

JAVA Shopping Cart

By Rabih El Khatib, Gerrell Bones, Zachary Bundarin, Solan Degefa

Group 20

COP 4331-001

Final Delivery

Github Link: <https://github.com/Rabihkhatib1/Group-20-Shopping-Cart>

Youtube Link: <https://youtu.be/JpeAb-kvwmU>

Contents

Introduction	4
Use Cases	5
CRC Cards	9
Class Diagram	13
Design Patterns	14
Strategy Pattern	14
Facade Pattern	15
Composite Pattern	16
Decorator Pattern	17
Observer Pattern.....	18
Iterator Pattern	19
Singleton Pattern	20
Sequence Diagrams.....	21
State Diagrams	37
Source Code	38
File: src/cop4331/application/Main.java.....	38
File: src/cop4331/application/User.java	39
File: src/cop4331/application/Inventory.java	43
File: src/cop4331/application/Profit.java	53
File: src/cop4331/model/LinItem.java	57
File: src/cop4331/model/Product.java	59
File: src/cop4331/model/DiscountedItem.java	61
File: src/cop4331/model/Bundle.java	63
File: src/cop4331/model/Invoice.java	66
File: src/cop4331/model/SellerBundle.java	69
File: src/cop4331/controller/UserController.java	71
File: src/cop4331/controller/CustomerController.java	74
File: src/cop4331/controller/SellerController.java.....	76
File: src/cop4331/controller/CardController.java	82
File: src/cop4331/gui/LoginView.java	85
File: src/cop4331/gui/SignupView.java	87
File: src/cop4331/gui/CustomerView.java	89

File: src/cop4331/gui/CardView.java	92
File: src/cop4331/gui/SellerView.java	95
File: src/cop4331/gui/SellerNewView.java.....	98
File: src/cop4331/gui/SellerBundleView.java.....	101
File: test/cop4331/gui/ProfitView.java	104
File: src/cop4331/formatter/InvoiceFormatter.java	106
File: src/cop4331/formatter/SimpleFormatter.java.....	107
File: src/cop4331/formatter/ProfitFormatter.java.....	108
File: src/cop4331/formatter/SellerBundleFormatter.java	109
File: test/cop4331/application/ProfitTest.java.....	110
File: test/cop4331/application/UserTest.java	112
File: test/cop4331/controller/CardControllerTest.java.....	113
File: test/cop4331/controller/SellerControllerTest.java	115
File: test/cop4331/model/BundleTest.java	117
File: test/cop4331/model/DiscountedItemTest.java	121
File: test/cop4331/model/ProductTest.java	125
Screenshots.....	128

Introduction

Project Title: JAVA Shopping Cart

Functional Specification:

The JAVA Shopping Cart application is created to manage all processes of online shopping. The user can create a new account and log in as either a customer or a seller. After logging in, the customer can see a list of items and their cart information, which starts out empty. The user can directly select an item to add to their invoice, and a tally of the total price is kept on the invoice section. Each item has an info button that the user can click on to view details about that specific item, name, description, price, available quantity, and discount if the item has one. Additionally, items can also come in bundles which are a set of different items.

At any point, the invoice section shows the whole list of items chosen, and the customer can proceed to checkout. At the checkout window the customer can confirm the list of purchases or go back and review. The customer is the customer will be prompted to enter their credit card information. After confirming the information. The purchase is complete.

A seller can log in by changing a toggle at the login screen. The seller menu shows the current inventory. The seller can view the inventory information including internal product ID, type, quantity, invoice price, selling price and discounts for items that have ones. On the Profits tab, the seller can view costs, revenue, and profits. The seller can add a product or update an existing one through the new product option, simply specifies the product name, invoice price, sell price, and quantity. If the product already exists, it will be updated. Otherwise, a new item will be added. The seller can also add a new bundle through the bundle option and can also remove any item from the inventory.

Use Cases

Contributors: Rabih, Zachary

List of Use Cases:

1. Customer Log in.
2. Seller Log in.
3. Create new account.
4. Customer adds item to invoice.
5. Customer removes item from invoice.
6. Customer reviews product information.
7. Customer completes payment.
8. Seller adds new item to inventory.
9. Seller adds new bundle to inventory.
10. Seller removes item from inventory.
11. Seller checks revenue, costs, and profit.

Use Cases 1: Customer Log in.

1. User clicks customer radio button.
2. User enters username and password.
3. System verifies login information.
4. System displays Customer menu..

Variation #1: Invalid login.

1. In step 2, user enters invalid username or password.
2. System displays label message "invalid login".

Use Cases 2: Seller Log in.

1. User clicks seller radio button.
2. User enters username and password.
3. System verifies login information..
4. System displays Seller menu

Use Cases 3: Create new account.

1. User clicks "Create Account".
2. System displays Sign up menu.
3. User enters username, password, and confirm password.
4. System verifies signup information.
5. System adds new account to users list.
6. System display login menu.

Variation #1: Account already exists.

1. In step 3, user enters existing username.
2. System displays label message "Account Already Exist".

Variation #2: Password doesn't match.

1. In step 3, user enters different password and password confirm.
2. System displays label message "password mismatch".

Use Case 4: Customer adds item to invoice.

1. User carries out **Customer Log in**.
2. User selects item from list.
3. User clicks add item.
4. System adds selected item to invoice.
5. System displays invoice.

Variation #1: Out of stock.

1. In step 4, Systems display the message "item out of stock".

Use Case 5: Customer removes item from invoice.

1. User carries out **Customer adds item to invoice**.
2. User clicks remove item.
3. System removes item from invoice.

Use Case 6: Customer reviews product information.

1. User carries out **Customer Log in**.
2. User selects item from list.
3. User clicks info button.
4. System displays item name, description, quantity, price and discount.

Use Case 7: Customer completes payment

1. User carries out **Customer adds item to invoice**.
2. User clicks Check out button.
3. System displays credit card menu.
4. User adds credit card information.
5. User clicks pay.
6. System verifies credit card information.
7. System adds revenue.
8. System displays message "Thank you for purchasing".

Use Case 8: Seller adds new item to inventory.

1. User carries out **Seller Log in**.
2. User clicks new item.
3. System displays new item menu.
4. User adds new item details.
5. User clicks add.
6. System verifies new item details.
7. System adds new item to inventory.
8. System updates costs.

Variation #1: Item already in inventory

1. In step 7, System updates details of item in inventory.
2. System updates costs if item quantity is higher than before.

Use Case 9: Seller adds new bundle to inventory.

1. User carries out **Seller Log in**.
2. User clicks new bundle.
3. System displays new bundle menu.
4. User adds new bundle details.
5. User clicks add.
6. System verifies new bundle details.
7. System adds new bundle to inventory.

Use Case 10: Seller removes item from inventory.

1. User carries out **Seller Log in**.
2. User selects item from list.
3. User clicks remove.
4. System removes item from inventory.

Use Case 11: Seller checks revenue, costs, and profit.

1. User carries out **Seller Log in**.
2. User clicks profit.
3. System displays menu showing revenue, costs, and profit.

CRC Cards

Contributors: Rabi, Solan

Profit	
<ul style="list-style-type: none"> - Manages revenue. - Manages costs. - Manages revenue. 	<ul style="list-style-type: none"> - Inventory

Inventory	
<ul style="list-style-type: none"> - Manages product, discounted item, and bundle details. - Stores new Lineltems. 	<ul style="list-style-type: none"> - Lineltem

UserModel	
<ul style="list-style-type: none"> - Manages username and password. - Store user accounts. 	

UserController	
<ul style="list-style-type: none"> - Manages user login and signup. 	

UserUI	
- Displays user login and signup menus.	

