTextField.setText("");

            cateTextField.setText("");

            disTextField.setText("");
            seller.getInventory().addBundle(ourBundle);
            dispose();
        }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(discountLabel)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(disTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(categoryLabel)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(cateTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(sellLabel)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(sellPriceTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(invoiceLabel)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(invoiceTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(qtyLabel)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                            .addComponent(qtyTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(nameLabel)
                            .addGap(85, 85, 85)
                            .addComponent(nameTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(cancelButton)
                        .addGap(18, 18, 18)
                        .addComponent(saveAdd)
                        .addGap(18, 18, 18)
                        .addComponent(saveCloseButton)))
                .addContainerGap(30, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(nameLabel)
                    .addComponent(nameTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(qtyLabel)
                    .addComponent(qtyTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(invoiceLabel)
                    .addComponent(invoiceTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(sellLabel)
                    .addComponent(sellPriceTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(categoryLabel)
                    .addComponent(cateTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(discountLabel)
```

```
                        .addComponent(disTextField, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(30, 30, 30)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(cancelButton)
                        .addComponent(saveAdd)
                        .addComponent(saveCloseButton))
                    .addContainerGap(89, Short.MAX_VALUE))
        );

        pack();
    }

    private javax.swing.JButton cancelButton;
    private javax.swing.JTextField cateTextField;
    private javax.swing.JLabel categoryLabel;
    private javax.swing.JTextField disTextField;
    private javax.swing.JLabel discountLabel;
    private javax.swing.JLabel invoiceLabel;
    private javax.swing.JTextField invoiceTextField;
    private javax.swing.JLabel nameLabel;
    private javax.swing.JTextField nameTextField;
    private javax.swing.JLabel qtyLabel;
    private javax.swing.JTextField qtyTextField;
    private javax.swing.JButton saveAdd;
    private javax.swing.JButton saveCloseButton;
    private javax.swing.JLabel sellLabel;
    private javax.swing.JTextField sellPriceTextField;
}
```

//cop4331\prj\src\cop4331\gui\CheckOutUI.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.gui;

import java.awt.*;
import java.awt.event.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;
import javax.sql.rowset.JdbcRowSet;
import javax.swing.*;
import cop4331.model.*;
import javax.swing.BorderFactory;
//import static cop4331.model.ShoppingCartSystem.query;

/**
* This UI class gives interface for user(buyer) to enter his credit card details
* and then checkout.
* @author Manisha
*/
public class CheckOutUI extends JFrame
{
    /**
     *
     * @author Manisha
     */

    private JFrame UPFrame;
    private JPanel UPPanel;
    private JPanel mainPanel;
    private JPanel creditCardPanel;
    private JPanel btnPanel;
    private JLabel lblUName;
    private JLabel lblPWD;
    private JLabel lblFName;
    //private JLabel lblLName;
    private JLabel lblUAddress;
    private JLabel lblUCity;
    private JLabel lblUState;
    private JLabel lblUZip;
    private JLabel lblUPhone;
    private JLabel lblUEmail;
    private JLabel lblUserID;

    private JTextField TxtUName;
    private JTextField TxtPWD;
    private JTextField TxtFName;
    //private JTextField TxtLName;
    private JTextField TxtUAddress;
    private JTextField TxtUCity;
    private JTextField TxtUState;
    private JTextField TxtUZip;
    private JTextField TxtUPhone;
    private JTextField TxtUEmail;
```

```
private JTextField TxtUserID;
private JLabel lblCCNo;
private JLabel lblCHolder;
private JLabel lblCType;
private JLabel lblValid;
private JLabel lblCVV;

private JTextField TxtCCNo;
private JTextField TxtCHolder;
private JTextField TxtCType;
private JTextField TxtValid;
private JTextField TxtCVV;

private JButton btnEdit;
private JButton btnSave;
private JButton btnContinue;
/**
 * Constructor for CheckOutUI.
 * @param buyer The object of Buyer who logged into the system.
 */
public  CheckOutUI(Buyer buyer)//(UsersofSystem newUser, String userType)
{

    lblFName= new JLabel("Name: ");
    lblUAddress= new JLabel("Address: ");
    lblUCity= new JLabel("City: ");
    lblUState= new JLabel("State: ");
    lblUZip= new JLabel("Zip Code: ");
    lblUPhone= new JLabel("Phone: ");
    lblUEmail= new JLabel("Email: ");

    TxtFName= new JTextField("", 15);
    TxtFName.setText(buyer.getUserName());
    TxtUAddress= new JTextField("", 15);
    TxtUAddress.setText(buyer.getAddress());
    TxtUCity= new JTextField("", 15);
    TxtUCity.setText(buyer.getCity());
    TxtUState= new JTextField("", 15);
    TxtUState.setText(buyer.getState());
    TxtUZip= new JTextField("", 15);
    TxtUZip.setText(buyer.getZip());
    TxtUPhone= new JTextField("", 15);
    TxtUPhone.setText(buyer.getPhone());
    TxtUEmail= new JTextField("", 15);
    TxtUEmail.setText(buyer.getEmail());

    lblCCNo= new JLabel("Credit Card Number: ");
    lblCHolder= new JLabel("Card Holder's Name: ");
    lblCType= new JLabel("Credit Card Type: ");
    lblValid= new JLabel("Valid Till: ");
    lblCVV= new JLabel("CVV Code: ");

    TxtCCNo= new JTextField("", 15);
    TxtCHolder= new JTextField("", 15);
    TxtCType= new JTextField("", 15);
    TxtValid= new JTextField("", 15);
    TxtCVV= new JTextField("", 15);

    btnEdit = new JButton("Edit");
    btnSave= new JButton("Save");
    btnContinue= new JButton("Checkout");
```

```
mainPanel= new JPanel(new BorderLayout());
mainPanel.setBorder(BorderFactory.createTitledBorder("Checkout"));
//    UPPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
//    creditCardPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
//    btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));
btnPanel= new JPanel(new FlowLayout());
UPPanel= new JPanel(new GridLayout(8, 2));
creditCardPanel =  new JPanel(new GridLayout(5, 2));

UPPanel.add(lblFName);
UPPanel.add(TxtFName);
UPPanel.add(lblUAddress);
UPPanel.add(TxtUAddress);
UPPanel.add(lblUCity);
UPPanel.add(TxtUCity);
UPPanel.add(lblUState);
UPPanel.add(TxtUState);
UPPanel.add(lblUZip);
UPPanel.add(TxtUZip);
UPPanel.add(lblUPhone);
UPPanel.add(TxtUPhone);
UPPanel.add(lblUEmail);
UPPanel.add(TxtUEmail);
UPPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));
creditCardPanel.add(lblCCNo);
creditCardPanel.add(TxtCCNo);
creditCardPanel.add(lblCHolder);
creditCardPanel.add(TxtCHolder);
creditCardPanel.add(lblCType);
creditCardPanel.add(TxtCType);
creditCardPanel.add(lblValid);
creditCardPanel.add(TxtValid);
creditCardPanel.add(lblCVV);
creditCardPanel.add(TxtCVV);

creditCardPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
//TxtUserID.setText(Integer.toString(newUser.getUserID()));

//btnPanel.setPreferredSize(new Dimension(40,40));
btnPanel.add(btnEdit);
btnEdit.addActionListener(new
ActionListener()
{
public void actionPerformed(ActionEvent event)
{
JOptionPane.showMessageDialog(null,"New RECORD created successfully.");
}
});
btnPanel.add(btnSave);
btnSave.addActionListener(new
ActionListener()
{
public void actionPerformed(ActionEvent event)
{
getTxtFieldData(buyer);
buyer.saveBuyerCreditcardInfo();
JOptionPane.showMessageDialog(null,"Record Updated!");

}
});
/*
btnSaveProfile.addActionListener(new
```

```
    ActionListener()
    {
    public void actionPerformed(ActionEvent event)
    {
    textField.setText("Hello, World!");
    }
    });
    */
    //btnPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));
    mainPanel.add(UPPanel, BorderLayout.NORTH);
    mainPanel.add(creditCardPanel, BorderLayout.CENTER);
    mainPanel.add(btnPanel, BorderLayout.SOUTH);


    JFrame frame = new JFrame("Checkout");
    frame.setSize(700, 400);
    frame.add(mainPanel);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.setVisible(true);
    btnPanel.add(btnContinue);
    btnContinue.addActionListener(new
    ActionListener()
    {
    public void actionPerformed(ActionEvent event)
    {
    OrderConfirmationUI orderConfirmationPage =  new OrderConfirmationUI((Buyer)buyer);
    frame.dispose();
    }
    });
}
/**
 * This method gets all text data into the Buyer object p.
 * @param p Buyer object.
 */
private void getTxtFieldData(Buyer p)
{
    p.setName(TxtFName.getText());
    p.setAddress(TxtUAddress.getText());
    p.setCity(TxtUCity.getText());
    p.setState(TxtUState.getText());
    p.setZip(TxtUZip.getText());
    p.setPhone(TxtUPhone.getText());
    p.setEmail(TxtUEmail.getText());
    p.setcreditCardNumber(TxtCCNo.getText());
    p.setcreditCardHolder(TxtCHolder.getText());
    p.setcardType(TxtCType.getText());
    p.setvalidTill(TxtValid.getText());
    p.setcvvCode(TxtCVV.getText());
}
/**
 * The method to set text with user profile information available.
 * @param p UserdofSystem object
 */
private void setTxtFieldData(UsersofSystem p)
{
    TxtUserID.setText(String.valueOf(p.getUserID()));
    TxtUName.setText(p.getUserName());
    TxtPWD.setText(p.getPassword());
    TxtFName.setText(p.getName());
    TxtUAddress.setText(p.getAddress());
    TxtUCity.setText(p.getCity());
```

```
        TxtUState.setText(p.getState());
        TxtUZip.setText(p.getZip());
        TxtUPhone.setText(p.getPhone());
        TxtUEmail.setText(p.getEmail());
    }

    // private boolean isEmptyFieldData() {
    //     return (TxtUserID.getText().trim().isEmpty()
    //         && TxtUName.getText().trim().isEmpty()
    //         && TxtPWD.getText().trim().isEmpty()
    //         && TxtFName.getText().trim().isEmpty()
    //         && TxtUAddress.getText().trim().isEmpty()
    //         && TxtUCity.getText().trim().isEmpty()
    //         && TxtUState.getText().trim().isEmpty()
    //         && TxtUZip.getText().trim().isEmpty()
    //         && TxtUPhone.getText().trim().isEmpty()
    //         && TxtUEmail.getText().trim().isEmpty());

    // }

}
```

//cop4331\prj\src\cop4331\gui\LogInUI.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/

package cop4331.gui;
import cop4331.model.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Random;
import javax.swing.*;
 //import static cop4331.model.ShoppingCartSystem.;
/**
* A UserInterface Class for Login Functionality
* @author Manisha
*/
public class LogInUI
{
   private JTextField username;
   private JPasswordField password;
   private JLabel usernamelabel;
   private JLabel passwordlabel;
   private JPanel loginUNPanel;
   private JPanel btnPanel;
   private JPanel mainPanel;
   private JButton btnLogin;
   private JButton btnNewSeller;
   private JButton btnNewBuyer;
   private JButton btnCancel;
   /**
    * Constructor that creates login User Interface
    * using JtextFields, JLabel and JButtons.
    */
   public LogInUI()
   {
      username = new JTextField("", 15);
      username.setSize(5, 2);
      password = new JPasswordField("",15);
      usernamelabel= new JLabel("Username:");
      usernamelabel.setPreferredSize(new Dimension(150,40));
      passwordlabel= new JLabel("Password:");
      passwordlabel.setPreferredSize(new Dimension(150,40));
      loginUNPanel = new JPanel(new GridLayout(2, 2));
      btnPanel = new JPanel(new GridLayout(1, 4, 5, 5));
      mainPanel= new JPanel(new BorderLayout());
      username.setBounds(15, 15, 155, 55);
      username.setPreferredSize(new Dimension(155,25));
      password.setPreferredSize(new Dimension(155,25));

      loginUNPanel.add(usernamelabel);
      loginUNPanel.add(username);
      loginUNPanel.add(passwordlabel);
      loginUNPanel.add(password);


      btnLogin = new JButton("Login");

      //anonymous class
      btnLogin.addActionListener(new
      ActionListener()
```

```
{
   public void actionPerformed(ActionEvent event)
   {

      char[] pwd = password.getPassword();
      String passwrd = new String(pwd);
      String uname = username.getText();

     // query = "SELECT * from USERS";
      String usertype = ShoppingCartSystem.getInstance().loginVerify(uname, passwrd);
      if(usertype.equals("N"))
      {
         username.setText("");
         password.setText("");
      }
      else
      if(usertype.equals("S"))
      {
         Seller sellerUser = new Seller(uname, passwrd);
         SellerProductPage sellersproductpage = new  SellerProductPage(sellerUser);
      }
      else
      if(usertype.equals("B"))
      {
         Buyer buyerUser = new Buyer(uname, passwrd);
         ProductCatalogUI browseCatalog = new  ProductCatalogUI(buyerUser);
      }

   }
});

btnNewSeller = new JButton("New Seller");
btnNewSeller.addActionListener(new
ActionListener()
{
   public void actionPerformed(ActionEvent event)
   {
      UsersofSystem sellerUser = new Seller();
      String s ="S";
      UserProfileUI userprofile = new  UserProfileUI(sellerUser, s);
   }
});
btnNewBuyer = new JButton("New Buyer");
btnNewBuyer.addActionListener(new
ActionListener()
{
   public void actionPerformed(ActionEvent event)
   {
      UsersofSystem buyerUser = new Buyer();
      String s ="B";
      UserProfileUI userprofile = new  UserProfileUI(buyerUser, s);
   }
});

btnCancel = new JButton("Cancel");
btnPanel.add(btnLogin);
btnPanel.add(btnNewSeller);
btnPanel.add(btnNewBuyer);
btnPanel.add(btnCancel);
mainPanel.add(loginUNPanel, BorderLayout.NORTH);
mainPanel.add(btnPanel,BorderLayout.SOUTH );
```

```
        mainPanel.setBorder(BorderFactory.createTitledBorder("Login"));
        loginUNPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));

        JFrame frame = new JFrame("Shopping Cart Application");
        frame.setSize(450, 200);
        frame.add(mainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);

        btnCancel.addActionListener(new
        ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                frame.dispose();
            }
        });
    }
    /**
     * This method gets data from textfield into the UsersofSystem object.
     * @param p This is a UserOfSystem object.
     */
    private void getTxtFieldData(UsersofSystem p)
    {
        p.setUserName(username.getText());
        p.setPassword(password.getPassword().toString());
    }
   /**
   * This is the main method which makes use of LogInUI() constructor.
   * @param args Unused.
   *
   */
    public static void main(String[] args)
    {
        LogInUI myUI = new LogInUI();
    }