CustomerController	
<ul style="list-style-type: none"> - Manages customer invoice - Manages customer payment - Shows item information 	<ul style="list-style-type: none"> - Invoice - LineItem

CustomerUI	
<ul style="list-style-type: none"> - Display customer invoice - Display available inventory - Display payment menu - Display pop up messages 	

SellerController	
<ul style="list-style-type: none"> - Manages seller option - Adds items to inventory - Removes items from inventory - Shows profit data 	<ul style="list-style-type: none"> - Inventory - LineItem

SellerUI	
<ul style="list-style-type: none">- Display seller menu- Display profit data	

Invoice	
<ul style="list-style-type: none">- Manages invoice details	<ul style="list-style-type: none">- InvoiceFormatter

<<Interface>> InvoiceFormatter	
<ul style="list-style-type: none">- Manages invoice formatting	

SimpleFormatter implements InvoiceFormatter	
<ul style="list-style-type: none">- Formats invoice to display title, items, and total price	

<<Interface>> LinItem	
- Creates a LinItem	

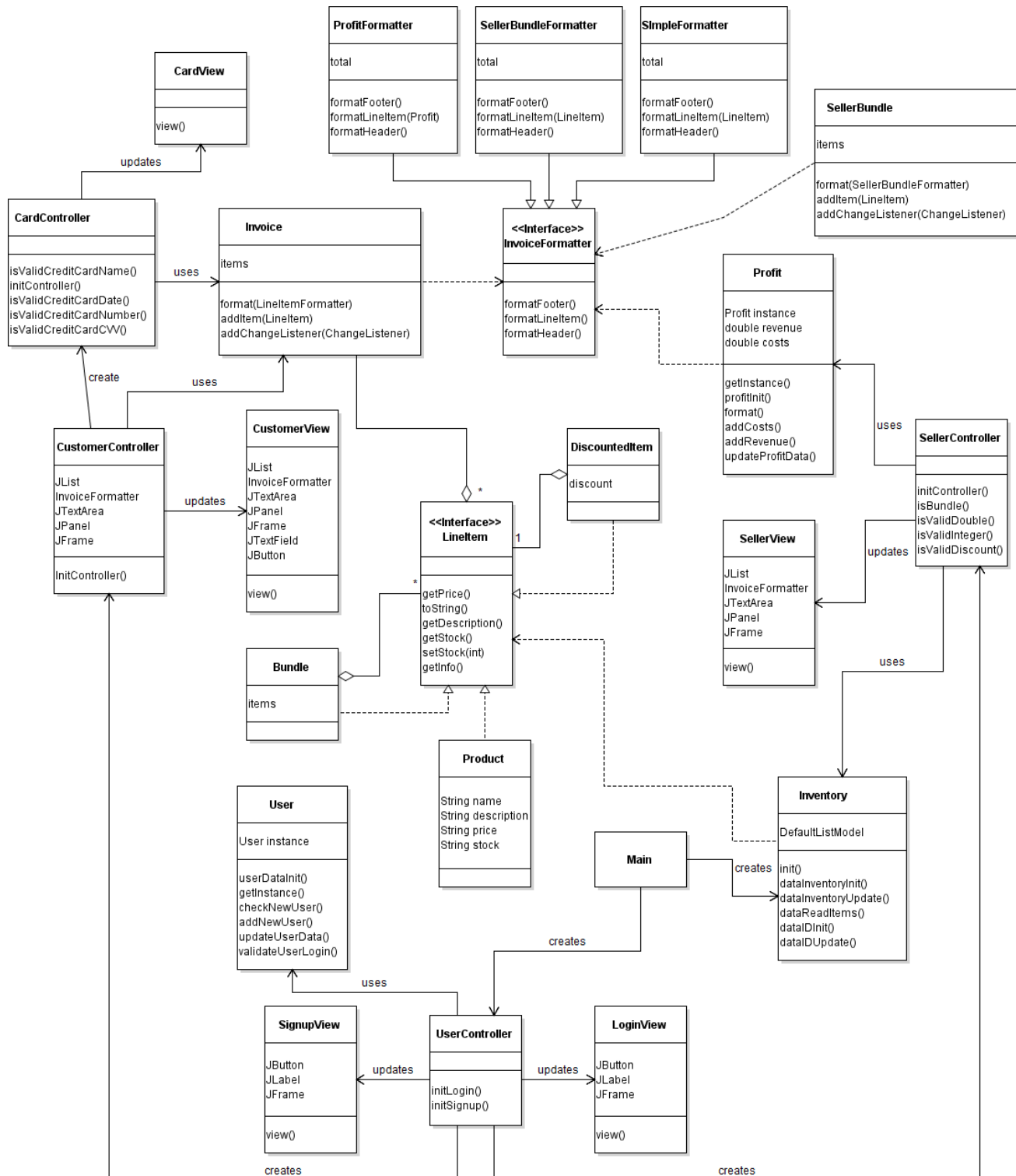
Product implements LinItem	
- Creates a Product	

DiscountedItem implements LinItem	
- Creates a discounted Product	

Bundle implements LinItem	
- Creates a bundle of Products	

Class Diagram

Contributors: Rabih

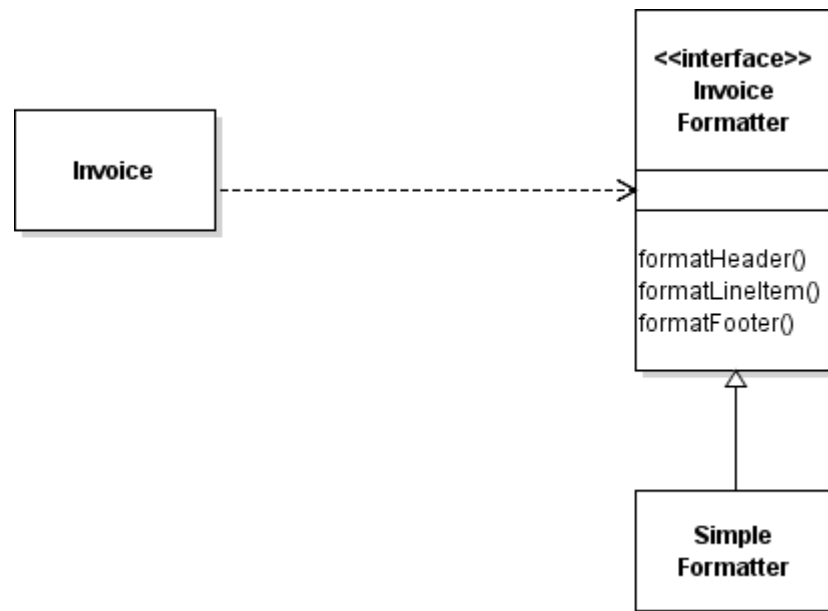


*Class Diagram was updated based on a partially completed version of the program

Design Patterns

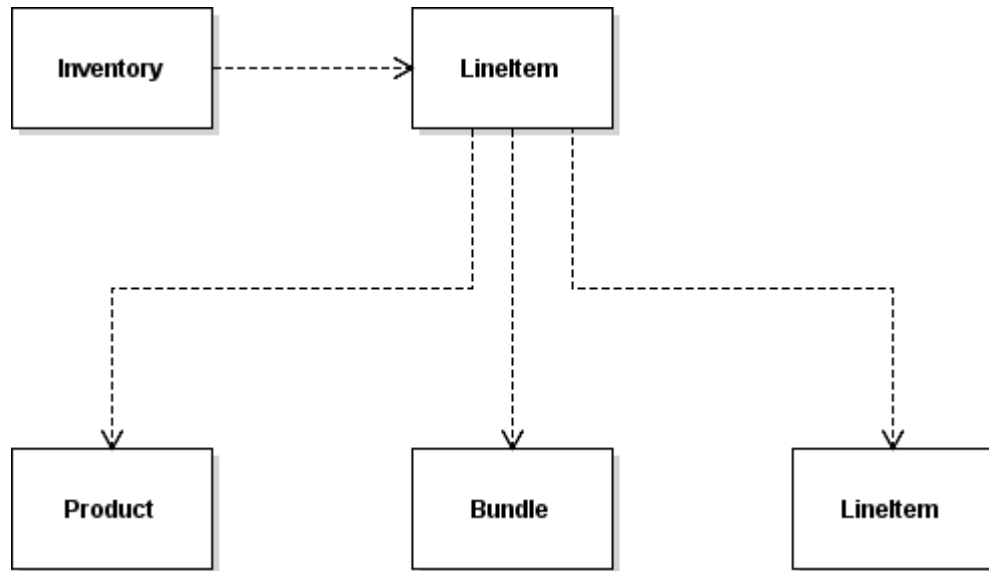
Contributors: Rabi, Gerrell

Strategy Pattern



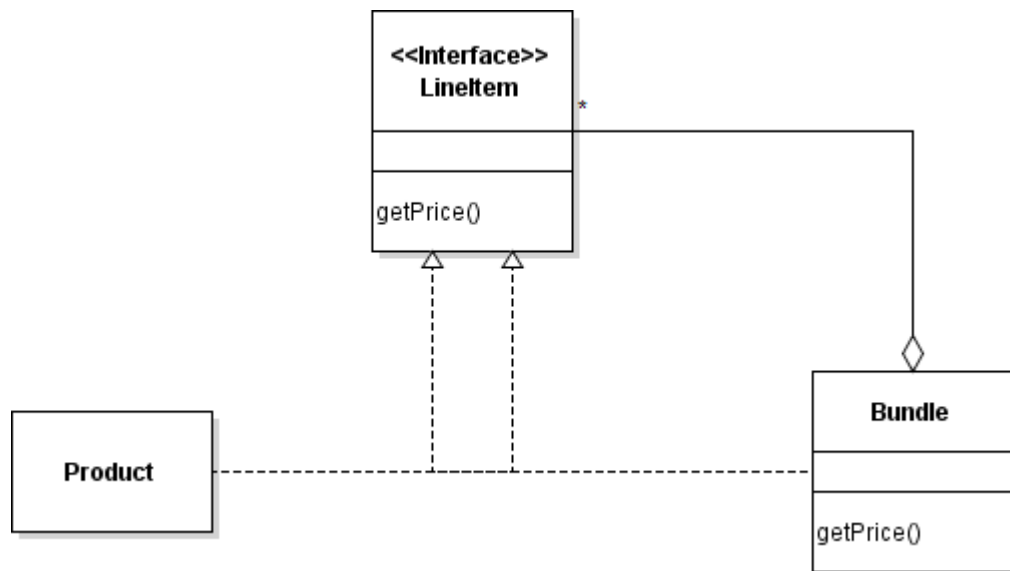
Name in Design Pattern	Actual Name
Context	Invoice
Strategy	InvoiceFormatter
ConcreteStrategy	SimpleFormatter
doWork()	formatHeader() formatLineItem() formatFooter()

Facade Pattern



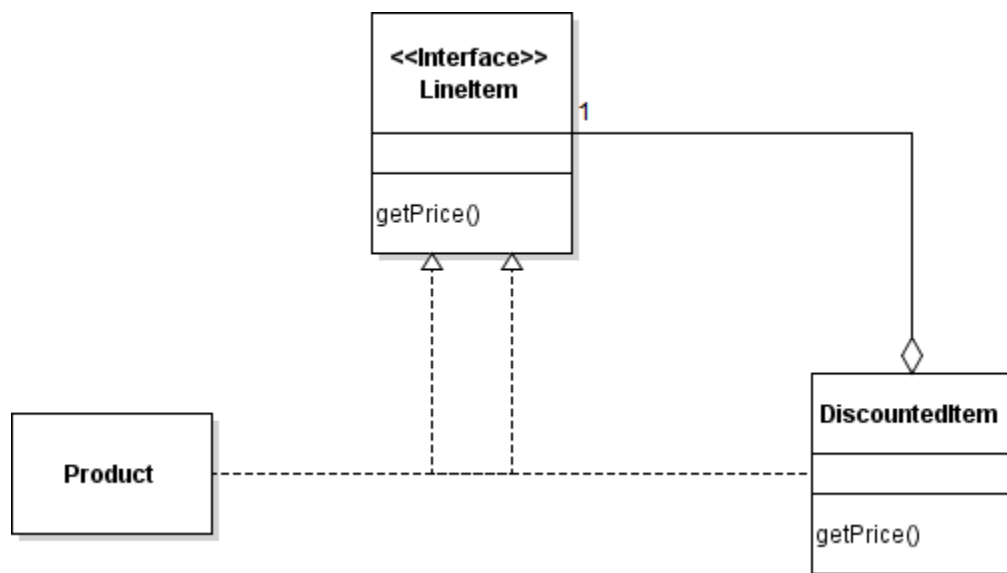
Name in Design Pattern	Actual Name
Client	Inventory
Facade	Lineltem
SubSystemClass 1	Product
SubSystemClass 2	Bundle
SubSystemClass 3	DiscountedItem

Composite Pattern



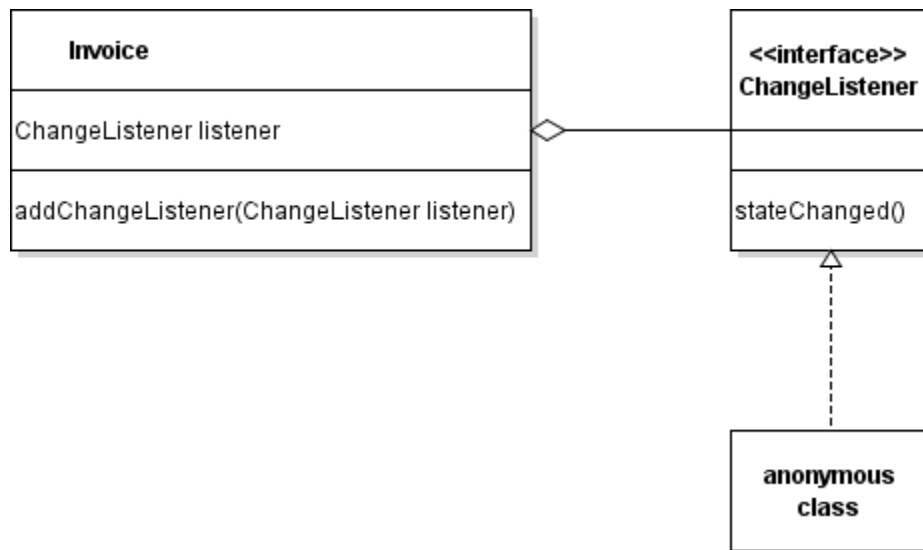
Name in Design Pattern	Actual Name
Primitive	Lineltem
Composite	Bundle
Leaf	Product
method	getPrice()

Decorator Pattern



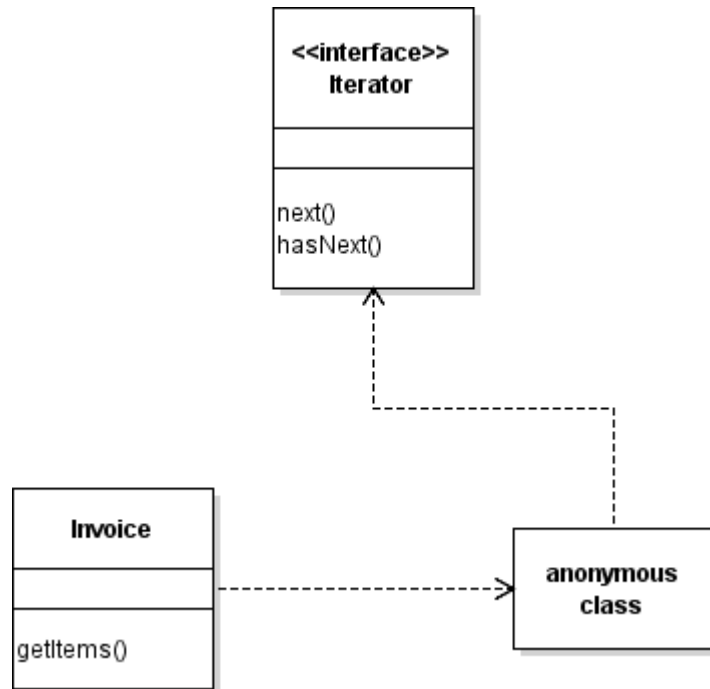
Name in Design Pattern	Actual Name
Component	LinItem
ConcreteComponent	Product
Decorator	DiscountedItem
Method()	getPrice()

Observer Pattern



Name in Design Pattern	Actual Name
Subject	Invoice
Observer	ChangeListener
ConcreteObserver	anonymous class
attach()	addChangeListener()
notify()	stateChanged()

Iterator Pattern



Name in Design Pattern	Actual Name
Aggregate	Invoice
Iterator	Iterator
createIterator()	anonymous class
next()	next()
isDone()	hasNext()
currentItem()	get()

Singleton Pattern

#1

Profit
Profit instance
private Profit() getInstance()