}
```

//cop4331\prj\src\cop4331\gui\OrderConfirmationUI.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.gui;
import cop4331.model.*;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.LinkedList;
import java.util.Random;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.AbstractTableModel;
/**
* The class That displays order
* @author Manisha
*/
public class OrderConfirmationUI extends JFrame
{
    private JLabel lblOrderNo;
    private JLabel lblOrderDate;
    private JLabel lblTotal;
    private JTextField txtOrderNo;
    private JTextField txtOrderDate;
    private JButton btnOK;
    private JTable table ;
    private JPanel btnPanel;
    private JPanel mainPanel;
    private JPanel tblPanel;
    private JPanel txtPanel;
    /**
     * Constructor for OrderConfirmationUI.
     * @param buyer The Buyer object, the buyer who logged into the system.
```

```
 */
public OrderConfirmationUI(Buyer buyer)
{
   updateInventory(buyer.getShoppingCart());
   updateSellerRevenue(buyer.getShoppingCart());
   tblPanel = new JPanel(new GridLayout(1, 0));
   txtPanel = new JPanel(new GridLayout(1, 4));
   btnPanel = new JPanel(new BorderLayout());
   lblTotal = new JLabel("Total: " + Double.toString(buyer.getShoppingCart().calculateTotal()) + "              ");
   btnPanel.add(lblTotal,BorderLayout.EAST );
   MyTableModel3 tableModel =  new MyTableModel3(buyer);
   table = new JTable(tableModel);
   table.setPreferredScrollableViewportSize(new Dimension(500, 100));

   //Create the scroll pane and add the table to it.
   JScrollPane scrollPane = new JScrollPane(table);

   //Add the scroll pane to this panel.
   tblPanel.add(scrollPane);

   lblOrderNo = new JLabel("Order No:");
   lblOrderDate = new JLabel("Order Date: ");
   txtOrderNo = new JTextField("", 15);
   txtOrderDate = new JTextField("", 15);
   btnOK = new JButton("OK");
   btnOK.setSize(20, 20);

   txtPanel.add(lblOrderNo);
   txtPanel.add(txtOrderNo);
   txtPanel.add(lblOrderDate);
   txtPanel.add(txtOrderDate);
   Random rn = new Random();
   int orderNo = rn.nextInt(Integer.MAX_VALUE) + 1;
   txtOrderNo.setText(Integer.toString(orderNo));
   //txtOrderDate = new JTextField(" ", 15);
   Date date = Calendar.getInstance().getTime();

   SimpleDateFormat format =  new SimpleDateFormat("MM-DD-YYYY");
   String DateToStr;
   DateToStr = format.format(date);
   DateToStr = format.format(date);
   txtOrderDate.setText(DateToStr);
   btnPanel.add( btnOK, BorderLayout.SOUTH );

   mainPanel= new JPanel(new BorderLayout());
   mainPanel.add(txtPanel, BorderLayout.NORTH);
   mainPanel.add(tblPanel, BorderLayout.CENTER);
   mainPanel.add(btnPanel,BorderLayout.SOUTH );
   mainPanel.setBorder(BorderFactory.createTitledBorder("Order Confirmation"));
   txtPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
   btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));

   ListSelectionModel selectionModel = table.getSelectionModel();
   selectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   selectionModel.addListSelectionListener(new OrderConfirmationUI.RowListener(this));

   JFrame frame = new JFrame("Order Confirmation Page");
   frame.setSize(500, 400);
   frame.add(mainPanel);
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   frame.setVisible(true);
```

```
        table.enableInputMethods(false);
        //anonymous class
        btnOK.addActionListener(new
        ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                frame.dispose();
                java.awt.Window win[] = java.awt.Window.getWindows();
                for(int i=0;i<win.length;i++)
                {
                    win[i].dispose();
                }
                new LogInUI();
            }
        });
    }
    /**
     * The method to update inventory of shopped items
     * @param shoppingCart The ShoppingCart object.
     */
    private void updateInventory(ShoppingCart shoppingCart)
    {
        ShoppingCartSystem.getInstance().updateInventory(shoppingCart);
    }
    /**
     * The method to update seller revenue.
     * @param shoppingCart The ShoppingCart object
     */
    private void updateSellerRevenue(ShoppingCart shoppingCart)
    {
        ShoppingCartSystem.getInstance().updateSellerRevenue(shoppingCart);
    }

//    private static class object {
//
//        public object() {
//        }
//    }

//    public static void main(String[] args) {
//    //OrderConfirmationUI orderConfirmationPage =  new OrderConfirmationUI((Buyer)buyer);
//    }

    private static class RowListener implements ListSelectionListener {

        public RowListener(OrderConfirmationUI aThis) {
        }

        @Override
        public void valueChanged(ListSelectionEvent e) {
            throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose
Tools | Templates.
        }
    }

}
/**
 * Inner class that extends AbstractTableModel which provides default implementations for most of the methods in the
TableModel interface.
 * @author Manisha
 */
```

```
class MyTableModel3 extends AbstractTableModel
{
    ResultSet rs;
    String query;
    private String[] columnNames = { "Product Name","Price", "Quantity","Amount" };
    LinkedList<Object[]> records = new LinkedList<>();
    Buyer currentBuyer;
    /**
     * Constructor for MyTableModel3
     * @param cb The Buyer current object that means a buyer who is currently logged in.
     */
    public MyTableModel3(Buyer cb)
    {
        currentBuyer = cb;
    }
    /**
     * Returns the number of columns in the model. A JTable uses this method to determine how many columns it should create
and display by default.
     * @return int the number of columns in the model.
     */
    public int getColumnCount() {
        return columnNames.length;
    }
    /**
     * Returns the number of rows in the model. A JTable uses this method to determine how many rows it should display.
     * @return int The number of rows in the model
     */
    public int getRowCount() {
        return currentBuyer.getShoppingCart().getItemsToBuy().size();
    }
    /**
     * Returns the name of the column at columnIndex.
     * @param col The columnIndex.
     * @return string the name of the column
     */
    public String getColumnName(int col) {
        return columnNames[col];
    }
    /**
     * Returns the value for the cell at columnIndex and rowIndex.
     * @param row the row whose value is to be queried.
     * @param col the column whose value is to be queried.
     * @return the value Object at the specified cell
     */
    public Object getValueAt(int row, int col)
    {

        ArrayList<LineItem> ItemsBought = currentBuyer.getShoppingCart().getItemsToBuy();
        ArrayList<Integer> ItemsBoughtQty = currentBuyer.getShoppingCart().getQuantity();
        ArrayList<Double>  amtForProductBought = currentBuyer.getShoppingCart().calculateUnitPrice();
        Object val = null ;

        for(int i =0; i< ItemsBought.size(); i++)
        {
        if( i == row )
           {
              switch(col)
              {
                 case 0:
                 val = ItemsBought.get(i).getDescription().trim();
                 break;
```

```
                case 1:
                val = ItemsBought.get(i).getPrice();
                break;
                case 2:
                val = ItemsBoughtQty.get(i);
                break;
                case 3:
                val = amtForProductBought.get(i);
                break;
            }
        }
    }
    return val;
}

/*
* JTable uses this method to determine the default renderer/ editor for
* each cell.  This is used by the JTable to set up a default renderer and editor for the column.
*/
/**
 * Returns the most specific superclass for all the cell values in the column.
 * @param c the index of the column
 * @return the class of the object values in the model.
 */
@Override
public Class getColumnClass(int c)
{
    return getValueAt(0, c).getClass();
}

/*
* Don't need to implement this method unless our table's editable.
*/
@Override
public boolean isCellEditable(int row, int col)
{
    // the data/cell address is constant,
    //no matter where the cell appears onscreen.
    if (col < getColumnCount()) {
        return false;
    } else {
        return true;
    }
}

/*
* Don't need to implement this method unless our table's data can
* change.
*/
@Override
public void setValueAt(Object value, int row, int col) {
//      if (DEBUG) {
//        ShoppingCartSystem.out.println("Setting value at " + row + "," + col
//            + " to " + value + " (an instance of "
//            + value.getClass() + ")");
//      }
//
//      data[row][col] = value;
//      fireTableCellUpdated(row, col);
//
//      if (DEBUG) {
//        ShoppingCartSystem.out.println("New value of data:");
```

```
//        printDebugData();
//      }
//
//        try {
//      if (!rs.absolute(row + 1)) {
//        return;
//      }
//      rs.updateObject(col + 1, value);
//    } catch (SQLException e) {
//    }
    }

    private void printDebugData()
    {
//        int numRows = getRowCount();
//        int numCols = getColumnCount();
//
//        for (int i = 0; i < numRows; i++) {
//            System.out.print("    row " + i + ":");
//            for (int j = 0; j < numCols; j++)
//            {
//            //System.out.print("  " + data[i][j]);
//                System.out.print("  " + records.get(i));
//            }
//            System.out.println();
//        }
//        System.out.println("--------------------------");
    }
}
```

//cop4331\prj\src\cop4331\gui\ProductCatalogUI.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.gui;
import cop4331.model.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.util.ArrayList;
import java.util.LinkedList;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.AbstractTableModel;

/**
* The class that provides user interface for viewing product catalog
* and adding Line Items to shopping cart.
* @author Manisha
*/
public class ProductCatalogUI extends JFrame {

private boolean DEBUG = false;
private JPanel tblPanel;
private JPanel txtPanel;
private JLabel PID;
private JLabel PDesc;
private JLabel PQty;
//private JLabel PIRate;
private JLabel PSRate;
private JLabel POrderQty;
private JLabel PBID;

private JTextField txtPID;
private JTextField txtPDesc;
private JTextField txtPQty;
//private JTextField txtPIRate;
private JTextField txtPSRate;
```

```
private JTextField txtPOrderQty;
private JTextField txtPBID;

private JButton btnAdd;
private JButton btnUpdate;
//private JButton btnDelete;
// private JButton btnSave;
private JButton btnCancel;

private JPanel btnPanel;
private JPanel mainPanel;
private JTable table ;
private JLabel label;
private int currentListItemIndex;
/**
 * Constructor for creating Product Catalog UI
 * @param buyer The buyer object the current buyer who is logged in.
 */
public ProductCatalogUI(UsersofSystem buyer) {
    tblPanel = new JPanel(new GridLayout(1, 0));
    txtPanel = new JPanel(new GridLayout(6, 2));
    btnPanel = new JPanel(new GridLayout(1, 4,5,5));
    table = new JTable(new MyTableModel2((Buyer) buyer));
    table.setPreferredScrollableViewportSize(new Dimension(500, 100));

    //Create the scroll pane and add the table to it.
    JScrollPane scrollPane = new JScrollPane(table);

    //Add the scroll pane to this panel.
    tblPanel.add(scrollPane);

    // PID = new JLabel("Product ID: ");
    PDesc = new JLabel("Product Name: ");
    PQty = new JLabel("Quantity Available: ");
    PSRate = new JLabel("Sell Price: ");
    POrderQty = new JLabel("Order Quantity: ");
    txtPDesc = new JTextField("", 15);
    txtPQty = new JTextField("", 15);
    txtPSRate = new JTextField("", 15);
    txtPOrderQty = new JTextField("", 15);

    txtPanel.add(PDesc);
    txtPanel.add(txtPDesc);
    txtPanel.add(PQty);
    txtPanel.add(txtPQty);
    txtPanel.add(PSRate);
    txtPanel.add(txtPSRate);
    txtPanel.add(POrderQty);
    txtPanel.add(txtPOrderQty);


    txtPanel.setBorder(BorderFactory.createTitledBorder("Product Details"));

    label = new JLabel("");
    btnAdd = new JButton("Add To Cart");
    btnAdd.setSize(20, 20);
    btnUpdate= new JButton("Review/Update Cart");
    btnUpdate.setSize(20, 30);
    //btnDelete= new JButton("Delete Product");
    // btnSave= new JButton("Save Product");
    // btnSave.setSize(20, 20);
    btnCancel= new JButton("Cancel");
```

```
btnCancel.setSize(20, 20);

btnPanel.add(btnAdd);
btnAdd.addActionListener(new
ActionListener()
{
   public void actionPerformed(ActionEvent event)
   {
      boolean stockeVerified = false;
      table.enableInputMethods(true);
      ArrayList<LineItem> allProducts = ShoppingCartSystem.getInstance().retrieveAllProducts();
      int stock =  Integer.parseInt(txtPQty.getText().trim());
      int quantity = Integer.parseInt(txtPOrderQty.getText().trim());
      if(quantity <= stock)
      {
         ((Buyer)buyer).addToShoppingCart(allProducts.get(currentListItemIndex), quantity);
         JOptionPane.showMessageDialog(null,"Item Added!!");
      }
      else
      {
         JOptionPane.showMessageDialog(null,"Not enough available in stock!!reenter the order quantity.");
         txtPOrderQty.setText("");
         txtPOrderQty.grabFocus();
      }
   }
});
btnPanel.add( btnUpdate);
btnUpdate.addActionListener(new
ActionListener()
{
   public void actionPerformed(ActionEvent event)
   {
      ReviewUpdateCartUI reviewUpdateCartUI = new ReviewUpdateCartUI((Buyer)buyer);
   }
});

//btnPanel.add( btnDelete);
//btnPanel.add(btnSave);

btnPanel.add( btnCancel);


mainPanel= new JPanel(new BorderLayout());

mainPanel.add(tblPanel, BorderLayout.NORTH);
mainPanel.add(txtPanel, BorderLayout.CENTER);
mainPanel.add(btnPanel,BorderLayout.SOUTH );

mainPanel.setBorder(BorderFactory.createTitledBorder("Browse Catalog Page"));
txtPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));

ListSelectionModel selectionModel = table.getSelectionModel();
selectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
selectionModel.addListSelectionListener(new RowListener(this));

JFrame frame = new JFrame("Browse Catalog ");
frame.setSize(500, 400);
frame.add(mainPanel);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//frame.pack();
frame.setVisible(true);
```

```
    table.enableInputMethods(false);
    btnCancel.addActionListener(new
    ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            frame.dispose();
        }
    });
}


//private static class object {
//
//    public object() {
//    }
//}

    /**
     * Sets the item index in the list.
     * @param index The index.
     */
    public void setCurrentListItemIndex(int index)
    {
        currentListItemIndex = index;
    }

    class RowListener implements ListSelectionListener
    {
        ProductCatalogUI readRow;
        JTable table;

        public RowListener(ProductCatalogUI rar)
        {
            readRow = rar;
            table = readRow.table;
        }

        public void valueChanged(ListSelectionEvent e)
        {
            if(!e.getValueIsAdjusting())
            {
                ListSelectionModel model = table.getSelectionModel();
                int lead = model.getLeadSelectionIndex();
                displayRowValues(lead);
            }
        }

        private void displayRowValues(int rowIndex)
        {
            int columns = table.getColumnCount();
            readRow.setCurrentListItemIndex(rowIndex);

            ArrayList<String> data= new ArrayList<String>();
            for(int col = 0; col < columns; col++)
            {
            Object o = table.getValueAt(rowIndex, col);
            data.add(col, o.toString());

            }

            //        readRow.txtPID.setText(data.get(0));
```

```
            readRow.txtPDesc.setText(data.get(0));
            readRow.txtPQty.setText(data.get(1));
            // readRow.txtPIRate.setText(data.get(2));
            readRow.txtPSRate.setText(data.get(2));


        }

        private void getTblRowInTxt(String str)
        {

        }

    }

}

/**
 * * Inner class that extends AbstractTableModel which provides default implementations for most of the methods in the
TableModel interface.
 * @author Manisha
 */
class MyTableModel2 extends AbstractTableModel
{
    ResultSet rs;
    String query;
    private String[] columnNames = {  "Product Name","Quantity","Sell Price"};
    LinkedList<Object[]> records = new LinkedList<>();
    ArrayList<LineItem> productsavailable;
    Buyer currentBuyer;
    /**
     * Constructor for MyTableModel3
     * @param cs The Buyer current buyer who is currently logged in
     */
    public MyTableModel2(Buyer cs)
    {
        currentBuyer = cs;
        productsavailable = ShoppingCartSystem.getInstance().retrieveAllProducts();
    }
    /**
     * Returns the number of columns in the model.
     * A JTable uses this method to determine how many columns it should create and display by default.
     * @return return  the number of columns in the model.
     */

    public int getColumnCount() {
        return columnNames.length;
    }
    /**
     * Returns the number of rows in the model. A JTable uses this method to determine how many rows it should display.
     * @return int The number of rows in the model
     */
    public int getRowCount() {

    //return currentBuyer.getProductCount();
        return productsavailable.size();
    }
    /**
     * Returns the name of the column at columnIndex.
     * @param col The columnIndex.
     * @return string the name of the column
     */
```

```
public String getColumnName(int col) {
    return columnNames[col];
}
/**
* Returns the value for the cell at columnIndex and rowIndex.
* @param row the row whose value is to be queried.
* @param col the column whose value is to be queried.
* @return the value Object at the specified cell

*/
public Object getValueAt(int row, int col) {
Object val = null;
//**********************Iterator Pattern Check***************//
int i = 0;
for(LineItem it:productsavailable)
{
    if( i == row )
    {
        System.out.println("works iterator");
        switch (col)
        {
            case 0:
                val = it.getDescription();
                break;
            case 1:
                val = it.getAvailableQty();
                break;

            case 2:
                val = it.getPrice();
                break;


        }

        System.out.println(it.getDescription());
    }
    i++;
}
//************************************************************//
return val;
}

/*
* JTable uses this method to determine the default renderer/ editor for
* each cell. If we didn't implement this method, then the last column
* would contain text ("true"/"false"), rather than a check box.
*/
/**
 * Returns the most specific superclass for all the cell values in the column.
 * @param c the index of the column
 * @return the class of the object values in the model.
 */
public Class getColumnClass(int c) {
    return getValueAt(0, c).getClass();
}

/*
* Don't need to implement this method unless our table's editable.
*/
/**
```

```
    * Returns true if the cell at rowIndex and columnIndex is editable. Otherwise, setValueAt on the cell will not change the value
of that cell.
    * @param row  the row whose value to be queried.
    * @param col the column whose value to be queried.
    * @return
    */
   public boolean isCellEditable(int row, int col) {
   //Note that the data/cell address is constant,
   //no matter where the cell appears onscreen.
      if (col < getColumnCount()) {
         return false;
      }
      else {
         return true;
      }
   }

   /*
   * Don't need to implement this method unless your table's data can
   * change.
   */
   public void setValueAt(Object value, int row, int col) {
   //      if (DEBUG) {
   //        ShoppingCartSystem.out.println("Setting value at " + row + "," + col
   //            + " to " + value + " (an instance of "
   //            + value.getClass() + ")");
   //      }
   //
   //      data[row][col] = value;
   //      fireTableCellUpdated(row, col);
   //
   //      if (DEBUG) {
   //        ShoppingCartSystem.out.println("New value of data:");
   //        printDebugData();
   //      }

   //        try {
   //      if (!rs.absolute(row + 1)) {
   //        return;
   //      }
   //      rs.updateObject(col + 1, value);
   //    } catch (SQLException e) {
   //    }
   }

   private void printDebugData() {
//    int numRows = getRowCount();
//    int numCols = getColumnCount();
//
//    for (int i = 0; i < numRows; i++) {
//    System.out.print("    row " + i + ":");
//    for (int j = 0; j < numCols; j++) {
//    //System.out.print("  " + data[i][j]);
//    System.out.print("  " + records.get(i));
//    }
//    System.out.println();
//    }
//    System.out.println("--------------------------");
   }

}
```

//cop4331\prj\src\cop4331\gui\RevenueReportUI.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.gui;
import cop4331.model.*;
import cop4331.model.ShoppingCartSystem.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import javax.swing.*;
//import static cop4331.model.ShoppingCartSystem.query;

/**
* A class that creates user interface to see the revenue report.
* @author Manisha
*/
public class RevenueReportUI extends JFrame
{
    private JFrame revFrame;

    private JLabel lblTotalRevenue;
    private JLabel lblTotalCost;
    private JLabel lblTotalProfit;

    private JLabel txtTotalRevenue;
    private JLabel txtTotalCost;
    private JLabel txtTotalProfit;

    private JButton btnBack;
    private JPanel revPanel;
    private JPanel mainPanel;
    private JPanel btnPanel;
    /**
     * constructor to create revenue report UI.
     * @param seller The Seller object- logged in seller.
     */
    public RevenueReportUI (Seller seller)
    {

        lblTotalRevenue = new JLabel("Total Revenue: ");
        lblTotalCost = new JLabel("Total Cost: ");
        lblTotalProfit = new JLabel("Total Profit: ");
        ArrayList<Integer> rev = ShoppingCartSystem.getInstance().retrieveSellerRevenueDetails(seller.getUserID());
        // txtTotalRevenue = new JTextField("", 15);
        txtTotalRevenue = new JLabel("");
        //txtTotalRevenue.setEnabled(false);
        txtTotalRevenue.setForeground(Color.DARK_GRAY);
        seller.setRevenue(rev.get(2));
//        txtTotalRevenue.setText(Integer.toString(rev.get(2)));
         txtTotalRevenue.setText(Double.toString(seller.getRevenue()));
        //  txtTotalCost = new JTextField("", 15);
        txtTotalCost =new JLabel("");
        // txtTotalCost.setEnabled(false);
        txtTotalCost.setText(Integer.toString(rev.get(1)));
        txtTotalCost.setForeground(Color.DARK_GRAY);
        // txtTotalProfit= new JTextField("", 15);
        txtTotalProfit= new JLabel("");
        if((rev.get(2)<rev.get(1)))
        txtTotalProfit.setForeground(Color.red);
```

```
        else
         txtTotalProfit.setForeground(Color.BLACK);
        // txtTotalProfit.setEnabled(false);
        txtTotalProfit.setText(Integer.toString(rev.get(0)));
        btnBack= new JButton("Back");
        btnBack.setSize(20, 20);
        revPanel= new JPanel(new GridLayout(4, 2));
        btnPanel = new JPanel(new FlowLayout());
        revPanel.add(lblTotalRevenue);
        revPanel.add(txtTotalRevenue);
        revPanel.add(lblTotalCost);
        revPanel.add(txtTotalCost);
        revPanel.add(lblTotalProfit);
        revPanel.add(txtTotalProfit);
        btnPanel.add(btnBack);


        revFrame = new JFrame("Revenue Report....");

        revFrame.setSize(400, 200);
        mainPanel= new JPanel(new BorderLayout());
        mainPanel.add(revPanel, BorderLayout.NORTH);
        mainPanel.add(btnPanel, BorderLayout.SOUTH);
        mainPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        revFrame.add(mainPanel);
        revFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //frame.pack();
        revFrame.setVisible(true);
        btnBack.addActionListener(new
        ActionListener()
        {
           public void actionPerformed(ActionEvent event)
           {
              revFrame.dispose();
           }
        });
     }


}
```

//cop4331\prj\src\cop4331\gui\ReviewUpdateCartUI.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.gui;

import cop4331.model.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Random;
import javax.swing.*;
//import static cop4331.model.ShoppingCartSystem.query;

/**
 * A class that gives interface for reviewing or updating the cart.
 *
 * @author Manisha
 */
public class ReviewUpdateCartUI
{

    private JTextField txtDescription;
    private JTextField txtPrice;
    private JTextField txtQty;
    private JTextField txtAmt;

    private JLabel lblDescription;
    private JLabel lblPrice;
    private JLabel lblQty;
    private JLabel lblAmt;
    private JLabel total;

    private JPanel productPanel;
    private JPanel btnPanel;
    private JPanel mainPanel;
    private JPanel middlePanel;

    private JButton btnprevious;
    private JButton btnNext;
    private JButton btnUpdateCart;
    private JButton btnReturn;
    private JButton btnPlaceOrder;
    int indexOfDisplayedProduct;

    /**
     * Constructor for for reviewUpdateCart UI.
     *
     * @param buyer the Buyer who is currently logged in
     */
    public ReviewUpdateCartUI(Buyer buyer)
    {
        ShoppingCart shoppingCart = buyer.getShoppingCart();
        ArrayList<LineItem> itemsToBuy = shoppingCart.getItemsToBuy();
        ArrayList<Integer> orderQty = shoppingCart.getQuantity();

        txtDescription = new JTextField("", 15);
        txtPrice = new JTextField("", 15);
        txtQty = new JTextField("", 15);
        txtAmt = new JTextField("", 15);
```

```
        lblDescription = new JLabel("Product Description:");
        lblDescription.setPreferredSize(new Dimension(150, 40));
        lblPrice = new JLabel("Price:");
        lblPrice.setPreferredSize(new Dimension(150, 40));

        lblQty = new JLabel("Quantity:");
        lblQty.setPreferredSize(new Dimension(150, 40));
        lblAmt = new JLabel("Amount:");
        lblAmt.setPreferredSize(new Dimension(150, 40));

        if (itemsToBuy.size() > 0) {
           indexOfDisplayedProduct = 0;
           txtDescription.setText(itemsToBuy.get(indexOfDisplayedProduct).getDescription());
           txtPrice.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice()));
           txtQty.setText(Integer.toString(orderQty.get(indexOfDisplayedProduct)));
           txtAmt.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice() *
orderQty.get(indexOfDisplayedProduct)));
           total = new JLabel("Total:  " + shoppingCart.calculateTotal());
        }
        btnprevious = new JButton("<<             ");
        btnprevious.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent event)
           {
              if (indexOfDisplayedProduct > 0) {
                 indexOfDisplayedProduct--;
                 txtDescription.setText(itemsToBuy.get(indexOfDisplayedProduct).getDescription());
                 txtPrice.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice()));
                 txtQty.setText(Integer.toString(orderQty.get(indexOfDisplayedProduct)));
                 txtAmt.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice() *
orderQty.get(indexOfDisplayedProduct)));
              }
           }
        });
        btnNext = new JButton("             >>");
        btnNext.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent event)
           {
              if (indexOfDisplayedProduct < itemsToBuy.size() - 1) {
                 indexOfDisplayedProduct++;
                 txtDescription.setText(itemsToBuy.get(indexOfDisplayedProduct).getDescription());
                 txtPrice.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice()));
                 txtQty.setText(Integer.toString(orderQty.get(indexOfDisplayedProduct)));
                 txtAmt.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice() *
orderQty.get(indexOfDisplayedProduct)));
              }
           }
        });
        btnUpdateCart = new JButton("UpdateCart");
        btnUpdateCart.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent event)
           {
              boolean isStockValidated = false;
              txtAmt.setText(Double.toString(itemsToBuy.get(indexOfDisplayedProduct).getPrice() *
Integer.parseInt(txtQty.getText())));
              if (itemsToBuy.get(indexOfDisplayedProduct).getAvailableQty() > Integer.parseInt(txtQty.getText())) {

                 buyer.updateQuantityAtIndex(indexOfDisplayedProduct, Integer.parseInt(txtQty.getText()));
                 JOptionPane.showMessageDialog(null, "Shopping Cart Updated!");
```

```
            total.setText("Total:  " + shoppingCart.calculateTotal());
        } else {
            JOptionPane.showMessageDialog(null, "Not enough available in stock!!reenter the order quantity.");
            txtQty.setText("");
            txtQty.grabFocus();
            total.setText("Total:  " + shoppingCart.calculateTotal());
        }
    }
});
btnReturn = new JButton("Return");
btnReturn.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {

    }
});

btnPlaceOrder = new JButton("ProceedToCheckout");
btnPlaceOrder.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        CheckOutUI checkOut = new CheckOutUI(buyer);
    }
});
productPanel = new JPanel(new GridLayout(6, 2));
productPanel.add(lblDescription);
productPanel.add(txtDescription);
productPanel.add(lblPrice);
productPanel.add(txtPrice);
productPanel.add(lblQty);
productPanel.add(txtQty);
productPanel.add(lblAmt);
productPanel.add(txtAmt);
// middlePanel = new JPanel(new GridLayout(1, 2));
JLabel space1 = new JLabel("              ");
JLabel space2 = new JLabel("              ");
productPanel.add(space1);
productPanel.add(total);
productPanel.add(btnprevious);
productPanel.add(btnNext);

btnPanel = new JPanel(new GridLayout(1, 3, 5, 5));
btnPanel.add(btnUpdateCart);
btnPanel.add(btnReturn);
btnPanel.add(btnPlaceOrder);
//btnPanel.add(total);
mainPanel = new JPanel(new BorderLayout());
mainPanel.add(productPanel, BorderLayout.CENTER);
//mainPanel.add(middlePanel, BorderLayout.CENTER);

mainPanel.add(btnPanel, BorderLayout.SOUTH);

mainPanel.setBorder(BorderFactory.createTitledBorder("Review Update Cart..."));
productPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));

JFrame frame = new JFrame("Review Update Cart");
frame.setSize(600, 400);
frame.add(mainPanel);
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //frame.pack();
        frame.setVisible(true);
        btnReturn.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                frame.dispose();
            }
        });
    }

    //////////private void getTxtFieldData(UsersofSystem p) {
    //////////
    //////////
    //////////      p.setUserName(username.getText());
    //////////      p.setPassword(password.getPassword().toString());
    //////////
    //////////
    //////////   }
    ////////////////////////////
    //public static void main(String[] args) {
    //    ReviewUpdateCartUI myUI = new ReviewUpdateCartUI();
    //}
    ///////////////////////
}
```

//cop4331\prj\src\cop4331\gui\SellerProductPage.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.gui;
import cop4331.model.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Random;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.AbstractTableModel;

/**
* This class provides user interface for seller to view his products for sell, add Line items, update Line Items.
* @author Manisha
*/
public class SellerProductPage extends JFrame
{

    private boolean DEBUG = false;
    private JPanel tblPanel;
    private JPanel txtPanel;
    //  private JLabel PID;
    private JLabel PDesc;
    private JLabel PQty;
    private JLabel PIRate;
    private JLabel PSRate;
    private JLabel PCategory;
    private JLabel PSID;

    //ADDED
    private JLabel PBID;
    private JLabel Pdiscount;

    //   private JTextField txtPID;
    private JTextField txtPDesc;
```

```
private JTextField txtPQty;
private JTextField txtPIRate;
private JTextField txtPSRate;
private JTextField txtPCategory;
private JTextField txtPSID;

//ADDED
private JTextField txtPBID;
private JTextField txtPdiscount;

private JButton btnAdd;
private JButton btnUpdate;
private JButton btnShowProfit;
private JButton btnSave;
private JButton btnCancel;

//ADDED
 private JButton btnBundle;
 private JButton btnDelete;

private JPanel btnPanel;
private JPanel mainPanel;
private JTable table ;
private JLabel label;
 Random rn = new Random();//ADDED
/**
 * Constructor that provides sellers product page.
 * @param seller The seller object who is logged into the system
 */
public SellerProductPage(Seller seller)
{
    tblPanel = new JPanel(new GridLayout(1, 0));
    txtPanel = new JPanel(new GridLayout(8, 2));//changed from 7,2 to 8,2 after ADDED
    btnPanel = new JPanel(new GridLayout(1, 5, 3, 3));////changed from 1,4,3,3 to 1,5,3,3 after ADDED
    //super(new GridLayout(1, 0));

    MyTableModel tableModel =  new MyTableModel(seller);

    table = new JTable(tableModel);
    table.setPreferredScrollableViewportSize(new Dimension(500, 100));

    //Create the scroll pane and add the table to it.
    JScrollPane scrollPane = new JScrollPane(table);

    //Add the scroll pane to this panel.
    tblPanel.add(scrollPane);

    //    PID = new JLabel("Product ID: ");
    PDesc = new JLabel("Product Name: ");
    PQty = new JLabel("Quantity Available: ");
    PIRate = new JLabel("Invoice Rate: ");
    PSRate = new JLabel("Sell Price: ");
    PCategory = new JLabel("Category: ");
    PSID = new JLabel("Seller ID: ");

    //ADDED
    PBID = new JLabel("Bundle ID(0 by default):");
    Pdiscount = new JLabel("Discount: ");


    //    txtPID = new JTextField(" ", 15);
    txtPDesc = new JTextField("", 15);
```

```
txtPQty = new JTextField("", 15);
txtPIRate = new JTextField("", 15);
txtPSRate = new JTextField("", 15);
txtPCategory = new JTextField("", 15);
txtPSID = new JTextField("", 15);
txtPSID.setText(Integer.toString(seller.getUserID()));

//ADDED
txtPBID = new JTextField("0", 15);
txtPdiscount = new JTextField("0", 15);

//   txtPanel.add(PID);
//   txtPanel.add(txtPID);
txtPanel.add(PDesc);
txtPanel.add(txtPDesc);
txtPanel.add(PQty);
txtPanel.add(txtPQty);
txtPanel.add(PIRate);
txtPanel.add(txtPIRate);
txtPanel.add(PSRate);
txtPanel.add(txtPSRate);
txtPanel.add(PCategory);
txtPanel.add(txtPCategory);
txtPanel.add(PSID);
txtPanel.add(txtPSID);

//ADDED
txtPanel.add(PBID);
txtPanel.add(txtPBID);
txtPanel.add(Pdiscount);
txtPanel.add(txtPdiscount);

label = new JLabel("");
btnAdd = new JButton("Add Product");
btnUpdate= new JButton("Update Product");
btnShowProfit= new JButton("Show Profit Report");
btnSave= new JButton("Save Product");
btnCancel= new JButton("Cancel");
//ADDED
btnBundle = new JButton("Bundle");
btnDelete = new JButton("Delete");

btnPanel.add(btnAdd);
btnPanel.add( btnUpdate);
btnPanel.add( btnShowProfit);
//ADDED
 btnPanel.add(btnBundle);
btnPanel.add(btnDelete);//4/19



btnPanel.add( btnCancel);
mainPanel= new JPanel(new BorderLayout());

mainPanel.add(tblPanel, BorderLayout.NORTH);
mainPanel.add(txtPanel, BorderLayout.CENTER);
mainPanel.add(btnPanel,BorderLayout.SOUTH );

mainPanel.setBorder(BorderFactory.createTitledBorder("Product Page"));
txtPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));
```

```
        ListSelectionModel selectionModel = table.getSelectionModel();
        selectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        selectionModel.addListSelectionListener(new RowListener(this));

        JFrame frame = new JFrame("Product Page");
        frame.setSize(500, 400);
        frame.add(mainPanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //frame.pack();
        frame.setVisible(true);

        //ADDED
        btnBundle.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                BundleUI bundleUI = new BundleUI(seller);
                bundleUI.setVisible(true);
            }
        });
        btnAdd.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                //TODO FIX USER ID PARAMTER

                int bundleID = 0;

                Product ourProduct;
                // ourProduct = new Product(5, txtPDesc.getText(),
                //    Double.parseDouble(txtPSRate.getText()), Integer.getInteger(txtPQty.getText()),
Double.parseDouble(txtPIRate.getText()), txtPCategory.getText(), 2);

                //if bundle.text != 0 and not null
                //if its a bundle, lineitme bundle
                ourProduct = new Product(rn.nextInt(Integer.MAX_VALUE) + 1, txtPDesc.getText(),
Double.parseDouble(txtPSRate.getText()),
                    Integer.parseInt(txtPQty.getText().trim()), Double.parseDouble(txtPIRate.getText().trim()),
                    txtPCategory.getText().trim(), seller.getUserID());

                if (txtPBID.getText() != null && !txtPBID.getText().isEmpty()) {
                    bundleID = Integer.parseInt(txtPBID.getText().trim());

                }

                seller.addNewItem(ourProduct, bundleID);

                JOptionPane.showMessageDialog(null,
                        "Prodcut saved!");
                tableModel.fireTableDataChanged();

                //TODO FIX USERID PARMATER
            }
        });

        //4/19
         btnDelete.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                if (tableModel.selectedProductID != 0) {
                    int productID = tableModel.selectedProductID;
```

```
                seller.deleteExistingItem(productID);
                //tableModel.fireTableDataChanged();
            }
        }
    });
    btnUpdate.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            if (tableModel.selectedProductID != 0) {
                int bundleID = 0;
                System.out.println(tableModel.selectedProductID);
                Product ourProduct;
                if (txtPBID.getText() != null && !txtPBID.getText().isEmpty()) {
                    bundleID = Integer.parseInt(txtPBID.getText());

                }
                ourProduct = new Product(tableModel.selectedProductID, txtPDesc.getText(),
Double.parseDouble(txtPSRate.getText()),
                        Integer.parseInt(txtPQty.getText()), Double.parseDouble(txtPIRate.getText()),
                        txtPCategory.getText(), seller.getUserID());
                seller.updateItem(ourProduct, bundleID);

                JOptionPane.showMessageDialog(null,
                        "Prodcut update made");
              //  tableModel.fireTableDataChanged();
            }

        }
    });


    ////////////////////////////////




    btnCancel.addActionListener(new
    ActionListener()
    {
    public void actionPerformed(ActionEvent event)
    {
    frame.dispose();
    }
    });
    btnShowProfit.addActionListener(new
    ActionListener()
    {
    public void actionPerformed(ActionEvent event)
    {
    RevenueReportUI revRepo = new RevenueReportUI(seller);
    }
    });
}



//////////////////////////////////////////
```

```
    ////////////////////////////////////
    private static class object {

    public object() {
    }
    }
/**
 * inner class
 */

    class RowListener implements ListSelectionListener
    {
        SellerProductPage readRow;
        JTable table;

        public RowListener(SellerProductPage rar)
        {
            readRow = rar;
            table = readRow.table;
        }
        /**
         * This method is called whenever the value of the selection changes
         * @param e the ListSelectionEvent that characterizes change
         */

        public void valueChanged(ListSelectionEvent e)
        {
            if(!e.getValueIsAdjusting())
            {
                ListSelectionModel model = table.getSelectionModel();
                int lead = model.getLeadSelectionIndex();
                displayRowValues(lead);
            }
        }
        /**
         * This method gets table row data
         * @param rowIndex
         */
        private void displayRowValues(int rowIndex)
        {
            int columns = table.getColumnCount();
            //String s = "";
            // String[] str = null;
            ArrayList<String> data= new ArrayList<String>();
            for(int col = 0; col < columns; col++)
            {
                Object o = table.getValueAt(rowIndex, col);
                data.add(col, o.toString());

            }

            //       readRow.txtPID.setText(data.get(0));
            readRow.txtPDesc.setText(data.get(0).trim());
            readRow.txtPQty.setText(data.get(1).trim());
            readRow.txtPIRate.setText(data.get(2).trim());
            readRow.txtPSRate.setText(data.get(3).trim());
            readRow.txtPCategory.setText(data.get(4).trim());
            readRow.txtPSID.setText(data.get(5).trim());
        }

        private void getTblRowInTxt(String str)
        {
```

```
        }

    }

}
/**
 * Inner class that extends AbstractTableModel which provides default implementations for most of the methods in the
TableModel interface.
 * @author Manisha
 */


class MyTableModel extends AbstractTableModel
{
    ResultSet rs;
    String query;
    private String[] columnNames = { "Product Name","Quantity","Invoice Price","Sell Price" ,"Category","Seller ID"};
    LinkedList<Object[]> records = new LinkedList<>();
    Seller currentSeller;
    //ADDED 4/19
    int selectedProductID;


    /**
        * Constructor for MyTableModel
        * @param cs The Seller current object that means a seller who is currently logged in.
        */

    public MyTableModel(Seller cs)
    {
        currentSeller = cs;
    }

 /**
  *
  * Returns the number of columns in the model. A JTable uses this method to determine how many columns it should create and
display by default.
  * @return int the number of columns in the model.
  */

    public int getColumnCount() {
        return columnNames.length;
    }
    /**
     *  Returns the number of rows in the model. A JTable uses this method to determine how many rows it should display.
     * @return int The number of rows in the model
     */

    public int getRowCount() {
        return currentSeller.getProductCount();
    }
    /**
     * Returns the name of the column at columnIndex.
     * @param col The columnIndex.
     * @return string the name of the column
     */

    public String getColumnName(int col) {
        return columnNames[col];
    }
    /**
```

```
 * Returns the value for the cell at columnIndex and rowIndex.
 * @param row the row whose value is to be queried.
 * @param col the column whose value is to be queried.
 * @return the value Object at the specified cell
 */


public Object getValueAt(int row, int col) {

    Inventory sellerInventory = currentSeller.getInventory();
    Object val = null;
    //************************Iterator Pattern Check***************//
    int i = 0;
    for(LineItem it:sellerInventory)
    {
       if( i == row )
       {
    selectedProductID = it.getProductId();
          System.out.println("works iterator");
          switch (col)
          {
             case 0:
             val = it.getDescription();
             break;
             case 1:
             val = it.getAvailableQty();
             break;
             case 2:
             val = it.getInvoicePrice();
             break;
             case 3:
             val = it.getPrice();
             break;
             case 4:
             val = it.getCategory();
             break;
             case 5:
             val = currentSeller.getUserID();
             break;
                ////////Miles4/19
             //case 6:
             //val = currentSeller.getProductId();
             //break;

          }

          // System.out.println(it.getDescription());
       }
       i++;
    }
    //************************************************************//
    return val;
}

/*
* JTable uses this method to determine the default renderer/ editor for
* each cell. If we didn't implement this method, then the last column
* would contain text ("true"/"false"), rather than a check box.
*/
/**
 * Returns the most specific superclass for all the cell values in the column.
 * @param c the index of the column
```

```
 * @return the class of the object values in the model.
 */

public Class getColumnClass(int c) {
    return getValueAt(0, c).getClass();
}

/*
* Don't need to implement this method unless your table's editable.
*/
public boolean isCellEditable(int row, int col) {
//the data/cell address is constant,
//no matter where the cell appears onscreen.
    if (col < getColumnCount()) {
        return false;
    }
    else {
        return true;
    }
}

/*
* Don't need to implement this method unless your table's data can
* change.
*/
public void setValueAt(Object value, int row, int col) {
//      if (DEBUG) {
//        ShoppingCartSystem.out.println("Setting value at " + row + "," + col
//            + " to " + value + " (an instance of "
//            + value.getClass() + ")");
//      }
//
//      data[row][col] = value;
//      fireTableCellUpdated(row, col);
//
//      if (DEBUG) {
//        ShoppingCartSystem.out.println("New value of data:");
//        printDebugData();
//      }

//        try {
//      if (!rs.absolute(row + 1)) {
//        return;
//      }
//      rs.updateObject(col + 1, value);
//    } catch (SQLException e) {
//    }
}

private void printDebugData() {
//      int numRows = getRowCount();
//      int numCols = getColumnCount();
//
//      for (int i = 0; i < numRows; i++) {
//        System.out.print("    row " + i + ":");
//        for (int j = 0; j < numCols; j++) {
//          //System.out.print("  " + data[i][j]);
//            System.out.print("  " + records.get(i));
//        }
//        System.out.println();
//      }
//      System.out.println("--------------------------");
```

```
    }
}
```