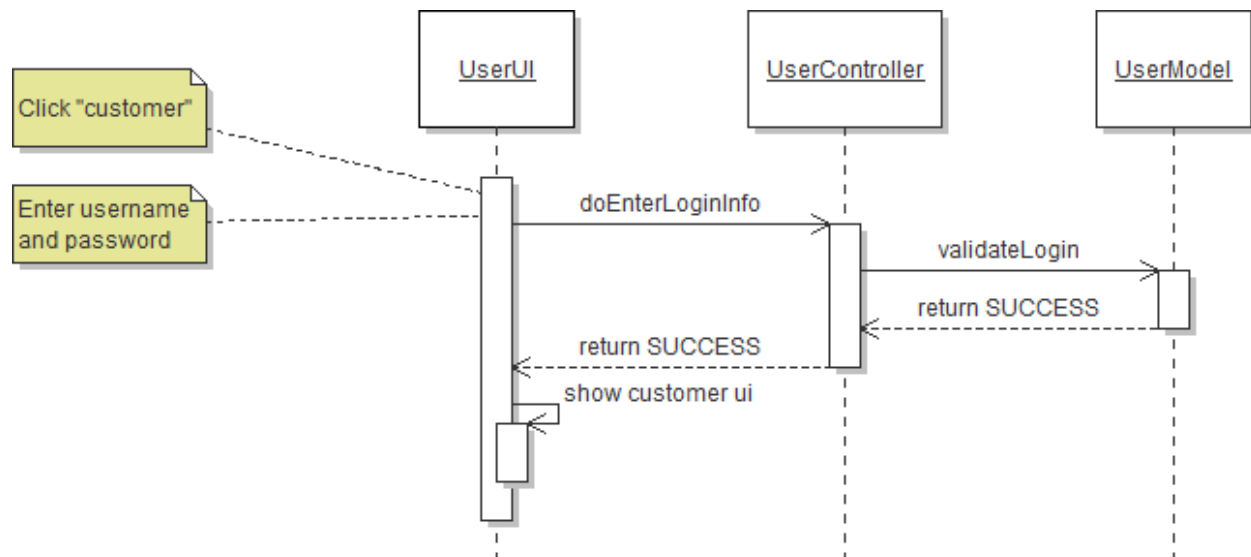
#2

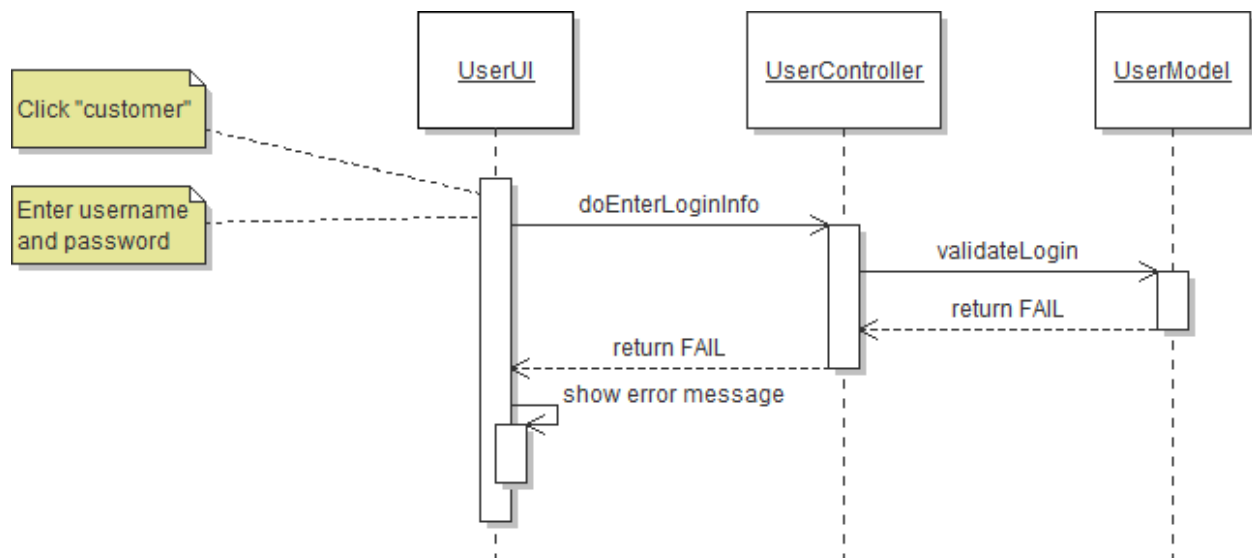
UserModel
UserModel instance
private UserModel() getInstance()