//cop4331\prj\src\cop4331\gui\UserProfileUI.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.gui;
import cop4331.model.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;
import javax.sql.rowset.JdbcRowSet;
import javax.swing.*;
import javax.swing.BorderFactory;
/**
 * A UI interface class that provides interface for user of the system to enter profile information.
 * @author Manisha
 */
public class UserProfileUI extends JFrame{
    private JFrame UPFrame;
    private JPanel UPPanel;
    private JPanel mainPanel;
    private JPanel btnPanel;

    private JLabel lblUName;
    private JLabel lblPWD;
    private JLabel lblFName;
    //private JLabel lblLName;
    private JLabel lblUAddress;
    private JLabel lblUCity;
    private JLabel lblUState;
    private JLabel lblUZip;
    private JLabel lblUPhone;
    private JLabel lblUEmail;
    private JLabel lblUserID;

     private JTextField TxtUName;
    private JTextField TxtPWD;
    private JTextField TxtFName;
    //private JTextField TxtLName;
    private JTextField TxtUAddress;
    private JTextField TxtUCity;
    private JTextField TxtUState;
    private JTextField TxtUZip;
    private JTextField TxtUPhone;
    private JTextField TxtUEmail;
    private JTextField TxtUserID;

    private JButton btnSaveProfile;
    private JButton btnUpdateProfile;
    private JButton btnContinue;

    private boolean isSeller;
    /**
     * Constructor for UserProfileUI
     * @param newUser  UsersofSystem
```

```
 * @param userType either "B" for buyer or "S for seller
 *
 */
public  UserProfileUI(UsersofSystem newUser, String userType)
{
   //UsersofSystem newUser = new Seller(Integer.getInteger(TxtUserID.getText().trim()).intValue());
    newUser.setUserID(new Random()
      .nextInt(Integer.MAX_VALUE) + 1);

    UserDBHandler userDB = new UserDBHandler();
    /*
    char[] pwd = password.getPassword();
         String passwrd = new String(pwd);
     */
  lblUserID = new JLabel("User ID: ");

   lblUName = new JLabel("User Name: ");
   lblPWD= new JLabel("Password: ");
   lblFName= new JLabel("Name: ");
   //lblLName= new JLabel("Last Name: ");
   lblUAddress= new JLabel("Address: ");
   lblUCity= new JLabel("City: ");
   lblUState= new JLabel("State: ");
   lblUZip= new JLabel("Zip Code: ");
   lblUPhone= new JLabel("Phone: ");
   lblUEmail= new JLabel("Email: ");

   TxtUserID = new JTextField("", 15);
   TxtUserID.setEnabled(false);
   TxtUName = new JTextField("", 15);
   TxtUName= new JTextField("", 15);
   TxtPWD= new JTextField("", 15);
   TxtFName= new JTextField("", 15);
   //TxtLName= new JTextField("", 15);
   TxtUAddress= new JTextField("", 15);
   TxtUCity= new JTextField("", 15);
   TxtUState= new JTextField("", 15);
   TxtUZip= new JTextField("", 15);
   TxtUPhone= new JTextField("", 15);
   TxtUEmail= new JTextField("", 15);

   btnSaveProfile = new JButton("Save Profile");
   btnUpdateProfile= new JButton("Update Profile");
   btnContinue= new JButton("Continue");

    mainPanel= new JPanel(new BorderLayout());
   btnPanel= new JPanel(new FlowLayout());
   UPPanel= new JPanel(new GridLayout(11, 2));


   UPPanel.add(lblUserID );
    UPPanel.add(TxtUserID);
    UPPanel.add(lblUName);
    UPPanel.add(TxtUName);
   UPPanel.add(lblPWD);
   UPPanel.add(TxtPWD);
   UPPanel.add(lblFName);
    UPPanel.add(TxtFName);
   //UPPanel.add(lblLName);
   //UPPanel.add(TxtLName);
   UPPanel.add(lblUAddress);
   UPPanel.add(TxtUAddress);
```

```
        UPPanel.add(lblUCity);
        UPPanel.add(TxtUCity);
        UPPanel.add(lblUState);
        UPPanel.add(TxtUState);
        UPPanel.add(lblUZip);
         UPPanel.add(TxtUZip);
        UPPanel.add(lblUPhone);
        UPPanel.add(TxtUPhone);
        UPPanel.add(lblUEmail);
        UPPanel.add(TxtUEmail);
       JFrame frame = new JFrame("User Profile");
    UPPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));


    TxtUserID.setText(Integer.toString(newUser.getUserID()));

    //btnPanel.setPreferredSize(new Dimension(40,40));
    btnPanel.add(btnSaveProfile);
    btnSaveProfile.addActionListener(new
        ActionListener()
        {
          public void actionPerformed(ActionEvent event)
          {
            //UserProfileUI userprofile = new  UserProfileUI();
              getTxtFieldData(newUser);

          if (isEmptyFieldData()) {
            JOptionPane.showMessageDialog(null,
            "Cannot create an empty record");
            return;
          }
          newUser.registerNewUser(userType);
          // newUser.create()
         //***********// userDB.create(p);
          JOptionPane.showMessageDialog(null,"New RECORD created successfully.");
//////////////            if (userDB.create(p) != null)
//////////////              JOptionPane.showMessageDialog(null,
//////////////              "New person created successfully.");

          }
        });
    btnPanel.add(btnUpdateProfile);
    btnUpdateProfile.addActionListener(new
        ActionListener()
        {
          public void actionPerformed(ActionEvent event)
          {
            //UserProfileUI userprofile = new  UserProfileUI();
              getTxtFieldData(newUser);

          if (isEmptyFieldData()) {
            JOptionPane.showMessageDialog(null,
            "Cannot create an empty record");
            return;
          }
          newUser.updateUserProfile(Integer.parseInt(TxtUserID.getText()));
         //***********// userDB.create(p);
          JOptionPane.showMessageDialog(null,"Record Updated!");
//////////////            if (userDB.create(p) != null)
//////////////              JOptionPane.showMessageDialog(null,
//////////////              "New person created successfully.");
```

```
            }
        });
    btnPanel.add(btnContinue);
    btnContinue.addActionListener(new
        ActionListener()
        {
          public void actionPerformed(ActionEvent event)
          {
             if(userType.equals("S"))
                //SellerProductPage InventoryPage =
                 //new SellerProductPage((Seller)newUser);
             {
                JOptionPane.showMessageDialog(null,"Please login again");
                frame.dispose();
             }
             else
             {
                JOptionPane.showMessageDialog(null,"Please login again");
                frame.dispose();
                 //new ProductCatalogUI((Buyer)newUser);
             }

          }
        });


    /*
    btnSaveProfile.addActionListener(new
        ActionListener()
        {
          public void actionPerformed(ActionEvent event)
          {
            textField.setText("Hello, World!");
          }
        });
    */
    mainPanel.add(UPPanel, BorderLayout.CENTER);
    mainPanel.add(btnPanel, BorderLayout.SOUTH);

     mainPanel.setBorder(BorderFactory.createTitledBorder("User Profile"));
    UPPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
    btnPanel.setBorder(BorderFactory.createLineBorder(Color.black));

//       UPPanel.add(UPlblPanel);
//       UPPanel.add(UPtxtPanel);

       frame.setSize(500, 400);
       frame.add(mainPanel);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//frame.pack();
       frame.setVisible(true);
//       p.setUserID(new Random()
//           .nextInt(Integer.MAX_VALUE) + 1);
    }
//    public static void main(String[] args)
//    {
//       UserProfileUI UserProfileUI = new UserProfileUI();
//
//    }
    /**
     * A method to get text field data into the model class object UsersofSystem
     * @param p object of UsersofSystem
```

```
     */
    private void getTxtFieldData(UsersofSystem p) {

      p.setUserID(Integer.parseInt(TxtUserID.getText()));
      p.setUserName(TxtUName.getText());
      p.setPassword(TxtPWD.getText());
      p.setName(TxtFName.getText());
      //p.setEmail(TxtLName.getText());
      p.setAddress(TxtUAddress.getText());
       p.setCity(TxtUCity.getText());
      p.setState(TxtUState.getText());
      p.setZip(TxtUZip.getText());
      p.setPhone(TxtUPhone.getText());
      p.setEmail(TxtUEmail.getText());

    }
     /**
      * a method to sett text field data with object of UsersofSystem
      * @param p object of UsersofSystem
      */
    private void setTxtFieldData(UsersofSystem p) {
      TxtUserID.setText(String.valueOf(p.getUserID()));
      TxtUName.setText(p.getUserName());
      TxtPWD.setText(p.getPassword());
      TxtFName.setText(p.getName());
      TxtUAddress.setText(p.getAddress());
      TxtUCity.setText(p.getCity());
      TxtUState.setText(p.getState());
      TxtUZip.setText(p.getZip());
      TxtUPhone.setText(p.getPhone());
      TxtUEmail.setText(p.getEmail());
    }
/**
 * The method to check if textField is empty
 * @return true if any of the text filed is empty false otherwise.
 */
    private boolean isEmptyFieldData() {
      return (TxtUserID.getText().trim().isEmpty()
         && TxtUName.getText().trim().isEmpty()
         && TxtPWD.getText().trim().isEmpty()
         && TxtFName.getText().trim().isEmpty()
         && TxtUAddress.getText().trim().isEmpty()
         && TxtUCity.getText().trim().isEmpty()
         && TxtUState.getText().trim().isEmpty()
         && TxtUZip.getText().trim().isEmpty()
         && TxtUPhone.getText().trim().isEmpty()
         && TxtUEmail.getText().trim().isEmpty());
    }



    public class UserDBHandler {
    ResultSet rs1;
     String query;
    Statement s;
    // private JdbcRowSet rs1 = null;
     public UserDBHandler(){
        //************************************************************//
//       try {
//          Class.forName("org.apache.derby.jdbc.ClientDriver");
//       } catch (ClassNotFoundException cnfe) {
//          System.err.println("Derby driver not found.");
```

```
//        }
//        try {
//            String url="jdbc:derby://localhost:1527/myDB2";
//            String username ="manisha";
//            String password = "manisha";
//            Connection con = DriverManager.getConnection(url,username,password);
//           // Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/myDB;user=manisha;pass=manisha");
//             s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
//                                 ResultSet.CONCUR_READ_ONLY);
           //*********************************************************************//
           /////////query ="INSERT INTO PRODUCT (PRODUCT_ID, PRODUCT_DESC, QUANTITY, INV_PRICE,
SELL_PRICE,CATEGORY, SELLERID)VALUES (3, 'Square Dining Table', 20, 1045.99, 1250.00,
'DINING',1)";////////////////////////////////
           //s.execute("CREATE TABLE test (id integer primary key not null, text varchar(32))");
           //s.execute("INSERT INTO BUYER VALUES ('" + txtid.getText() + "' , ' " + usertypecombo.getSelectedItem() + ","'+
nametxt.getText +"')";
           //s.execute("SELECT /* FROM test");
          ///////// s.execute(query);//////////////////////////////////////////////
          ///////// JOptionPane.showMessageDialog(null, "Record Added");///////////////
            //query = "SELECT * FROM USERS";

           //rs1 =  s.executeQuery(query);


//           int cols = rs.getMetaData().getColumnCount();
//while(rs.next()){
//    Object[] arr = new Object[cols];
//    for(int i=0; i<cols; i++){
//     arr[i] = rs.getObject(i+1);
//
//    }
////    ListSelectionModel selectionModel = table.getSelectionModel();
////       selectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
//       //selectionModel.addListSelectionListener(new RowListener(this));
//}
         //*********************************************************************//
//       } catch (SQLException ex) {
//           ex.printStackTrace();
//       }
      //*********************************************************************//
  }

//   public UsersofSystem create(UsersofSystem p) {
//      try {
//          String url="jdbc:derby://localhost:1527/myDB";
//            String username ="manisha";
//            String password = "manisha";
//            Connection con = DriverManager.getConnection(url,username,password);
////           // Connection conn =
DriverManager.getConnection("jdbc:derby://localhost:1527/myDB;user=manisha;pass=manisha");
////            Statement s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
////                                ResultSet.CONCUR_READ_ONLY);
//
//          p.registerNewUser();
//          query = "INSERT INTO USERS (USERID, NAME, USERNAME, PASSWORD, ADDRESS,CITY, ZIPCODE,
STATE, PHONE, EMAIL) "
//                  + "VALUES (" + p.getUserID() + " , ' " +   p.getName() + "' , ' " +   p.getUserName()
//                  + "' , ' " +   p.getPassword()+ "' , ' " +   p.getAddress()+ "' , ' " +   p.getCity()
//                  + "' , ' " +   p.getZip()+ "' , ' " +   p.getState()+ "' , ' " +   p.getPhone()
//                  + "' , ' " +   p.getEmail()+"')";
//          //String insertNewUserSQL = "INSERT INTO USERS  VALUES (?, ?, ?, ?, ?, ?, ?)";
//          PreparedStatement pstmt = con.prepareStatement(query);
```

```
//          //rs1 = s.executeQuery(query);
//          pstmt.executeUpdate();
//
//
//      } catch (SQLException ex) {
//
//          ex.printStackTrace();
//      }
//      return p;
//    }

   public UsersofSystem update(UsersofSystem p) {
try {
          String url="jdbc:derby://localhost:1527/myDB";
            String username ="manisha";
            String password = "manisha";
            Connection con = DriverManager.getConnection(url,username,password);
//          // Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/myDB;user=manisha;pass=manisha");
//           Statement s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
//                                ResultSet.CONCUR_READ_ONLY);

         ////////////////////////////////////////// p.registerNewUser(userType);
          query = "Update USERS set NAME ='" +  p.getName() + "', set USERNAME ='" + p.getUserName() +"',set PASSWORD
='" + p.getPassword()+ "',set ADDRESS ='" + p.getCity() +"', set CITY ='"
                  + p.getCity() +"', set ZIPCODE ='," + p.getZip() +"',set STATE ='" + p.getState()+ "',set PHONE ='" +  p.getPhone()
+"', set EMAIL='" + p.getEmail()+"'";

          //String insertNewUserSQL = "INSERT INTO USERS  VALUES (?, ?, ?, ?, ?, ?, ?)";
          PreparedStatement pstmt = con.prepareStatement(query);
           //rs1 = s.executeQuery(query);
          pstmt.executeUpdate();


      } catch (SQLException ex) {

          ex.printStackTrace();
      }
      return p;
    }

    public void delete() {
      try {
          rs1.moveToCurrentRow();
          rs1.deleteRow();
           }
                catch (SQLException ex) {
          ex.printStackTrace();
           }

                }

 }
}
```

//cop4331\prj\src\cop4331\model\ShoppingCartSystem.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;

import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.LinkedList;
import javax.swing.JOptionPane;

/**
 * ShoppingCartSystem is a singleton class to make sure only single object gets
 * created This class provides a way to access its only object which can be
 * accessed directly without need to instantiate the object of the class.
 *
 * @author Manisha
 */
public class ShoppingCartSystem
{

    //create an object of ShoppingCartSystem
    public static ShoppingCartSystem instance = new ShoppingCartSystem();
    Connection con;
    ////// private ArrayList<User> allSellers;
    static Object err;
    static ResultSet rs;
    static ResultSet rs1;
    static ResultSet rs2;
    static ResultSet rs3;
    static String query;
    static Statement s;
    static PreparedStatement pstmt;
//    public void connect()
//    {
//
//        try {
//            Class.forName("org.apache.derby.jdbc.ClientDriver");
//        } catch (ClassNotFoundException cnfe) {
//            System.err.println("Derby driver not found.");
//        }
//        try {
//            String url="jdbc:derby://localhost:1527/myDB";
//            String username ="manisha";
//            String password = "manisha";
//            Connection con = DriverManager.getConnection(url,username,password);
//           // Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/myDB;user=manisha;pass=manisha");
//            s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
//                                    ResultSet.CONCUR_READ_ONLY);
//            //query = "SELECT * FROM PRODUCT";
//
//            //rs =  s.executeQuery(query);
//
//        } catch (SQLException ex) {
//            ex.printStackTrace();
//        }
//    }
    //make the constructor private so that this class cannot be
    //instantiated

    /**
     * Constructor for shoppingCartSystem that initiates the database.
     */
    private ShoppingCartSystem()
```

```
    {
        con = null;
        initDatabase();
    }

    /**
     * returns the only object/instance available to use for singleton.
     *
     * @return instance of ShoppingCartSystem class
     */
    public static ShoppingCartSystem getInstance()
    {
        return instance;
    }

    public Seller getSeller(int USERID)
    {
        return null;
    }

    public Buyer getBuyer(int USERID)
    {
        return null;
    }

    public void setSeller()
    {

    }

    public void setBuyer()
    {

    }

    /**
     * This method connects to database.
     *
     * @throws Exception
     */
    private void initDatabase()
    {
        if (con == null) {
            try {
                Class.forName("org.apache.derby.jdbc.EmbeddedDriver");//original
                //***************was trying*************
                // String applicationDirPath = "C:/Users/Manisha/Documents/NetBeansProjects/COP4331/ShoppingCartProject";
                //  System.setProperty("derby.system.home", applicationDirPath);
//                System.setProperty("derby.system.home",
//    System.getProperty("user.home")+".netbeans-derby");
                //*****************************************

//                                        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");//derby embedded
driver//working embedded
            } catch (Exception e) {//catch (ClassNotFoundException cnfe) {
                System.err.println("Derby driver not found.");
            }
            try {
//                String url = "jdbc:oracle:thin:@131.91.168.91:1521:r11g";
//                String username = "mrobson3";
//                String password = "fau5096";
//                con = DriverManager.getConnection(url, username, password);
```

```
            // String url = "jdbc:derby://localhost:1527/myDB2";//original
            //  String username = "manisha";//original
            //  String password = "manisha";//original
            con = DriverManager.getConnection("jdbc:oracle:thin:@131.91.168.91:1521:r11g", "mrobson3", "fau5096");//original

//              con = DriverManager.getConnection("jdbc:derby:C:\\Users\\Manisha\\.netbeans-derby\\myDB2", "manisha",
"manisha");  //working embedded
//                                          con = DriverManager.getConnection("jdbc:derby:.\\myDB2", "manisha", "manisha");   //
Use during submission jar is running
            s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                  ResultSet.CONCUR_READ_ONLY);
         } catch (SQLException ex) {
            ex.printStackTrace();
         }
      }
   }

   /**
    * This method closes database connection.
    */
   private void closeDatabaseConnection()
   {
      // close connecion
      con = null;
   }

//    public void updateSellerOrBuyer(String query) {
//
//    }
   /**
    * This method adds or update users of the system seller or buyer into the
    * database. It uses query supplied.
    *
    * @param query a String query to insert or update record in a database.
    */
   public void addUpdateSellerOrBuyer(String query)
   {

      try {
         PreparedStatement pstmt = con.prepareStatement(query);
         //rs1 = s.executeQuery(query);
         pstmt.executeUpdate();
      } catch (SQLException ex) {
         ex.printStackTrace();
      }
   }

   /**
    * The method to retrieve user profile information and set the current user
    * with the information retrieved using the query.
    *
    * @param currentUser The UsersofSystem object, he may be seller or buyer.
    * @param query the sql query that retreives the user profile information
    * for buyer or seller.
    */
   public void retrieveUser(UsersofSystem currentUser, String query)
   {
      try {
         rs = s.executeQuery(query);
         if (rs.next()) {
            currentUser.setName(rs.getString(2));
```

```
                currentUser.setCity(rs.getString(6));
                currentUser.setAddress(rs.getString(5));
                currentUser.setEmail(rs.getString(10));
                currentUser.setPassword(rs.getString(4));
                currentUser.setPhone(rs.getString(9));
                currentUser.setState(rs.getString(8));
                currentUser.setUserID(rs.getInt(1));
                currentUser.setZip(rs.getString(7));
                currentUser.setUserName(rs.getString(3));
//              if(currentUser.userType.equals("S"))
//              {
//                  for(LineItem it:sellerInventory)
////          {
////              System.out.println("works iterator");
////              System.out.println(it.getDescription());
////          }
//
//              }

            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

    }

    /**
     * The method to retrieve all products or bundle or discounted Line Items.
     *
     * @return the arrayList containing all Line Items.
     */
    public ArrayList<LineItem> retrieveAllProducts()
    {
        String query1 = "select * from Product";
        ArrayList<LineItem> availableProducts = new ArrayList<LineItem>();
        try {
            rs1 = s.executeQuery(query1);
            rs1.next();
            int userid = rs1.getInt(1);
            String query2 = "select DISTINCT BUNDLE_ID from PRODUCT";
            rs2 = s.executeQuery(query2);
            ArrayList<Integer> bundleNumbers = new ArrayList<Integer>();
            while (rs2.next()) {
                bundleNumbers.add(rs2.getInt(1));
            }

            for (int bundleNumber : bundleNumbers) {
                if (bundleNumber == 0) {
                    String query3 = "select PRODUCT_ID, PRODUCT_DESC, QUANTITY, "
                            + "INV_PRICE, SELL_PRICE, CATEGORY, DISCOUNT, USERID "
                            + "FROM PRODUCT WHERE BUNDLE_ID = " + bundleNumber;
                    rs3 = s.executeQuery(query3);
                    while (rs3.next()) {
                        if (rs3.getInt(7) == 0) {
                            // create a product
                            Product prod = new Product(Integer.parseInt(rs3.getString(1)), rs3.getString(2), rs3.getDouble(5), rs3.getInt(3),
                                    rs3.getDouble(4), rs3.getString(6), rs3.getInt(8));
                            availableProducts.add(prod);
                        } else {
                            // create discounted product
                            Product prod = new Product(Integer.parseInt(rs3.getString(1)), rs3.getString(2), rs3.getDouble(5), rs3.getInt(3),
                                    rs3.getDouble(4), rs3.getString(6), rs3.getInt(8));
```

```
                    DiscountedItem discProd = new DiscountedItem(prod, rs3.getInt(7));
                    availableProducts.add(discProd);
                }
            }
        } else // bundle number != 0
        {
            String query3 = "select PRODUCT_ID, PRODUCT_DESC, QUANTITY, "
                + "INV_PRICE, SELL_PRICE, CATEGORY, DISCOUNT, USERID "
                + "FROM PRODUCT WHERE BUNDLE_ID = " + bundleNumber;
            rs3 = s.executeQuery(query3);
            Bundle bundle = new Bundle();
            int discount = 0;
            while (rs3.next()) {
                if (rs3.getInt(7) > 0) {
                    discount = rs3.getInt(7);
                }
                Product prod = new Product(Integer.parseInt(rs3.getString(1)), rs3.getString(2), rs3.getDouble(5), rs3.getInt(3),
                        rs3.getDouble(4), rs3.getString(6), rs3.getInt(8));
                bundle.add(prod);
            }
            if (discount > 0) {
                DiscountedItem discBundle = new DiscountedItem(bundle, discount);
                availableProducts.add(discBundle);
            } else {
                availableProducts.add(bundle);
            }
        }
    }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return availableProducts;

}

/**
 * The method that retrieves seller revenue details
 *
 * @param userId the Seller id
 * @return arrayList of integer that holds revenue information like profit,
 * total invoice price, total revenue.
 */
public ArrayList<Integer> retrieveSellerRevenueDetails(int userId)
{
    String query1 = "select PROFIT, TOTAL_INV_PRICE, TOTAL_SELL_PRICE from SELLER where USERID = " +
Integer.toString(userId);

    ArrayList<Integer> revenueDetails = new ArrayList<Integer>();
    try {
        rs1 = s.executeQuery(query1);
        rs1.next();
        revenueDetails.add(rs1.getInt(1));
        revenueDetails.add(rs1.getInt(2));
        revenueDetails.add(rs1.getInt(3));
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return revenueDetails;
}

/**
```

```
 * method to retrieve seller inventory from database
 *
 * @param username the seller username.
 * @return ArraList with all LineItems in seller inventory
 */
public ArrayList<LineItem> retrieveSellerInventory(String username)
{
   String query1 = "select USERID from USERS where USERNAME ='" + username + "'";

   ArrayList<LineItem> itemsInInventory = new ArrayList<LineItem>();
   try {
      rs1 = s.executeQuery(query1);
      rs1.next();
      int userid = rs1.getInt(1);
      String query2 = "select DISTINCT BUNDLE_ID from PRODUCT where USERID = " + userid;
      rs2 = s.executeQuery(query2);
      ArrayList<Integer> bundleNumbers = new ArrayList<Integer>();
      while (rs2.next()) {
         bundleNumbers.add(rs2.getInt(1));
      }

      for (int bundleNumber : bundleNumbers) {
         if (bundleNumber == 0) {
            String query3 = "select PRODUCT_ID, PRODUCT_DESC, QUANTITY, "
                  + "INV_PRICE, SELL_PRICE, CATEGORY, DISCOUNT, USERID "
                  + "FROM PRODUCT WHERE BUNDLE_ID = " + bundleNumber
                  + "  AND USERID = " + userid;
            rs3 = s.executeQuery(query3);
            while (rs3.next()) {
               if (rs3.getInt(7) == 0) {
                  // create a product
                  Product prod = new Product(Integer.parseInt(rs3.getString(1)), rs3.getString(2), rs3.getDouble(5), rs3.getInt(3),
                        rs3.getDouble(4), rs3.getString(6), rs3.getInt(8));
                  itemsInInventory.add(prod);
               } else {
                  // create discounted product
                  Product prod = new Product(Integer.parseInt(rs3.getString(1)), rs3.getString(2), rs3.getDouble(5), rs3.getInt(3),
                        rs3.getDouble(4), rs3.getString(6), rs3.getInt(8));
                  DiscountedItem discProd = new DiscountedItem(prod, rs3.getInt(7));
                  itemsInInventory.add(discProd);
               }
            }
         } else // bundle number != 0
         {
            String query3 = "select PRODUCT_ID, PRODUCT_DESC, QUANTITY, "
                  + "INV_PRICE, SELL_PRICE, CATEGORY, DISCOUNT,USERID "
                  + "FROM PRODUCT WHERE BUNDLE_ID = " + bundleNumber
                  + "  AND USERID = " + userid;
            rs3 = s.executeQuery(query3);
            Bundle bundle = new Bundle();
            int discount = 0;
            while (rs3.next()) {
               if (rs3.getInt(7) > 0) {
                  discount = rs3.getInt(7);
               }
               Product prod = new Product(Integer.parseInt(rs3.getString(1)), rs3.getString(2), rs3.getDouble(5), rs3.getInt(3),
                     rs3.getDouble(4), rs3.getString(6), rs3.getInt(8));
               bundle.add(prod);
            }
            if (discount > 0) {
               DiscountedItem discBundle = new DiscountedItem(bundle, discount);
               itemsInInventory.add(discBundle);
```

```
                } else {
                    itemsInInventory.add(bundle);
                }
            }
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return itemsInInventory;
    //return null;
}

/**
 * The method to update seller inventory/stock of purchased item
 *
 * @param shoppingCart the ShoppingCart object.
 */
public void updateInventory(ShoppingCart shoppingCart)
{
    ArrayList<String> queries = shoppingCart.getUpdateInventoryQueries();
    for (String query : queries) {
        try {
            s.executeUpdate(query);
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

/**
 * A method to update seller inventory of purchased items.
 *
 * @param shoppingCart the ShoppingCart object.
 */
public void updateSellerRevenue(ShoppingCart shoppingCart)
{
    ArrayList<String> queries = shoppingCart.getUpdateSellerRevenueQueries();
    for (String query : queries) {
        try {
            s.executeUpdate(query);
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

////ADDED
public void addItem(String query)
{

    try {
        PreparedStatement pstmt = con.prepareStatement(query);
        //rs1 = s.executeQuery(query);
        pstmt.executeUpdate();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

/**
 * Intakes a query to delete an Item from the database
```

```
 *
 * @param query the string holding query to database
 */
public void deleteItem(String query)
{

   try {
      PreparedStatement pstmt = con.prepareStatement(query);
      //rs1 = s.executeQuery(query);
      pstmt.executeUpdate();
   } catch (SQLException ex) {
      ex.printStackTrace();
   }
}
//Miles 4/19

/**
 * Intakes a query to update an Item from the database
 *
 * @param query the string holding query to database
 */
public void updateItem(String query)
{

   try {
      PreparedStatement pstmt = con.prepareStatement(query);
      //rs1 = s.executeQuery(query);
      pstmt.executeUpdate();
   } catch (SQLException ex) {
      ex.printStackTrace();
   }
}
//Miles 4/19

/**
 * The method to check if the user is a valid user of the system or not
 *
 * @param unm the username of user
 * @param pwd the password of the system.
 * @return the type of user "S" for seller and "B" for buyer.
 */
public String loginVerify(String unm, String pwd)
{

   int count = 0;
   String usertype = "";
   String usernm = "";
   String passwd = "";
   boolean isFound = false;
   try {
      // PreparedStatement pstmt = con.prepareStatement(query);
      query = "SELECT * from USERS";
      rs = s.executeQuery(query);
      // count = pstmt.executeUpdate();
      while (rs.next()) {
         count = rs.getRow();

         usernm = rs.getString(3).trim();
         passwd = rs.getString(4).trim();
         usertype = rs.getString(11);
                              if ((usernm.trim().equals(unm.trim())) && (passwd.trim().equals(pwd.trim()))) {
            {
```

```
                    isFound = true;
                    break;
                }
            }

        }

//          if(pstmt.getMetaData().equals(usertype)){
//              usertype = "S";
//          }
//          else{
//              usertype = "B";
//          }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        //if(count > 0){
        if (isFound) {
            JOptionPane.showMessageDialog(null,
                    "Login verified");

        } else {
            JOptionPane.showMessageDialog(null,
                    "Invalid Login");
            // System.exit(0);
            usertype = "N";
        }

        return usertype;
    }

}
```

//cop4331\prj\src\cop4331\model\ShoppingCart.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;
import cop4331.gui.*;
import java.util.*;
import static cop4331.model.ShoppingCartSystem.query;

/**
 * This is a model class that holds shopping cart information. and gives the required functionality for shopping cart.
 * @author Manisha
 */
public class ShoppingCart
{

    private ArrayList<LineItem> ItemsToBuy;
    int cartID;
    ArrayList<Integer> orderQty;
    Order order;
    private ArrayList<String> updateInventoryQueries;
    /**
     * Constructor for shopping cart that creates shopping cart object.
     */
    public  ShoppingCart()
    {
        ItemsToBuy = new ArrayList<LineItem>();
        cartID = new Random().nextInt(Integer.MAX_VALUE) + 1;
        order = null;
        orderQty = new ArrayList<Integer> ();
        updateInventoryQueries = new ArrayList<String>();

    }
    /**
     * adds the Line item to shopping cart with the given quantity.
     * @param item the Line Item to be added
     * @param quantity the quantity of corresponding Line Item.
     */
    public void addItemToCart(LineItem item, int quantity){
        ItemsToBuy.add(item);
        orderQty.add(quantity);
    }
    /**
     * Updates quantity of the Line Item at specified index in the ItemsToBuy list.
     * @param index The index in the list.
     * @param quantity the quantity of item at specified index.
     * precondition ItemsToBuy.size()greater than 0
     * postcondition ItemsToBuy.get(index)= quantity
     */
    public void updateQuantityAtIndex(int index, int quantity)
    {
        if(quantity > 0)
        {
            orderQty.set(index, quantity);
        }
        else
        {
            orderQty.remove(index);
            ItemsToBuy.remove(index);
```

```
        }
    }
    /**
     *
     * @param Item
     */
//    public void deleteItemFromCart(LineItem Item){
//
//    }
//    public void updateCart(){
//
//    }
    /**
     * Calculates price for every LineItem in the shopping cart list itemsToBuy.
     * @return list of calculated Price For Each Item.
     * precondition ItemsToBuy.size()greater than 0
     * postcondition arrayList of calculated Price For Each Item is ready for use.
     */
    public ArrayList<Double> calculateUnitPrice(){
        int j =0;
        ArrayList<Double> calculatedPriceForEachItem = new ArrayList<Double>();
            for(LineItem i:ItemsToBuy)
            {
                calculatedPriceForEachItem.add(j, i.getPrice()* orderQty.get(j));
                j++;
            }
        return calculatedPriceForEachItem;

    }
    /**
     * calculates of total shopping cart price.
     * @return double the total price of all items in shopping cart.
     * precondition ItemsToBuy.size() greater than 0
     */
    public double calculateTotal(){
        double total = 0.0;

        ArrayList<Double> PriceForEachItem =calculateUnitPrice();
        for(int i = 0; i < PriceForEachItem.size(); i++)
        {
            total = total + PriceForEachItem.get(i);

        }

        return total;
    }
    /**
     * creates the order for buyer.
     * @return Order object.
     */
    public Order createOrder()
    {
        order = new Order(cartID, ItemsToBuy);
        return order;
    }


    /**
     * Gets the list of items to purchase.
     * @return ArrayList of LineItem the list of items to buy.
     *
     */
```

```
public ArrayList<LineItem> getItemsToBuy()
{
   return ItemsToBuy;
}
/**
 * Gets the list of quantities of items to purchase.
 * @return ArrayList of LineItem the list of quantities of items to buy.
 *
 */
public ArrayList<Integer> getQuantity()
{
   return orderQty;
}
/**
 * Gets queries needed to update the inventories/stocks of items after items are purchased.
 * @return ArrayList of update queries for adjusting stock/inventories of Line Items bought.
 */
public ArrayList<String> getUpdateInventoryQueries()
{
   if( updateInventoryQueries.size() > 0)
   {
      updateInventoryQueries.clear();
   }

   int j = 0;
   for(LineItem item:ItemsToBuy)
   {
      item.getUpdateInventoryQueries(updateInventoryQueries, orderQty.get(j));
      j++;
   }
   return updateInventoryQueries;
}
/**
 * Gets queries for updating seller revenue.
 * @return ArrayList of queries to adjust revenue.
 */
public ArrayList<String> getUpdateSellerRevenueQueries()
{
   ArrayList<String> updateSellerInventoryQueries = new ArrayList<String>();
   ArrayList<Integer> sellerIds = new ArrayList<Integer>();

   for(LineItem item:ItemsToBuy)
   {
      int sellerId = item.getSellerId();
      boolean found = false;
      for(Integer id:sellerIds)
      {
         if(id == sellerId)
         {
            found = true;
            break;
         }
      }
      if(!found)
      {
         sellerIds.add(sellerId);
      }
   }


   for(Integer id:sellerIds)
   {
```

```
        double totalSellPrice = 0;
        double totalInvPrice = 0;
        double profit =0;
        int j = 0;
        for(LineItem item:ItemsToBuy)
        {
           if( id == item.getSellerId() )
           {
              totalSellPrice += (item.getPrice() * orderQty.get(j));
              totalInvPrice += (item.getInvoicePrice() * orderQty.get(j));
              profit = totalSellPrice - totalInvPrice;
           }
           j++;
        }