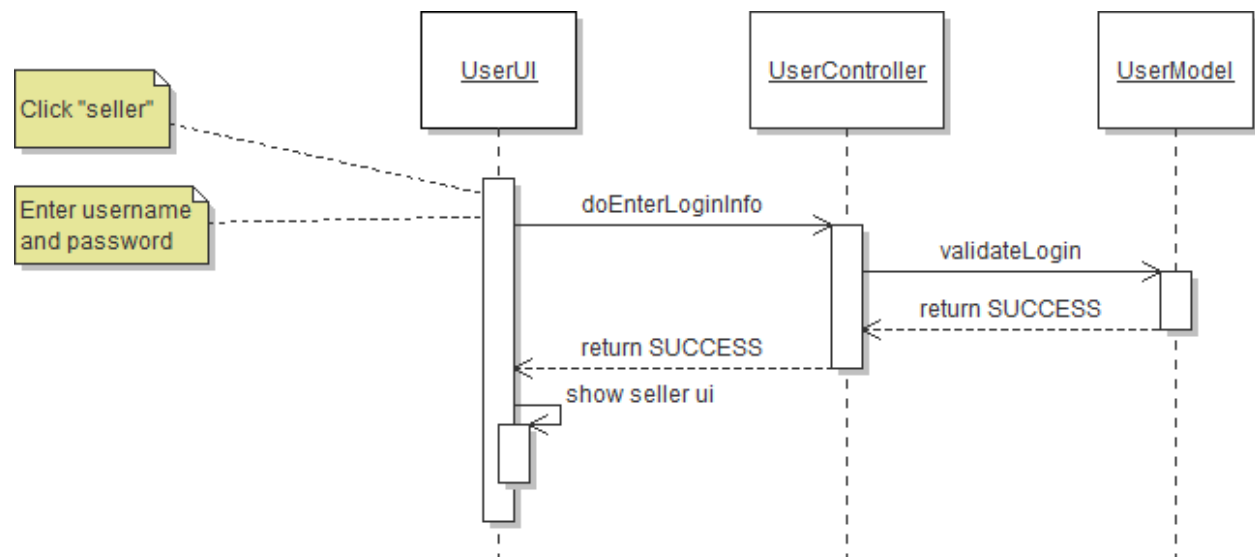
Sequence Diagrams

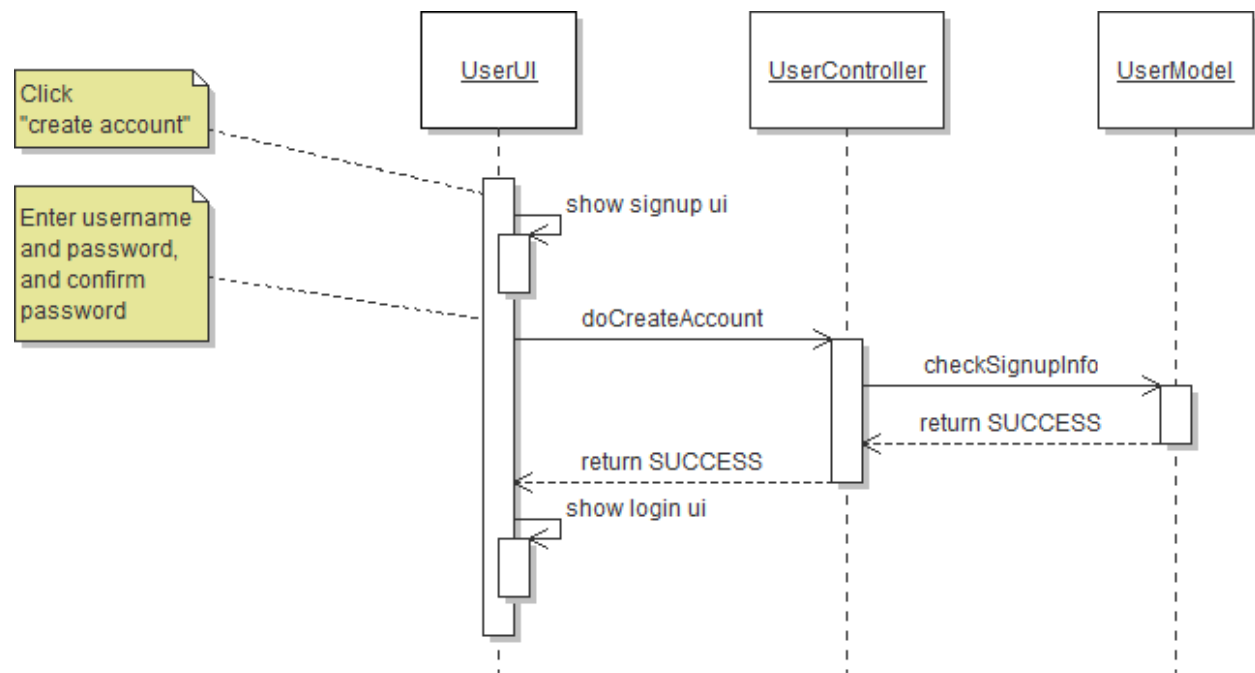
Contributors: Gerrell, Rabih, Zackary

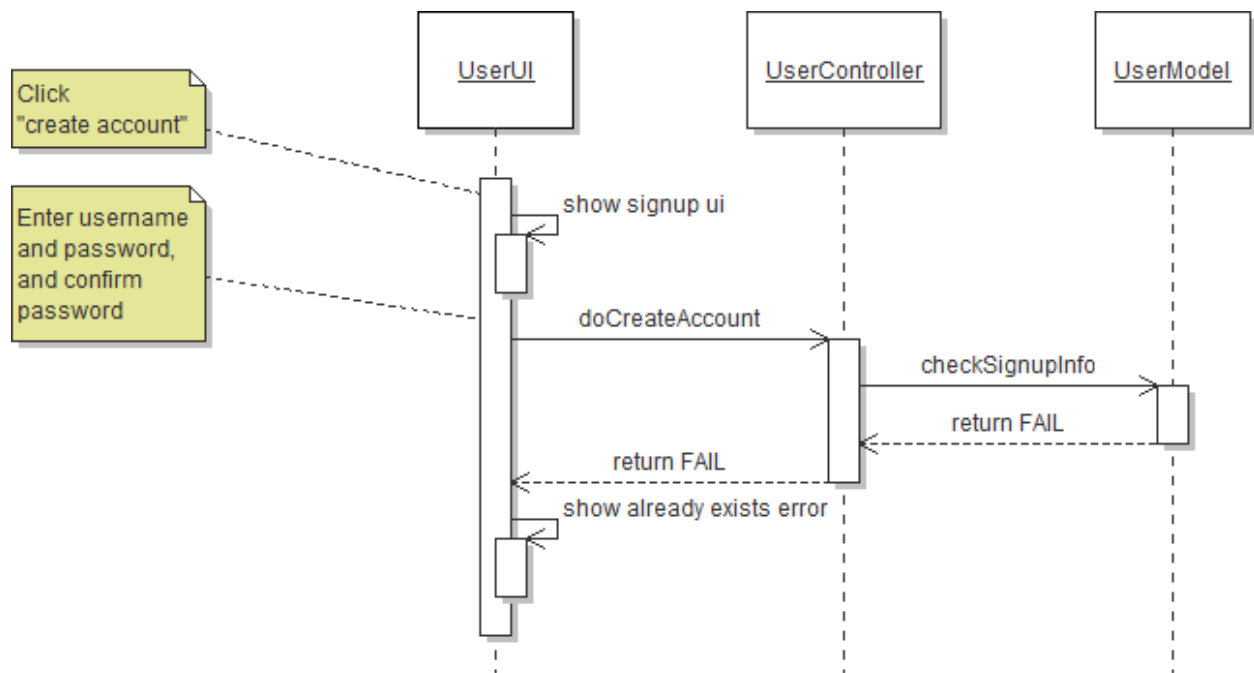
Use Case 1: Customer Log in.

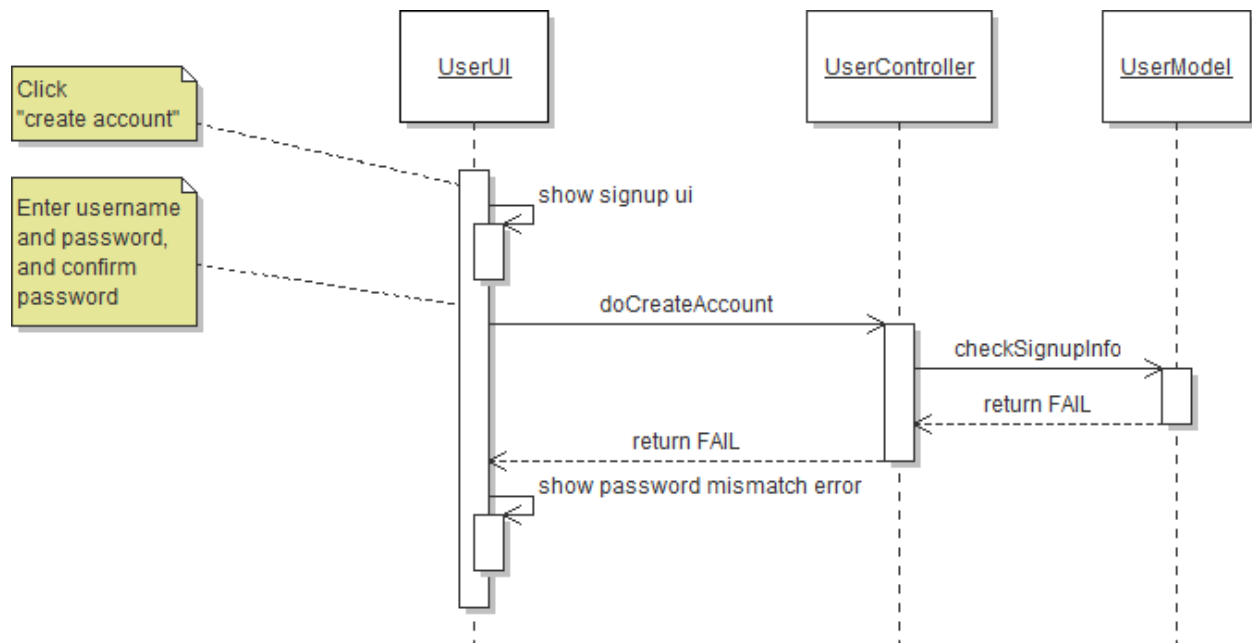


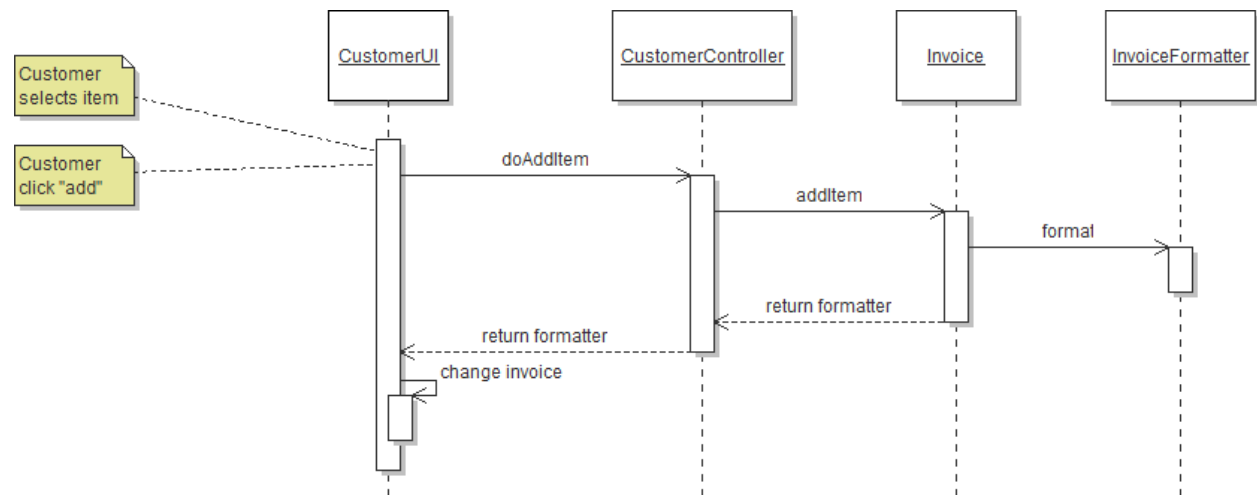
Variation #1: Invalid login.

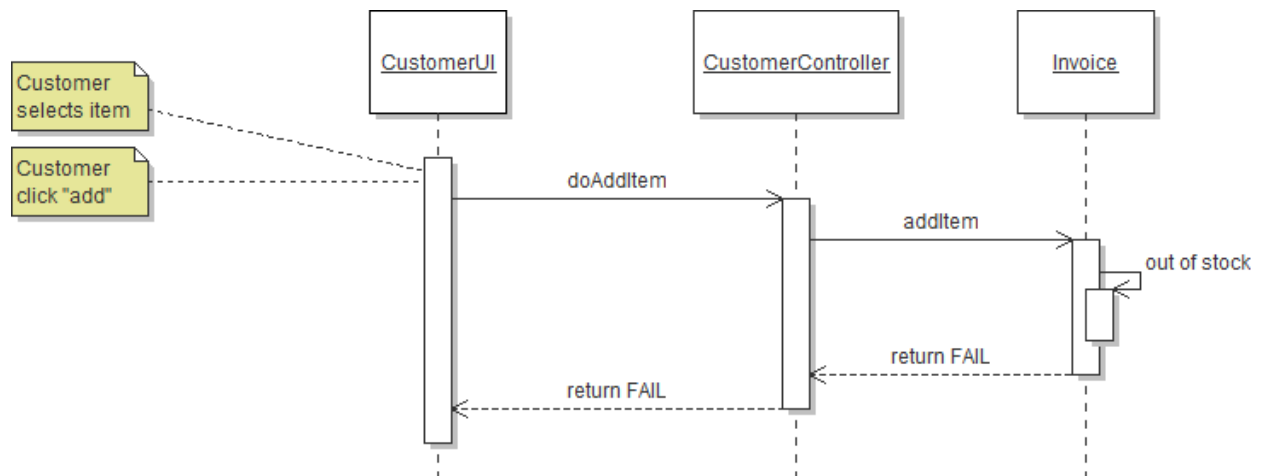
Use Case 2: Seller Log in.

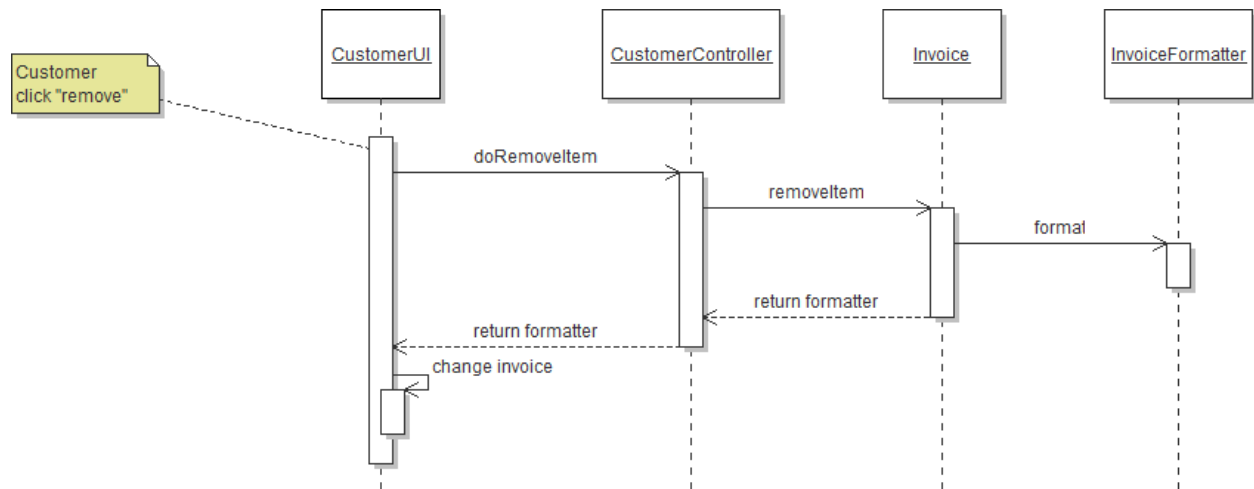
Use Case 3: Create new account.

Variation #1: Account already exists.

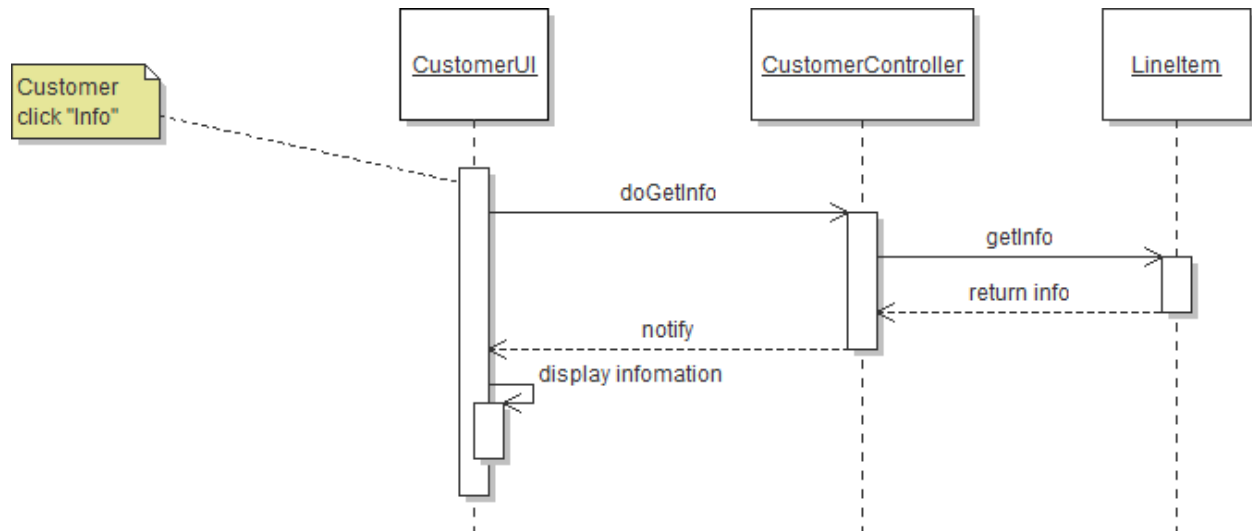
Variation #2: Password doesn't match.