        String query = "UPDATE SELLER SET PROFIT = PROFIT + " + Double.toString(profit)
        + " WHERE USERID = " + id;
        updateSellerInventoryQueries.add(query);
        query = "UPDATE SELLER SET TOTAL_INV_PRICE = TOTAL_INV_PRICE + " + Double.toString(totalInvPrice)
        + " WHERE USERID = " + id;
        updateSellerInventoryQueries.add(query);
        query = "UPDATE SELLER SET TOTAL_SELL_PRICE = TOTAL_SELL_PRICE + " + Double.toString(totalSellPrice)
        + " WHERE USERID = " + id;
        updateSellerInventoryQueries.add(query);
     }

     return updateSellerInventoryQueries;
  }

}
```

//cop4331\prj\src\cop4331\model\Seller.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.model;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.LinkedList;
import cop4331.gui.*;
import static cop4331.model.ShoppingCartSystem.query;
import static cop4331.model.ShoppingCartSystem.rs;
import static cop4331.model.ShoppingCartSystem.s;
/**
* A class that holds seller information.
* @author Manisha
*/
public class Seller extends UsersofSystem
{

    Inventory sellerInventory;
    private double revenue;
    /**
     * Constructor with no arguments for seller.
     */
    public Seller()
    {
        revenue = 0.0;
        super.userType = "S";
    }
    /**
     * Constructor that creates seller
     * @param usernm The seller name.
     * @param password The password.
     */
    public Seller(String usernm, String password)
    {
        revenue = 0.0;
        super.userType = "S";
        query = "select * from USERS where USERNAME ='" + usernm + "'and PASSWORD = '" + password + "'";
        ShoppingCartSystem.getInstance().retrieveUser(this, query);
        sellerInventory = new Inventory(usernm);
        //************************Iterator Pattern Check****************//
        //        for(LineItem it:sellerInventory)
        //        {
        //
        //            System.out.println("works iterator");
        //            System.out.println(it.getDescription());
        //        }
        //*************************************************************//
    }
    public void updateProfile(){

    }
    /**
     * Returns the number of products in seller inventory.
     * @return the count of product.
     * precondition sellerInventory.size() is greater than 0
     */
    public int getProductCount()
```

```
    {
        return sellerInventory.getProductCount();
    }
    /**
     * Gets the seller inventory
     * @return  The inventory object in current seller. It has list of sellers line Item.
     */
    public Inventory getInventory()
    {
        return sellerInventory;
    }
    /**
     * A method to add new Line item to sellers inventory
     * @param item Line item to be added to seller inventory.
     * @param bundleID the bundle id
     * postcondition sellerInventory will have item added.
     * it is not taking precondition tag
     */
    //************************previous*********************//
//    public void addNewItem(LineItem item){
//        sellerInventory.addItem(item, userName);
//    }
    //*****************************************************//
    //ADDED

    public void addNewItem(LineItem item, int bundleID)
    {

        sellerInventory.addItem(item, userName, bundleID);

        //INSERT INTO PRODUCT(PRODUCT_ID, PRODUCT_DESC, QUANTITY, INV_PRICE, SELL_PRICE,
CATEGORY, USERID, BUNDLE_ID, DISCOUNT)
        // VALUES(2, 'dsfs' , 2, 3, 4, 's', 2, 0, 0)
    }

    //ADDED BUNDLE

    public void addNewBundle(Bundle inputBundle)
    {
        sellerInventory.addBundle(inputBundle);
    }

    /**
     * Intakes a int itemNumber to remove from inventory. PreCon: Item with
     * matching Item.ID in inventory. PostCon: Item removed from inventory
     *
     * @param itemNumber integer the item number
     */
    public void deleteExistingItem(int itemNumber)

    {
        sellerInventory.deleteItem(itemNumber);
    }
    //Miles 4/19

    /**
     * Intakes a LineItem item, and a bundleID, in order to update item. PreCon:
     * Item with matching Item.ID not updated. PostCon: Item updated
     *
     * @param item LineItem
     * @param bundleID integer the bundle id
     */
```

```
public void updateItem(LineItem item, int bundleID)
{
   System.out.println("at seller");
   sellerInventory.updateItem(item, userName, bundleID);
}
//Miles 4/19
public void calculateRevenue(){

}
/**
 * Gets the user name of seller
 * @return String the seller username.
 */
public  String getUserName(){
   return super.userName;
}
/**
 * Gets the seller password.
 * @return String the password.
 */
public  String getPassword(){
   return super.password;
}
/**
 * Gets the seller revenue.
 * @return double the revenue.
 */
public  double getRevenue(){
   return revenue;
}
/**
 * Gets name of seller
 * @return string the name
 */
public  String getName(){
   return super.name;
}
/**
 * Gets the address of seller
 * @return String the seller address.
 */
public  String getAddress(){
   return super.address;
}
/**
 * Gets city of seller
 * @return the city
 */
public  String getCity(){
   return super.city;
}
/**
 * Gets state of seller
 * @return the state
 */
public  String getState(){
   return super.state;
}
/**
 * Gets zip code of seller
 * @return the zip code.
 */
```

```
public  String getZip(){
   return super.zipcode;
}
/**
 * Gets phone number of seller.
 * @return the phone number.
 */
public  String getPhone(){
   return super.phone;
}
/**
 * Gets email of seller.
 * @return the email id.
 */
public  String getEmail(){
   return super.email;
}
/**
 * Gets the userid of seller.
 * @return the user id.
 */
public int getUserID(){
   return super.userID;
}
/**
 * Gets user type of seller
 * @return String This returns S as the buyer type.
 */
public  String getUserType(){
   return super.userType;
}
/**
 * sets user name of the seller
 * @param uNm User name of seller is set with uNm.
 */
public  void setUserName(String uNm)
{
   super.userName = uNm;
}
/**
 * sets password of the seller
 * @param PWD password of seller is set with PWD.
 */
public  void setPassword(String PWD){
   super.password = PWD;
}
/**
 * sets name of the seller
 * @param name the name of seller is set with name.
 */
public  void setName(String name){
   super.name = name;
}
/**
 * sets address of the seller
 * @param address the address of seller is set with address
 */
public  void setAddress(String address){
   super.address = address;
}
/**
 * sets city of the seller
```

```
     * @param city the city of seller is set with city.
     */
    public  void setCity(String city){
        super.city = city;
    }
    /**
     * sets state of the seller
     * @param state The state of seller is set with state.
     */
    public  void setState(String state){
        super.state = state;
    }
    /**
     * sets zip code of the seller
     * @param zipCode the zip code of seller is set with zip code.
     */
    public  void setZip(String zipCode){
        super.zipcode = zipCode;
    }
    /**
     * sets phone of the seller
     * @param phone the phone of seller is set with phone.
     */
    public  void setPhone(String phone){
        super.phone = phone;
    }
    /**
     * sets email of the seller
     * @param email the email of seller is set with email.
     */
    public  void setEmail(String email){
        super.email = email;
    }
    /**
     * sets userID of the seller
     * @param UID the user id of seller is set with user id.
     */
    public  void setUserID(int UID){
        super.userID = UID;
    }
    /**
     * sets user type of the seller as Seller
     */
    public  void setUserType(){
        super.userType ="Seller";

    }
    /**
     * sets name of the seller
     * @param r double the revenue
     */
    public  void setRevenue(double r){
        this.revenue = r;
    }
}
```

//cop4331\prj\src\cop4331\model\ProductIterator.java
//created an inner class in Inventory class instead of this Please ignore this file.

```
package cop4331.model;
import cop4331.gui.*;
///*
// * To change this license header, choose License Headers in Project Properties.
// * To change this template file, choose Tools | Templates
// * and open the template in the editor.
// */
//package shoppingcartproject;
//
///**
// *
// * @author Manisha
// */
//import java.util.ArrayList;
//import java.util.*;
//
//public abstract class ProductIterator implements Iterator<LineItem> {
//
//  private ArrayList<LineItem> items;
//  private int position;
//
//  public ProductIterator(LineItem itemList) {
//      //this.items = itemList.getItem();
//  }
//
//  @Override
//  public boolean hasNext() {
//      if (position < items.size())
//          return true;
//      else
//          return false;
//  }
//
//  @Override
//  public LineItem next() {
//      LineItem aniObj = items.get(position);
//      position++;
//      return aniObj;
//  }
//
//  @Override
//  public void remove() {
//      //items.remove(position);
//           throw new UnsupportedOperationException();
//  }
//
//}
//
//
//
```

//cop4331\prj\src\cop4331\model\Product.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;
import cop4331.gui.*;
import java.util.ArrayList;

/**
 * A class that holds Product information.Product is a Line Item.
 * @author Manisha
 */

public class Product implements LineItem
{
    private final int productId;
   private  String description;
   private double price;
   private int availableQty;
   private double invoicePrice;
   private String category;
    private final int sellerId;
   /**
    * Constructor that constructs a product.
    * @param prodId The product ID
    * @param description The product description.
    * @param price The product price.
    * @param availableQty The stock of this product.
    * @param invoicePrice The invoice price for this product.
    * @param category The category of this product.
    * @param sellerId The Id of the seller who is currently logged into the system and sells this product.
    */
   public Product(int prodId, String description, double price,   int availableQty, double invoicePrice, String category, int sellerId)
   {
      this.productId = prodId;
      this.description = description;
      this.price = price;
      this.availableQty = availableQty;
      this.category = category;
      this.invoicePrice = invoicePrice;
      this.sellerId = sellerId;
   }
   /**
    * Gets the product price.
    * @return the product price.
    */
    @Override
   public double getPrice() { return price; }
   /**
    * Returns the product information.
    * @return the string description of this product.
    */
    @Override
   public String toString() {
       String desc = description +" "+ Double.toString(price) +" "+Integer.toString(availableQty)+ "
"+Double.toString(invoicePrice) + " " + category;
       return desc;
   }
   /**
    * Gets the product available quantity.
```

```
     * @return the product available quantity.
     */
     @Override
    public int getAvailableQty() { return availableQty; }

    /**
     * Gets the product Invoice price.
     * @return the product Invoice price.
     */
     @Override
    public double getInvoicePrice() { return invoicePrice; }

    /**
     * Gets the product description.
     * @return the product description.
     */
     @Override
    public String getDescription(){
        return description;
    }
    /**
     * Gets the product category.
     * @return the product category.
     */
     @Override
    public String getCategory(){
        return category;
    }
    /**
     * Gets the product productId.
     * @return the product productId.
     */
     @Override
    public int getProductId()
    {
        return productId;
    }
    /**
     * Gets the product SellerId who is selling this product.
     * @return the product SellerId who is selling this product.
     */
     @Override
    public int getSellerId()
    {
        return sellerId;
    }

    /**
         * This method is used to get queries to update inventory after checkout.
         * @param queries This is the first parameter that has queries in string ArrayList.
         * @param orderedQty This is the second integer parameter that has order quantity.
         */
     @Override
     public void getUpdateInventoryQueries( ArrayList<String> queries, int orderedQty ){
         String query1 = "UPDATE PRODUCT SET QUANTITY = ";
         String query2 = " WHERE PRODUCT_ID = ";
         int finalQty = getAvailableQty() - orderedQty;
         queries.add(query1 + finalQty + query2 + Integer.toString(getProductId()));
     }


}
```

//cop4331\prj\src\cop4331\model\Order.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.model;
import cop4331.gui.*;
import java.util.ArrayList;

/**
*
* @author Manisha
*/
public class Order
{
   ArrayList<LineItem> itemsBoughtList;
   public  Order(int sc,ArrayList<LineItem> orderItem ){
   sc =0;
   itemsBoughtList = orderItem;
   }

}
```

//cop4331\prj\src\cop4331\model\LineItem.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;
import cop4331.gui.*;
import java.util.ArrayList;



/**
 * This is a Interface.
 * @author Manisha
 */
/**
   A line item in an invoice.
*/
public interface LineItem
{
   /**
      Gets the price of this line item.
      @return double the price
   */
   double getPrice();
   /**
      Gets the description of this line item.
      @return String the description
   */
   String toString();
   /**
    * Gets the description of this line item.
    * @return the description of line item
    */
   public String getDescription();
   /**
    * Gets the Available Quantity/ stock of this line item.
    * @return the available quantity of line item
    */
   public int getAvailableQty();
   /**
    * Gets the invoice price of this line item.
    * @return double the invoice price of the Line Item
    */
   public double getInvoicePrice();
   /**
    * Gets the category of this line item.
    * @return String the category of Line Item
    */
   public String getCategory();
   /**
    * Gets the queries needed to update inventory of this line item after  checkout.
    * @param queries This hold items to buy list
    * @param orderedQty This has order quantity of the line item.
    */
   public void getUpdateInventoryQueries( ArrayList<String> queries, int orderedQty );
   /**
    * Gets the product ID of Line item.
    * @return integer The product ID.
    */
   public int getProductId();
```

```
  /**
   * Gets the seller ID who sells this LineItem.
   * @return integer The seller ID.
   */
  public int getSellerId();

}
```

//cop4331\prj\src\cop4331\model\Inventory.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.model;
import cop4331.gui.*;
import java.util.*;
import static cop4331.model.ShoppingCartSystem.query;

/**
* A class that holds a sellers list of Line items.
* This class implements iterable LineItem interface.
* we get iterator() method needed for iterator design pattern to work.
* @author Manisha
*/
public class Inventory implements Iterable<LineItem>
{
   Random rn = new Random();//ADDED
   private ArrayList<LineItem> itemsInInventory;
   /**
    * This is a constructor to create inventory object.
    * @param username The user name.
    */
   public Inventory(String username)
   {
      itemsInInventory = ShoppingCartSystem.getInstance().retrieveSellerInventory( username);
   }

//    public void addItem(LineItem item, String userName, int bundleID){
//
//
//    }
   //This method ADDED by Miles
   //ADDED
/**this method adds products to the
 * @author Miles Robson //This method ADDED by Miles
 * @param item LineItem in the inventory
 * @param username String the username of seller
 * @param bundleID integer the bundle id of bundle
 */
   public void addItem(LineItem item, String username, int bundleID)
   {

      itemsInInventory.add(item);
      query = "INSERT INTO PRODUCT (PRODUCT_ID, PRODUCT_DESC, QUANTITY, INV_PRICE, SELL_PRICE,
CATEGORY, USERID, BUNDLE_ID, DISCOUNT)"
            + "VALUES (" + item.getProductId() + ",'" + item.getDescription().trim() + "'," + item.getAvailableQty() + "," +
item.getInvoicePrice() + "," + item.getPrice()
            + ",'" + item.getCategory() + "'," + item.getSellerId() + "," + bundleID + ",0)";

      ShoppingCartSystem.getInstance().addItem(query);

      Bundle ourBundle = new Bundle();
      ourBundle.setBundleID(rn.nextInt(Integer.MAX_VALUE) + 1);
      Product ourProduct = new Product(rn.nextInt(Integer.MAX_VALUE) + 1, "Test", 5, 5, 5, "Q", 1848099529);
      Product ourProduct2 = new Product(rn.nextInt(Integer.MAX_VALUE) + 1, "Test2", 3, 3, 3, "Q", 1848099529);
      ourBundle.add(ourProduct);
      ourBundle.add(ourProduct2);
      /*itemsInInventory.add(ourBundle);
       for (int i = 0; i < ourBundle.sizeOfBundle(); i++) {
```

```
        {

        LineItem inputItem = ourBundle.getItem(i);
        //itemsInInventory.add(inputItem);
        query = "INSERT INTO PRODUCT (PRODUCT_ID, PRODUCT_DESC, QUANTITY, INV_PRICE, SELL_PRICE,
CATEGORY, USERID, BUNDLE_ID, DISCOUNT)"
        + "VALUES ('" + inputItem.getProductId() + "','" + inputItem.getDescription().trim() + "'," + inputItem.getAvailableQty() +
"," + inputItem.getInvoicePrice() + "," + inputItem.getPrice()
        + ",'" + inputItem.getCategory() + "',"
        + inputItem.getSellerId() + "," + ourBundle.getBundleID() + ",0)";
        ShoppingCartSystem.getInstance().addItem(query);

        }

        }*/

    }

    //ADDED

    public void addBundle(Bundle inputBundle)
    {
        itemsInInventory.add(inputBundle);
        for (int i = 0; i < inputBundle.sizeOfBundle(); i++) {
            {

                LineItem inputItem = inputBundle.getItem(i);
                //itemsInInventory.add(inputItem);
                query = "INSERT INTO PRODUCT (PRODUCT_ID, PRODUCT_DESC, QUANTITY, INV_PRICE, SELL_PRICE,
CATEGORY, USERID, BUNDLE_ID, DISCOUNT)"
                        + "VALUES ('" + inputItem.getProductId() + "','" + inputItem.getDescription().trim() + "'," +
inputItem.getAvailableQty() + "," + inputItem.getInvoicePrice() + "," + inputItem.getPrice()
                        + ",'" + inputItem.getCategory() + "',"
                        + inputItem.getSellerId() + "," + inputBundle.getBundleID() + ",0)";
                ShoppingCartSystem.getInstance().addItem(query);
            }
        }
    }

     /**
     * updateItem Updates an item in this inventory, and updates the item in the
     * ShoppingCartSystem Pre:Item not updated Post: Item updated
     *
     * @param item LineItem in the inventory
     * @param username String the username of seller
     * @param bundleID integer the bundle id of bundle
     */
    public void updateItem(LineItem item, String username, int bundleID)
    {

        //itemsInInventory.add(item);
        query = "UPDATE PRODUCT "
                + "SET PRODUCT_DESC='" + item.getDescription().trim()
                + "', QUANTITY =" + item.getAvailableQty()
                + ", INV_PRICE =" + item.getInvoicePrice()
                + ",SELL_PRICE =" + item.getPrice()
                + ",CATEGORY = '" + item.getCategory()
                + "', BUNDLE_ID =" + bundleID
                + "WHERE PRODUCT_ID = " + item.getProductId();

        ShoppingCartSystem.getInstance().updateItem(query);
```

```
    }
    //Miles 4/19

    /**
     * deleteItem Updates an item in this inventory, and updates the item in the
     * ShoppingCartSystem Pre:Item not delete size = x Post: Item delete size =
     * x -1
     *
     * @param itemNumber integer the item number of item to be deleted
     */
    public void deleteItem(int itemNumber)
    {
        query = "DELETE FROM PRODUCT "
                + "WHERE PRODUCT_ID =" + itemNumber;

        ShoppingCartSystem.getInstance().deleteItem(query);
    }
    //Miles 4/19


//    public void updateItem(LineItem item){
//
//    }
//    public void deleteItem(LineItem item){
//    }
//    public double calculateRevenue(){
//        return 0.0;
//    }
//    public void displayRevenueDtls(){
//
//        }
    /**
     * Gets product count in itemsInInventory list
     * @return integer the size
     */
    public int getProductCount()
    {
        return itemsInInventory.size();
    }
    /**
     * Returns an iterator over elements of type LineItem.
     * @return an iterator over elements of type LineItem.
     */
    public Iterator<LineItem> iterator()
    {
        return new ProductIterator();
    }
    /**
     * inner class that helps implement the iterator design pattern by providing required methods hasNext(), next(), remove().
     */
    private class ProductIterator implements Iterator<LineItem>
    {
        private int position;
        /**
         * constructor that sets position of iterator.
         */
        public ProductIterator()
        {
            position = 0;
        }
        /**
         * This method returns true if iteration has more elements
```

```
     * @return boolean true if the iteration has more elements false otherwise.
     */
    @Override
    public boolean hasNext()
    {
        if (position < itemsInInventory.size())
            return true;
        else
            return false;
    }
    /**
     * Returns the next element in the iteration.
     * @return The next element in the iteration.
     */
    @Override
    public LineItem next()
    {
        LineItem aniObj = itemsInInventory.get(position);
        position++;
        return aniObj;
    }
    /**
     * Removes from the underlying collection the last element returned by this iterator.
     */
    @Override
    public void remove()
    {
        itemsInInventory.remove(position);
    // throw new UnsupportedOperationException();
    }

  }


}
```