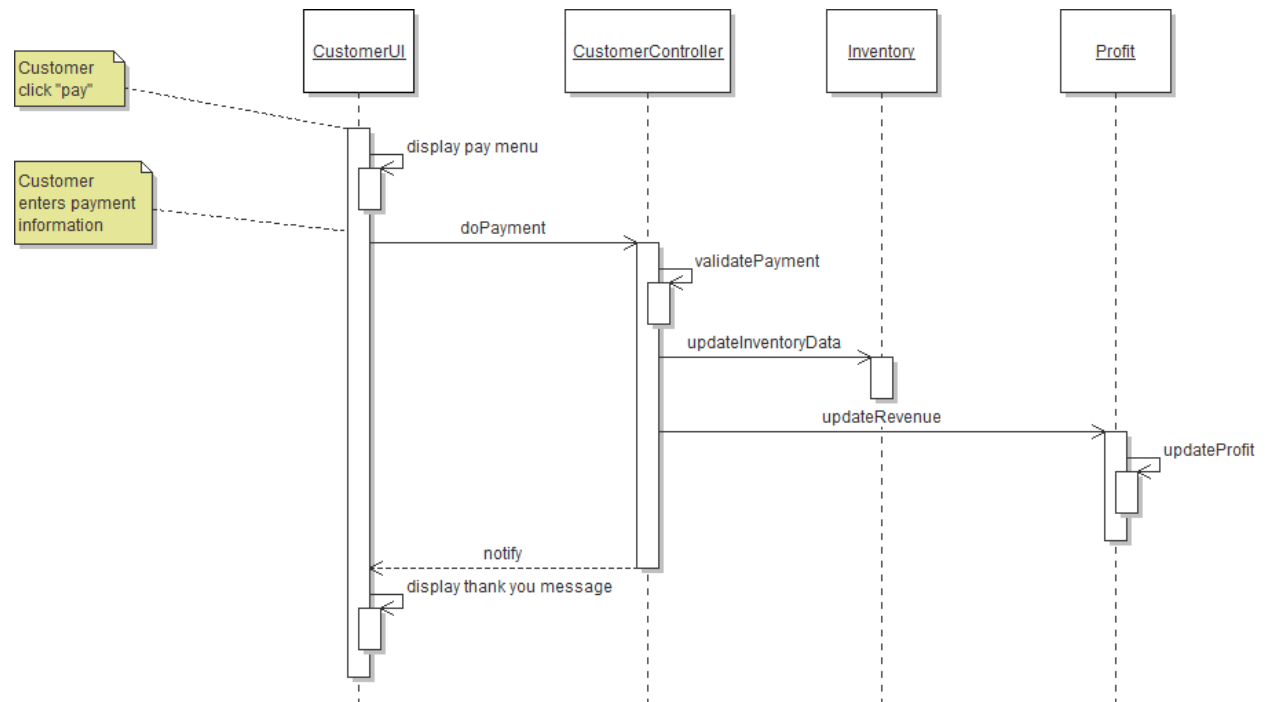
Use Case 4: Customer adds item to invoice.

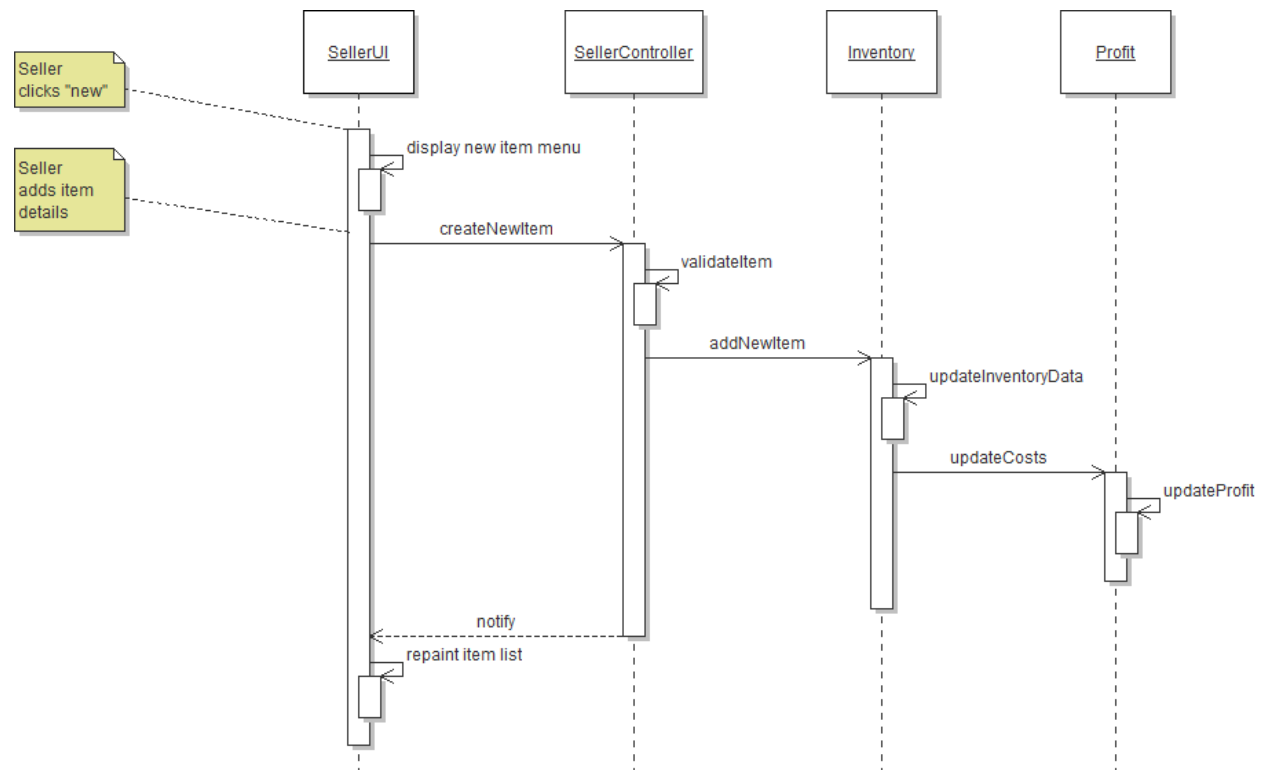
Variation #1: Out of stock.

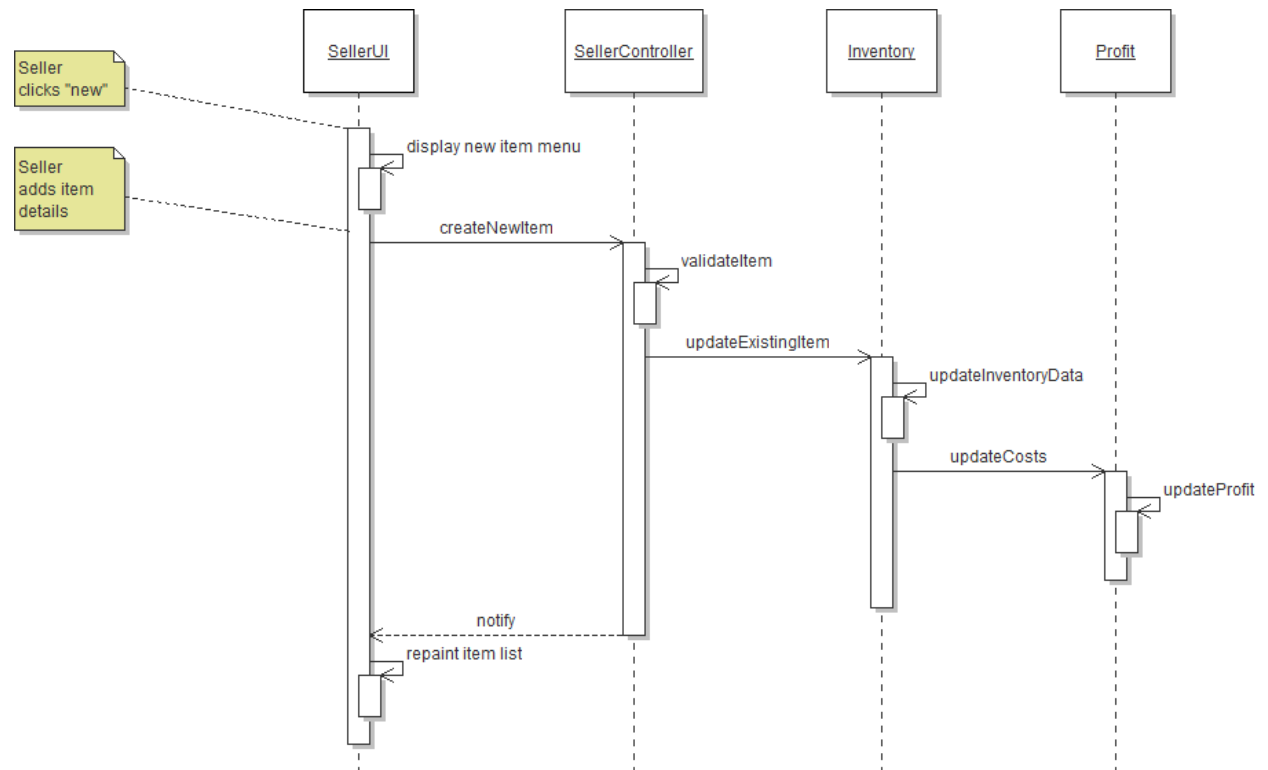
Use Case 5: Customer removes item from invoice.