//cop4331\prj\src\cop4331\model\DiscountedItem.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.model;

import java.util.ArrayList;
import cop4331.gui.*;
/**
*
* @author Manisha
*/

/**
A decorator for an item that applies a discount.
*/
public class DiscountedItem implements LineItem
{
   /**
   Constructs a discounted item.
   @param item the item to be discounted
   @param discount the discount percentage
   */
   public DiscountedItem(LineItem item, double discount)
   {
      this.item = item;
      this.discount = discount;
   }
   /**
    * This method returns discounted item price.
    * @return double the discounted price.
    */
   public double getPrice()
   {
      return item.getPrice() * (1 - discount / 100);
   }
   /**
    * This method returns string representation of the discounted item.
    * @return String the string representation of the discounted item.
    */
   public String toString()
   {
      return item.toString() + " (Discount " + discount
      + "%)";
   }
   /**
    * This method returns the description of the discounted item.
    * @return String The description of the discounted item.
    */

   public String getDescription(){
      return "Discounted (" + getDiscount() + " %): " + item.getDescription().trim();
   }

   /**
    * This method returns the available of the discounted item.
    * @return integer the available quantity
    */

   public int getAvailableQty(){
```

```
        return item.getAvailableQty();

    }
    /**
     * This method helps in getting the queries required to update the inventory/stock of the discounted item.
     * This method is used by shopping cart to create update queries.
     * @param queries The ArrayList that holds update query
     * @param orderedQty This holds order quantity.
     */
    public void getUpdateInventoryQueries( ArrayList<String> queries, int orderedQty ){
      item.getUpdateInventoryQueries(queries, orderedQty );
    }
    /**
     * This method returns the product ID of the discounted item.
     * @return int the product ID.
     */
    public int getProductId()
    {
      return item.getProductId();
    }
    /**
     * This method returns the sellerID of the discounted item.
     * @return int the seller ID.
     */
    public int getSellerId()
    {
      return item.getSellerId();
    }
    /**
     * This method returns the Invoice Price of the discounted item.
     * @return double The invoice price of discounted item.
     */
    public double getInvoicePrice(){
        return item.getInvoicePrice();

    }
    /**
     * This method returns the category of the discounted item.
     * @return String The category of discounted Item
     */
    public String getCategory(){
        return item.getCategory();

    }
    /**
     * This method returns the Discount of the discounted item.
     * @return double The discount.
     */
    public double getDiscount(){
        return discount;

    }

    private LineItem item;
    private double discount;
}
```

//cop4331\prj\src\cop4331\model\Buyer.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.model;
import cop4331.gui.*;

import java.util.Date;
import static cop4331.model.ShoppingCartSystem.query;

/**
* A class that holds buyer information and it has
* methods to access the buyer information.
* @author Manisha
*/
public class Buyer extends UsersofSystem
{
   private String creditCardNumber;
   private String creditCardHolder;
   private String cardType;
   private String validTill;
   private String cvvCode;
   private ShoppingCart shoppingCart;
   /**
    * Empty Constructor
    */
   public Buyer()
   {

   }
   /**
    * constructor to create  Buyer object.
    * @param usernm This is the first parameter to the constructor.
    * @param password This is the second parameter to the constructor.
    */
   public Buyer(String usernm, String password)
   {

      query = "select * from USERS where USERNAME ='" + usernm + "' and PASSWORD = '" + password + "'";
      ShoppingCartSystem.getInstance().retrieveUser(this, query);
      shoppingCart = new ShoppingCart();
   }
   /**
    *
    */
   public void updateProfile(){

   }

   /**
    * This method gets buyer's credit card number
    * @return String The credit card number.
    */
   public  String getcreditCardNumber(){
      return creditCardNumber;
   }
   /**
    * This method gets buyer's card type.
    * @return String The card type.
    */
```

```
public  String getcardType(){
   return cardType;
}
/**
 * This method returns credit card holder name.
 * @return String the credit card number.
 */
public  String getcreditCardHolder(){
   return creditCardHolder;
}
/**
 * This method returns valid date.
 * @return String The valid date.
 */
public  String getvalidTill(){
   return validTill;
}
/**
 * This method gets cvvCode.
 * @return String the cvvCode.
 */
public  String getcvvCode(){
   return cvvCode;
}
/**
 * This method sets credit card number
 * @param cNo The credit card number.
 */
public  void setcreditCardNumber(String cNo)
{
   creditCardNumber = cNo;
}
/**
 * This method sets the credit card type.
 * @param cType The credit card Type.
 */
public  void setcardType(String cType){
   cardType = cType;
}
/**
 * This method sets credit card holder name.
 * @param name The card holder name.
 */
public  void setcreditCardHolder(String name){
   creditCardHolder = name;
}
/**
 * This method sets credit card's valid date.
 * @param validDt The valid date.
 */
public  void setvalidTill(String validDt){
   validTill = validDt;
}
/**
 * This method sets cvv code.
 * @param cvv The cvv code.
 */
public  void setcvvCode(String cvv){
   cvvCode = cvv;
}
/**
 * This method helps saving buyer's information into the database.
```

```
 *
 */
public final void saveBuyerCreditcardInfo(){
   query = "INSERT INTO BUYER (USERID, CCNO, CCHOLDER, CCTYPE, VALIDDATE,CVV) "
   + "VALUES (" + getUserID() + ",'" +   getcreditCardNumber() + "','" +   getcreditCardHolder()
   + "','" +   getcardType()+ "','" +   getvalidTill()+ "','" +   getcvvCode()
   +"')";

   ShoppingCartSystem.getInstance().addUpdateSellerOrBuyer(query);
}
/**
 * This method returns the buyer username.
 * @return String the username.
 */
public  String getUserName(){
   return super.userName;
}
/**
 * This method returns the password.
 * @return String the password.
 */
public  String getPassword(){
   return super.password;
}
/**
 * This method returns the name of the buyer.
 * @return String This returns name of the buyer.
 */
public  String getName(){
   return super.name;
}
/**
 * This method returns the address of the buyer.
 * @return String This returns address of the buyer.
 */
public  String getAddress(){
   return super.address;
}
/**
 * This method returns the city of the buyer.
 * @return String This returns city of the buyer.
 */
public  String getCity(){
   return super.city;
}
/**
 * This method returns the state of the buyer.
 * @return String This returns state of the buyer.
 */
public  String getState(){
return super.state;
}
/**
 * This method returns the zip code of the buyer.
 * @return String This returns zip code of the buyer.
 */
public  String getZip(){
   return super.zipcode;
}
/**
 * This method returns the phone of the buyer.
 * @return String This returns phone of the buyer.
```

```
     */
    public  String getPhone(){
        return super.phone;
    }
    /**
     * This method returns the email of the buyer.
     * @return String This returns email of the buyer.
     */
    public  String getEmail(){
        return super.email;
    }
    /**
     * This method returns the user Id of the buyer.
     * @return String This returns user ID of the buyer.
     */
    public int getUserID(){
        return super.userID;
    }
    /**
     * This method returns the Type of the buyer.
     * @return String This returns B as the buyer type.
     */
    public  String getUserType(){
        return super.userType;
    }
    /**
     * This method returns the name of the buyer.
     * @return String This returns name of the buyer.
     */
//    public  int getSellerID(){
//        return super.userID;
//    }
    /**
     * This method sets the user name of the buyer.
     * @param uNm The username.
     */
    public  void setUserName(String uNm)
    {
        super.userName = uNm;
    }
    /**
     * This method sets the password of the buyer.
     * @param PWD The password.
     */
    public  void setPassword(String PWD){
        super.password = PWD;
    }
    /**
     * This method sets the name of the buyer.
     * @param name The name of the buyer.
     */
    public  void setName(String name){
        super.name = name;
    }
    /**
     * This method sets the name of the buyer.
     * @param address the address of the buyer
     */
    public  void setAddress(String address){
        super.address = address;
    }
    /**
```

```
 * This method sets the name of the buyer.
 * @param city the city of buyer
 */
public  void setCity(String city){
   super.city = city;
}
/**
 * This method sets the state of the buyer.
 * @param state String the state that buyer lives
 */
public  void setState(String state){
   super.state = state;
}
/**
 * This method sets the zip code of the buyer.
 * @param zipCode the zip code.
 */
public  void setZip(String zipCode){
   super.zipcode = zipCode;
}
/**
 * This method sets the phone of the buyer.
 * @param phone the phone number.
 */
public  void setPhone(String phone){
   super.phone = phone;
}
/**
 * This method sets the email of the buyer.
 * @param email The email of buyer.
 */
public  void setEmail(String email){
   super.email = email;
}
/**
 * This method sets the userID of the buyer.
 * @param UID The user id of buyer.
 */
public  void setUserID(int UID){
   super.userID = UID;

}
/**
 * This method sets the name of the buyer.
 */
public  void setUserType(){
   super.userType ="B";

}
/**
 * This method is used to add items to buyer's shopping cart.
 * @param item The product to be added.
 * @param quantity The quantity of product to be added.
 */
public void addToShoppingCart(LineItem item, int quantity)
{
   shoppingCart.addItemToCart(item, quantity);
}
/**
 * This method returns the shopping cart of the buyer.
 * @return shoppingCart The ShoppingCart object.
 */
```

```
    public ShoppingCart getShoppingCart()
    {
        return shoppingCart;
    }
    /**
     * This method helps buyer update quantity added.
     * @param index The index of the item in the items to buy list in shopping cart.
     * @param quantity The quantity corresponding to index of item in the list in shopping cart.
     */
    public void updateQuantityAtIndex(int index, int quantity)
    {
        shoppingCart.updateQuantityAtIndex( index,  quantity);
    }
}
```

//cop4331\prj\src\cop4331\model\Bundle.java

```
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package cop4331.model;


/**
*
* @author Manisha
*/
import java.util.*;
import cop4331.gui.*;
/**
   A bundle of line items that is again a line item.
*
*/
public class Bundle implements LineItem
{
   /**
   Constructs a bundle with no items.
   */
   public Bundle() { items = new ArrayList<LineItem>(); }

   /**
   Adds an item to the bundle.
   @param item the item to add
   */
   public void add(LineItem item) { items.add(item); }

   /**
   * returns price of the LineItem
   * @return double the price
   */

   public double getPrice()
   {
      double price = 0;

      for (LineItem item : items)
       price += item.getPrice();
      return price;
   }
   /**
   * gets the description of all items in bundle
   * @return string this item's description
   */
   public String toString()
   {
      String description = "Bundle: ";
      for (int i = 0; i < items.size(); i++)
      {
         if (i > 0) description += ", ";
            description += items.get(i).toString();

   }
   return description;
   }
   /**
   * This method is used to get the product ID
```

```
* @return integer the product id
*/
public int getProductId()
{
   return -1;
}
/**
* This method returns the seller ID who sells this item
* @return seller ID an integer value
*/
public int getSellerId()
{
   return items.get(0).getSellerId();
}


/**
* This method is used to get description of bundle as a product
* @return string the description of all items in bundle
* precondition item.size greater than 0
*/
public String getDescription()
{
   String description = "Bundle: ";
   for (int i = 0; i < items.size(); i++)
   {
      if (i > 0) description += ", ";
      description += items.get(i).getDescription().trim();
   }
   return description;
}
/**
* This method is used to get queries to update inventory after checkout.
* @param queries This is the first parameter that has queries in string ArrayList.
* @param orderedQty This is the second integer parameter that has order quantity.
*/
public void getUpdateInventoryQueries( ArrayList<String> queries, int orderedQty )
{

   for (int i = 0; i < items.size(); i++)
   {
     items.get(i).getUpdateInventoryQueries(queries, orderedQty );
   }
}
/**
* The method to get available stock of items in a bundle.
* @return integer This returns available Quantity /Stock
*/
public int getAvailableQty()
{
   int minAvailable = Integer.MAX_VALUE;
      for (int i = 0; i < items.size(); i++)
      {
        if(minAvailable > items.get(i).getAvailableQty())
           minAvailable = items.get(i).getAvailableQty();

      }
   return minAvailable;
}
/**
* This method returns the invoice price of a bundle.
* @return double the invoice price of a bundle.
```

```
        */
        public double getInvoicePrice(){
            double invPrice = 0;
            for (int i = 0; i < items.size(); i++)
            {

                invPrice  = invPrice + items.get(i).getInvoicePrice();

            }
            return invPrice;
        }
        /**
        * This method returns the category of the bundle.
        * @return string that represents category.
        */
        public String getCategory()
        {
            String Category = "Category: ";
            for (int i = 0; i < items.size(); i++)
            {
                if (i > 0) Category += ", ";
                Category += items.get(i).getCategory();
            }
            return Category;
        }
        //ADDED
        /**
         * The method to get BundleID
         * @return int the BundleID
         */
        public int getBundleID()
        {
            return bundleID;
        }
    //ADDED
    /**
     * A method to set the bundle ID.
     * @param inputBundleID int the bundle Id of a Bundle.
     */
        public void setBundleID(int inputBundleID)
        {
            bundleID = inputBundleID;
        }
    //ADDED
    /**
     * Gets the LineItem at given position
     * @param inputItem the index / position.
     * @return LineItem at given index
     * precondition items.size() is greater than 0
     */
        //@precondition items.size() is greater than 0
        public LineItem getItem(int inputItem)
        {
            return items.get(inputItem);
        }
    //ADDED
    /**
     * Gets no of items in a bundle
     * @return int the count of items in a bundle
     *
     */
        public int sizeOfBundle()
```

```
    {
        return items.size();
    }
//ADDED
    private int bundleID;
        private ArrayList<LineItem> items;//list of items in a bundle.
    }
```

//cop4331\prj\src\cop4331\model\UsersofSystem.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;
import cop4331.gui.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import static cop4331.model.ShoppingCartSystem.query;

/**
 * A model class that holds user profile info.
 * It has template method registerNewUser to register new seller or new buyer.
 * @author Manisha
 */
public abstract class UsersofSystem {
    protected String userName;
    protected String password;
    protected String name;
    protected String address;
    protected String city;
    protected String state;
    protected String zipcode;
    protected String phone;
    protected String email;
    protected int userID;
    protected String userType;

    /**
     * This is a template method used to register seller or buyer
     * @param s The user type "S" for seller or "B" for buyer.
     */
     public final void registerNewUser(String s){
          query = "INSERT INTO USERS (USERID, NAME, USERNAME, PASSWORD, ADDRESS,CITY, ZIPCODE,
STATE, PHONE, EMAIL, USERTYPE) "
                + "VALUES (" + getUserID() + ",'" +   getName() + "','" +   getUserName()
                + "','" +   getPassword() + "','" +   getAddress()+ "','" +   getCity()
                + "','" +   getZip()+ "','" +   getState()+ "','" +   getPhone()
                + "','" +   getEmail()+"','" + s +"')";

              ShoppingCartSystem.getInstance().addUpdateSellerOrBuyer(query);

          query = "INSERT INTO SELLER (USERID, PROFIT, TOTAL_INV_PRICE, TOTAL_SELL_PRICE) VALUES (" +
getUserID() + ", 0 ,0, 0)";
          ShoppingCartSystem.getInstance().addUpdateSellerOrBuyer(query);

    }

    //*****************************************************************************//
//    public final void loginVerifiCation(String uNm, String PWD){
//
//        if(login(uNm, PWD))
//            System.out.println("Login Verified");
//        else
//            System.out.println("Login Failed!!");
//
//
//    }
```

```
//
//    public  boolean login(String uNm, String PWD){
//       if((this.userName).equals(uNm) && (this.password).equals(PWD))
//          return true;
//       else
//          return false;
//
//    }
    //*********************************************************//////////////////
    /**
     * This method updates uer profile info.
     * @param id the userid.
     */
    public final void updateUserProfile(int id){
////////////////        try {
////////////////         String url="jdbc:derby://localhost:1527/myDB";
////////////////           String username ="manisha";
////////////////           String password = "manisha";
////////////////           Connection con = DriverManager.getConnection(url,username,password);
//          // Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/myDB;user=manisha;pass=manisha");
//           Statement s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
//                                ResultSet.CONCUR_READ_ONLY);

        query = "Update USERS set NAME ='" +  getName() + "', set USERNAME ='" + getUserName() +"',set PASSWORD ='"
+ getPassword()+ "',set ADDRESS ='" + getCity() +"', set CITY ='"
              + getCity() +"', set ZIPCODE ='" + getZip() +"',set STATE ='" + getState()+ "',set PHONE ='" +  getPhone() +"', set
EMAIL='" + getEmail()+"' where USERID=" + id ;
           ShoppingCartSystem.getInstance().addUpdateSellerOrBuyer(query);
         //String insertNewUserSQL = "INSERT INTO USERS  VALUES (?, ?, ?, ?, ?, ?, ?)";
//////////////          PreparedStatement pstmt1 = con.prepareStatement(query);
          //rs1 = s.executeQuery(query);
////////////////          int count = pstmt1.executeUpdate();



////////////      } catch (SQLException ex) {
////////////
////////////        ex.printStackTrace();
////////////      }


   }


  /**
   * the method that displays string representation of user profile info.
   * @return String the formated user profilr info.
   */
    public final String toString() {
       String str = "User Name:" + getName() + "  Address =" + getAddress() + "  City =" + getCity() + "  City =" + getState()
                + "  Zip Code =" + getZip() + "  Phone =" + getPhone()+ "  Email =" + getEmail();
       System.out.println(str);
       return str;
    }

    //public abstract void updateUserProfile();

     public abstract String getUserName();
     public abstract String getPassword();
     public abstract String getName();
     public abstract String getAddress();
     public abstract String getCity();
```

```
    public abstract String getState();
    public abstract String getZip();
    public abstract String getPhone();
    public abstract String getEmail();
    public abstract int getUserID();
    public abstract String getUserType();



   public abstract void setUserName(String uNm);
    public abstract void setPassword(String PWD);
    public abstract void setName(String name);
    public abstract void setAddress(String Address);
    public abstract void setCity(String city);
    public abstract void setState(String state);
    public abstract void setZip(String zip);
    public abstract void setPhone(String phone);
    public abstract void setEmail(String email);
     public abstract void setUserID(int UID);
    public abstract void setUserType();
}
```

//Testing Code

//cop4331\prj\test\cop4331\model\ShoppingCartTest.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;

import java.util.ArrayList;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Manisha
 */
public class ShoppingCartTest {

    public ShoppingCartTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of addItemToCart method, of class ShoppingCart.
     */
    @Test
    public void testAddItemToCart() {
        System.out.println("addItemToCart");
        LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
        int quantity = 5;
        ShoppingCart instance = new ShoppingCart();
        instance.addItemToCart(item, quantity);
        // TODO review the generated test code and remove the default call to fail.
       // fail("The test case is a prototype.");
        item = new Product( 865576422, "glasses", 15,   133, 10, "K", 2033513407);
        quantity = 2;
        instance.addItemToCart(item, quantity);
    }

    /**
```

```
 * Test of updateQuantityAtIndex method, of class ShoppingCart.
 */
@Test
public void testUpdateQuantityAtIndex() {
    System.out.println("updateQuantityAtIndex");

    int index = 1;

    ShoppingCart instance = new ShoppingCart();
    LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
    int quantity = 5;

    instance.addItemToCart(item, quantity);
    // TODO review the generated test code and remove the default call to fail.
   // fail("The test case is a prototype.");
    item = new Product( 865576422, "glasses", 15,   133, 10, "K", 2033513407);
    quantity = 1;
    instance.addItemToCart(item, quantity);
     quantity = 2;
    instance.updateQuantityAtIndex(index, quantity);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of calculateUnitPrice method, of class ShoppingCart.
 */
@Test
public void testCalculateUnitPrice() {
    System.out.println("calculateUnitPrice");
    ShoppingCart instance = new ShoppingCart();
    ArrayList<Double> myResult = new ArrayList<Double>();
          myResult.add(Double.parseDouble("125.0"));
          myResult.add(Double.parseDouble("15.0"));
    ArrayList<Double> expResult = new ArrayList<Double>();
    for(int i=0; i<myResult.size(); i++)
    {
       expResult.add(i, myResult.get(i));
    }
     LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
    int quantity = 5;

    instance.addItemToCart(item, quantity);
    // TODO review the generated test code and remove the default call to fail.
   // fail("The test case is a prototype.");
    item = new Product( 865576422, "glasses", 15,   133, 10, "K", 2033513407);
    quantity = 1;
    instance.addItemToCart(item, quantity);
    ArrayList<Double> result = instance.calculateUnitPrice();

    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of calculateTotal method, of class ShoppingCart.
 */
@Test
public void testCalculateTotal() {
    System.out.println("calculateTotal");
    ShoppingCart instance = new ShoppingCart();
```

```
        LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
        int quantity = 5;

        instance.addItemToCart(item, quantity);
        // TODO review the generated test code and remove the default call to fail.
       // fail("The test case is a prototype.");
        item = new Product( 865576422, "glasses", 15,   133, 10, "K", 2033513407);
        quantity = 1;
        instance.addItemToCart(item, quantity);
        double expResult = 140.0;
        double result = instance.calculateTotal();
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of createOrder method, of class ShoppingCart.


     */
//    @Test
//    public void testCreateOrder() {
//        System.out.println("createOrder");
//        ShoppingCart instance = new ShoppingCart();
//        Order expResult = null;
//        Order result = instance.createOrder();
//        assertEquals(expResult, result);
//        // TODO review the generated test code and remove the default call to fail.
//        //fail("The test case is a prototype.");
//    }



    /**
     * Test of getItemsToBuy method, of class ShoppingCart.
     */
    @Test
    public void testGetItemsToBuy() {
        System.out.println("getItemsToBuy");
        ShoppingCart instance = new ShoppingCart();
        LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
        int quantity = 5;
         ArrayList<LineItem> expResult = new ArrayList<LineItem>();
        instance.addItemToCart(item, quantity);
       expResult.add(item);
        // TODO review the generated test code and remove the default call to fail.
       // fail("The test case is a prototype.");
        item = new Product( 865576422, "glasses", 15,   133, 10, "K", 2033513407);
        quantity = 1;
        instance.addItemToCart(item, quantity);

        expResult.add(item);

        ArrayList<LineItem> result = instance.getItemsToBuy();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of getQuantity method, of class ShoppingCart.
```

```
     */
    @Test
    public void testGetQuantity() {
        System.out.println("getQuantity");
        ShoppingCart instance = new ShoppingCart();
        ArrayList<Integer> expResult = null;
        LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
        int quantity = 5;
        expResult = new ArrayList<Integer>();
        instance.addItemToCart(item, quantity);
       expResult.add(quantity);
        // TODO review the generated test code and remove the default call to fail.
       // fail("The test case is a prototype.");
        item = new Product( 865576422, "glasses", 15,   133, 10, "K", 2033513407);
        quantity = 1;
        instance.addItemToCart(item, quantity);

        expResult.add(quantity);
        ArrayList<Integer> result = instance.getQuantity();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
      // fail("The test case is a prototype.");
    }

    /**
     * Test of getUpdateInventoryQueries method, of class ShoppingCart.
     */
    @Test
    public void testGetUpdateInventoryQueries() {
        System.out.println("getUpdateInventoryQueries");
        ShoppingCart instance = new ShoppingCart();
         ArrayList<String> expResult = null;

        LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
        int quantity = 5;
        expResult = new ArrayList<String>();
        expResult.add("UPDATE PRODUCT SET QUANTITY = 28 WHERE PRODUCT_ID = 865576421");
        instance.addItemToCart(item, quantity);


        ArrayList<String> result = instance.getUpdateInventoryQueries();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of getUpdateSellerRevenueQueries method, of class ShoppingCart.
     */
    @Test
    public void testGetUpdateSellerRevenueQueries() {
        System.out.println("getUpdateSellerRevenueQueries");
        ShoppingCart instance = new ShoppingCart();
        ArrayList<String> expResult = null;
        LineItem item = new Product( 865576421, "chair", 25,   33, 20, "L", 2033513407);
        int quantity = 5;
        expResult = new ArrayList<String>();

          expResult.add("UPDATE SELLER SET PROFIT = PROFIT + 25.0 WHERE USERID = 2033513407");

         expResult.add("UPDATE SELLER SET TOTAL_INV_PRICE = TOTAL_INV_PRICE + 100.0 WHERE USERID =
2033513407");
```

```
      expResult.add("UPDATE SELLER SET TOTAL_SELL_PRICE = TOTAL_SELL_PRICE + 125.0 WHERE USERID =
2033513407");
       for(int i =0; i <expResult.size(); i++)
      {
         System.out.println(expResult.get(i));
      }

      instance.addItemToCart(item, quantity);



      ArrayList<String> result = instance.getUpdateSellerRevenueQueries();
      ArrayList<String> result1 = new ArrayList<String>();
      for(int i =0; i <result.size(); i++)
      {
         result1.add(i,result.get(i));
      }

      assertEquals(expResult, result1);
      // TODO review the generated test code and remove the default call to fail.
      //fail("The test case is a prototype.");
   }

}
```

//cop4331\prj\test\cop4331\model\SellerIT.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Miles Robson
 */
public class SellerIT
{

    public SellerIT()
    {
    }

    @BeforeClass
    public static void setUpClass()
    {
    }

    @AfterClass
    public static void tearDownClass()
    {
    }

    @Before
    public void setUp()
    {
    }

    @After
    public void tearDown()
    {
    }

    /**
     * Test of getProductCount method, of class Seller.
     */
    @Test
    public void testGetProductCount()
    {
        System.out.println("getProductCount");
        Seller instance = new Seller("example", "example");

        int expResult = 0;
        int result = instance.getProductCount();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
```

```
    /**
     * Test of getInventory method, of class Seller.
     */
    @Test
    public void testGetInventory()
    {
        System.out.println("getInventory");
        Seller instance = new Seller("example", "example");
        Inventory expResult = instance.getInventory();
        Inventory result = instance.getInventory();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of addNewItem method, of class Seller.
     */
    @Test
    public void testAddNewItem()
    {
        System.out.println("addNewItem");
        LineItem item = null;
        int bundleID = 0;
        Seller instance = new Seller("example", "example");

        instance.addNewItem(new Product(2, "Test", 2, 3, 1, "Q", 5), bundleID);
        int exVal = 1;
        int result = instance.getProductCount();
        assertEquals(exVal, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of getRevenue method, of class Seller.
     */
    @Test
    public void testGetRevenue()
    {
        System.out.println("getRevenue");
        Seller instance = new Seller("test", "test");
        double expResult = 0.0;
        double result = instance.getRevenue();
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
//        fail("The test case is a prototype.");
    }

}
```

//cop4331\prj\test\cop4331\model\DiscountedItemIT.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;

import java.util.ArrayList;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Miles Robson
 */
public class DiscountedItemIT
{

    public DiscountedItemIT()
    {
    }

    @BeforeClass
    public static void setUpClass()
    {
    }

    @AfterClass
    public static void tearDownClass()
    {
    }

    @Before
    public void setUp()
    {
    }

    @After
    public void tearDown()
    {
    }

    /**
     * Test of getPrice method, of class DiscountedItem.
     */
    @Test
    public void testGetPrice()
    {
        System.out.println("getPrice");
        Product ourProduct = new Product(2, "Test", 10, 3, 1, "Q", 5);
        DiscountedItem instance = new DiscountedItem(ourProduct, 50);
        double expResult = 5;
        double result = instance.getPrice();
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }
```

```
@Test
public void testGetDiscount()
{
   System.out.println("getDiscount");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
   double expResult = 50;
   double result = instance.getDiscount();
   assertEquals(expResult, result, 0.0);
   // TODO review the generated test code and remove the default call to fail.
   //fail("The test case is a prototype.");
}

/**
 * Test of toString method, of class DiscountedItem.
 */
@Test
public void testToString()
{
   System.out.println("toString");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);

   String expResult = ourProduct.toString() + " (Discount " + instance.getDiscount() + "%)";
   String result = instance.toString();
   assertEquals(expResult, result);
   // TODO review the generated test code and remove the default call to fail.
   //fail("The test case is a prototype.");
}

/**
 * Test of getDescription method, of class DiscountedItem.
 */
@Test
public void testGetDescription()
{
   System.out.println("getDescription");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
   //"Discounted (" + getDiscount() + " %): " + item.getDescription().trim();
   String expResult = "Discounted (50.0 %): Test";
   String result = instance.getDescription();
   assertEquals(expResult, result);
   // TODO review the generated test code and remove the default call to fail.
   // fail("The test case is a prototype.");
}

/**
 * Test of getAvailableQty method, of class DiscountedItem.
 */
@Test
public void testGetAvailableQty()
{
   System.out.println("getAvailableQty");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
   int expResult = 3;
   int result = instance.getAvailableQty();
   assertEquals(expResult, result);
   // TODO review the generated test code and remove the default call to fail.
   //fail("The test case is a prototype.");
```

```
}

/**
 * Test of getUpdateInventoryQueries method, of class DiscountedItem.
 */
/**
 * Test of getProductId method, of class DiscountedItem.
 */
@Test
public void testGetProductId()
{
   System.out.println("getProductId");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
   int expResult = 2;
   int result = instance.getProductId();
   assertEquals(expResult, result);
   // TODO review the generated test code and remove the default call to fail.
   //fail("The test case is a prototype.");
}

/**
 * Test of getSellerId method, of class DiscountedItem.
 */
@Test
public void testGetSellerId()
{
   System.out.println("getSellerId");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
   int expResult = 5;
   int result = instance.getSellerId();
   assertEquals(expResult, result);
   // TODO review the generated test code and remove the default call to fail.
   //fail("The test case is a prototype.");
}

/**
 * Test of getInvoicePrice method, of class DiscountedItem.
 */
@Test
public void testGetInvoicePrice()
{
   System.out.println("getInvoicePrice");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
   double expResult = 1;
   double result = instance.getInvoicePrice();
   assertEquals(expResult, result, 0.0);
   // TODO review the generated test code and remove the default call to fail.
   //fail("The test case is a prototype.");
}

/**
 * Test of getCategory method, of class DiscountedItem.
 */
@Test
public void testGetCategory()
{
   System.out.println("getCategory");
   Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
   DiscountedItem instance = new DiscountedItem(ourProduct, 50);
```

```
        String expResult = "Q";
        String result = instance.getCategory();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of getDiscount method, of class DiscountedItem.
     */
}
```

```
//cop4331\prj\test\cop4331\model\InventoryIT.java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;

import java.util.Iterator;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Miles Robson
 */
public class InventoryIT
{

    public InventoryIT()
    {
    }

    @BeforeClass
    public static void setUpClass()
    {
    }

    @AfterClass
    public static void tearDownClass()
```

```
    {
    }

    @Before
    public void setUp()
    {
    }

    @After
    public void tearDown()
    {
    }

    /**
     * Test of addItem method, of class Inventory.
     */
    @Test
    public void testAddItem()
    {
        System.out.println("addItem");
        Product ourProduct = new Product(2, "Test", 2, 3, 1, "Q", 5);
        String username = "";
        int bundleID = 0;
        Inventory instance = new Inventory("Example");
        int expectedVaule = instance.getProductCount() + 1;
        instance.addItem(ourProduct, username, bundleID);
        int result = instance.getProductCount();
        assertEquals(expectedVaule, result);

        // TODO review the generated test code and remove the default call to fail.
//        fail("The test case is a prototype.");
    }

    /**
```

```
 * Test of addBundle method, of class Inventory.
 */
@Test
public void testAddBundle()
{
    System.out.println("addBundle");
    Bundle inputBundle = new Bundle();
    Product ourProduct = new Product(2, "Test", 2, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(3, "Test", 2, 3, 1, "Q", 5);
    inputBundle.add(ourProduct);
    inputBundle.add(ourProduct2);
    Inventory instance = new Inventory("Example");
    int exVal = instance.getProductCount() + 1;
    instance.addBundle(inputBundle);
    int result = instance.getProductCount();
    assertEquals(exVal, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of getProductCount method, of class Inventory.
 */
@Test
public void testGetProductCount()
{
    System.out.println("getProductCount");
    Inventory instance = new Inventory("Example");
    Product ourProduct = new Product(2, "Test", 2, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(3, "Test", 2, 3, 1, "Q", 5);
    instance.addItem(ourProduct, "Example", 3);
    instance.addItem(ourProduct, "Example", 2);
    int expResult = 2;
    int result = instance.getProductCount();
```

```
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```

//cop4331\prj\test\cop4331\model\ BundleIT.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cop4331.model;

import java.util.ArrayList;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Miles Robson
 */
public class BundleIT
{

    public BundleIT()
    {
    }

    @BeforeClass
    public static void setUpClass()
    {
    }
```

```
@AfterClass
public static void tearDownClass()
{
}

@Before
public void setUp()
{
}

@After
public void tearDown()
{
}

/**
 * Test of add method, of class Bundle.
 */
@Test
public void testAdd()
{
    System.out.println("add");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    int expValue = 1;
    int result = instance.sizeOfBundle();
    assertEquals(expValue, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of getPrice method, of class Bundle.
```

```
 */
@Test
public void testGetPrice()
{
    System.out.println("getPrice");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    double expResult = 40;
    double result = instance.getPrice();
    assertEquals(expResult, result, 0.0);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of toString method, of class Bundle.
 */
@Test
public void testToString()
{
    System.out.println("toString");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    String expResult = "Bundle: Test 20.0 3 1.0 Q, Test 20.0 7 1.0 Q";
    String result = instance.toString();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    // fail("The test case is a prototype.");
```

```
}

/**
 * Test of getSellerId method, of class Bundle.
 */
@Test
public void testGetSellerId()
{
    System.out.println("getSellerId");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    int expResult = 5;
    int result = instance.getSellerId();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    // fail("The test case is a prototype.");
}

/**
 * Test of getDescription method, of class Bundle.
 */
@Test
public void testGetDescription()
{
    System.out.println("getDescription");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    String expResult = "Bundle: Test, Test";
```

```
    String result = instance.getDescription();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of getAvailableQty method, of class Bundle.
 */
@Test
public void testGetAvailableQty()
{
    System.out.println("getAvailableQty");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    int expResult = 3;
    int result = instance.getAvailableQty();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    // fail("The test case is a prototype.");
}

/**
 * Test of getInvoicePrice method, of class Bundle.
 */
@Test
public void testGetInvoicePrice()
{
    System.out.println("getInvoicePrice");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
```

```
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    double expResult = 2.0;
    double result = instance.getInvoicePrice();
    assertEquals(expResult, result, 0.0);
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}

/**
 * Test of getCategory method, of class Bundle.
 */
@Test
public void testGetCategory()
{
    System.out.println("getCategory");
    Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
    Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
    Bundle instance = new Bundle();
    instance.add(ourProduct);
    instance.add(ourProduct2);
    String expResult = "Category: Q, Q";
    String result = instance.getCategory();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    // fail("The test case is a prototype.");
}

/**
 * Test of getBundleID method, of class Bundle.
 */
@Test
public void testGetBundleID()
```

```
    {
        System.out.println("getBundleID");
        Product ourProduct = new Product(2, "Test", 20, 3, 1, "Q", 5);
        Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
        Bundle instance = new Bundle();
        instance.add(ourProduct);
        instance.add(ourProduct2);
        instance.setBundleID(3);
        int expResult = 3;
        int result = instance.getBundleID();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of setBundleID method, of class Bundle.
     */
    @Test
    public void testSetBundleID()
    {
        System.out.println("setBundleID");
        int inputBundleID = 0;
        Bundle instance = new Bundle();
        instance.setBundleID(inputBundleID);
        int result = instance.getBundleID();
        assertEquals(inputBundleID, result);
// TODO review the generated test code and remove the default call to fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getItem method, of class Bundle.
     */
```

```
    @Test
    public void testGetItem()
    {
        System.out.println("getItem");
        int inputItem = 0;
        Bundle instance = new Bundle();
        Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
        instance.add(ourProduct2);
        LineItem expResult = ourProduct2;
        LineItem result = instance.getItem(inputItem);
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of sizeOfBundle method, of class Bundle.
     */
    @Test
    public void testSizeOfBundle()
    {
        System.out.println("sizeOfBundle");
        Bundle instance = new Bundle();
        Product ourProduct2 = new Product(2, "Test", 20, 7, 1, "Q", 5);
        instance.add(ourProduct2);
        int expResult = 1;
        int result = instance.sizeOfBundle();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        //fail("The test case is a prototype.");
    }

}
```