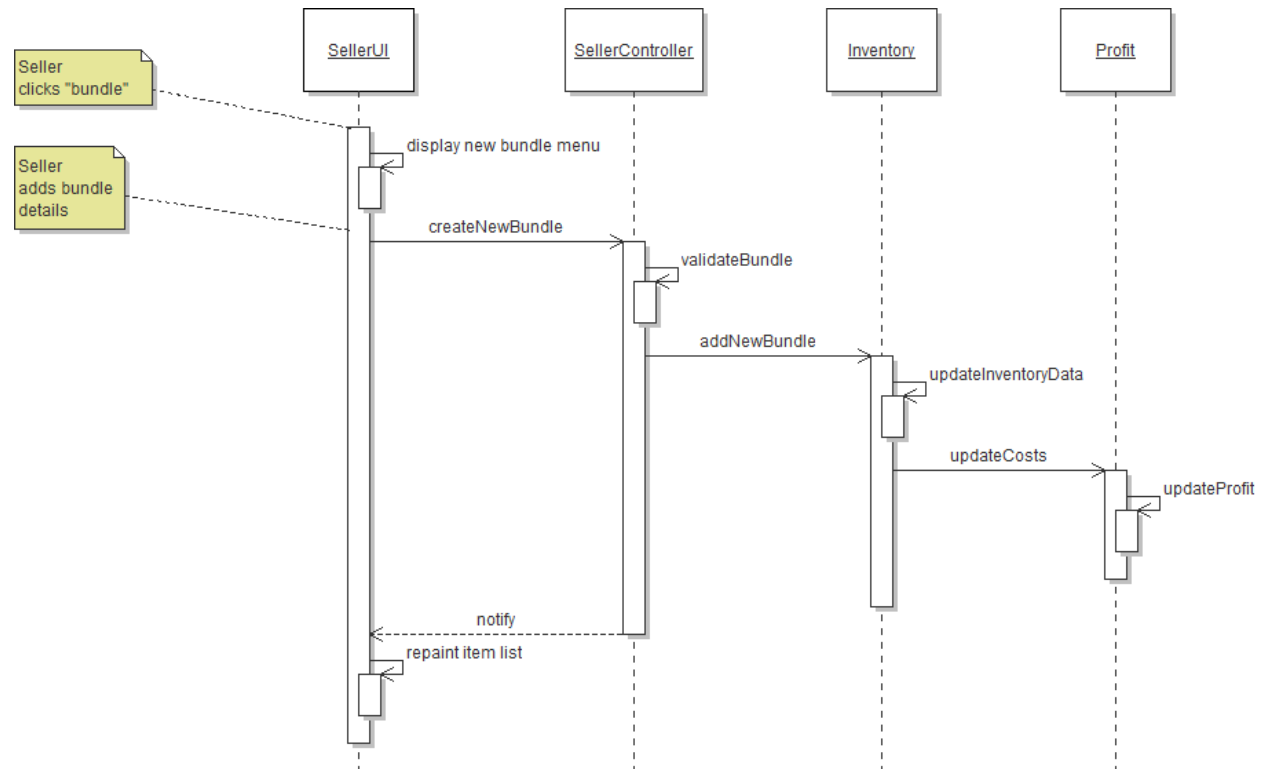
Use Case 6: Customer reviews product information.

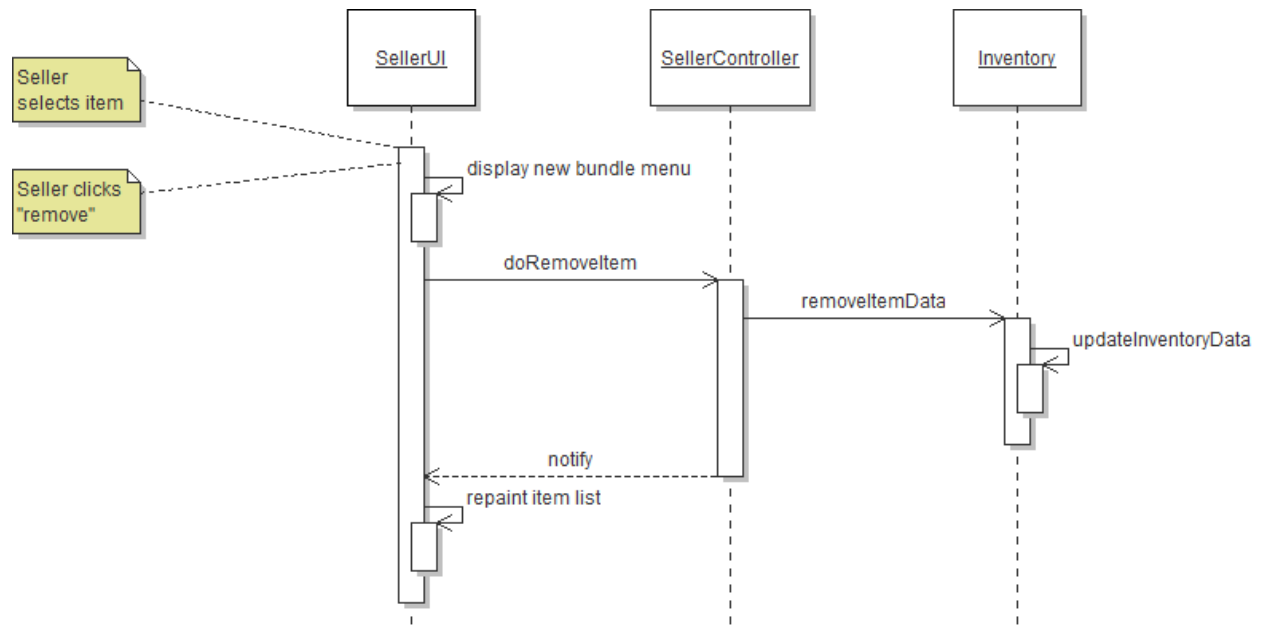


Use Case 7: Customer completes payment.

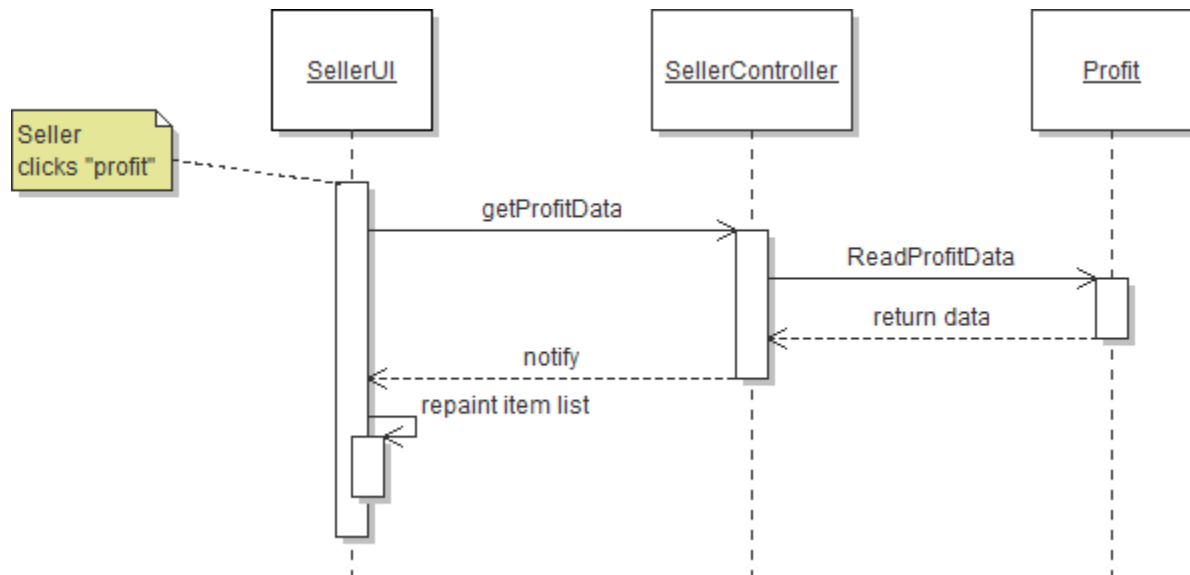
Use Case 8: Seller adds new item to inventory.

Variation #1: Item already in inventory

Use Case 9: Seller adds new bundle to inventory.

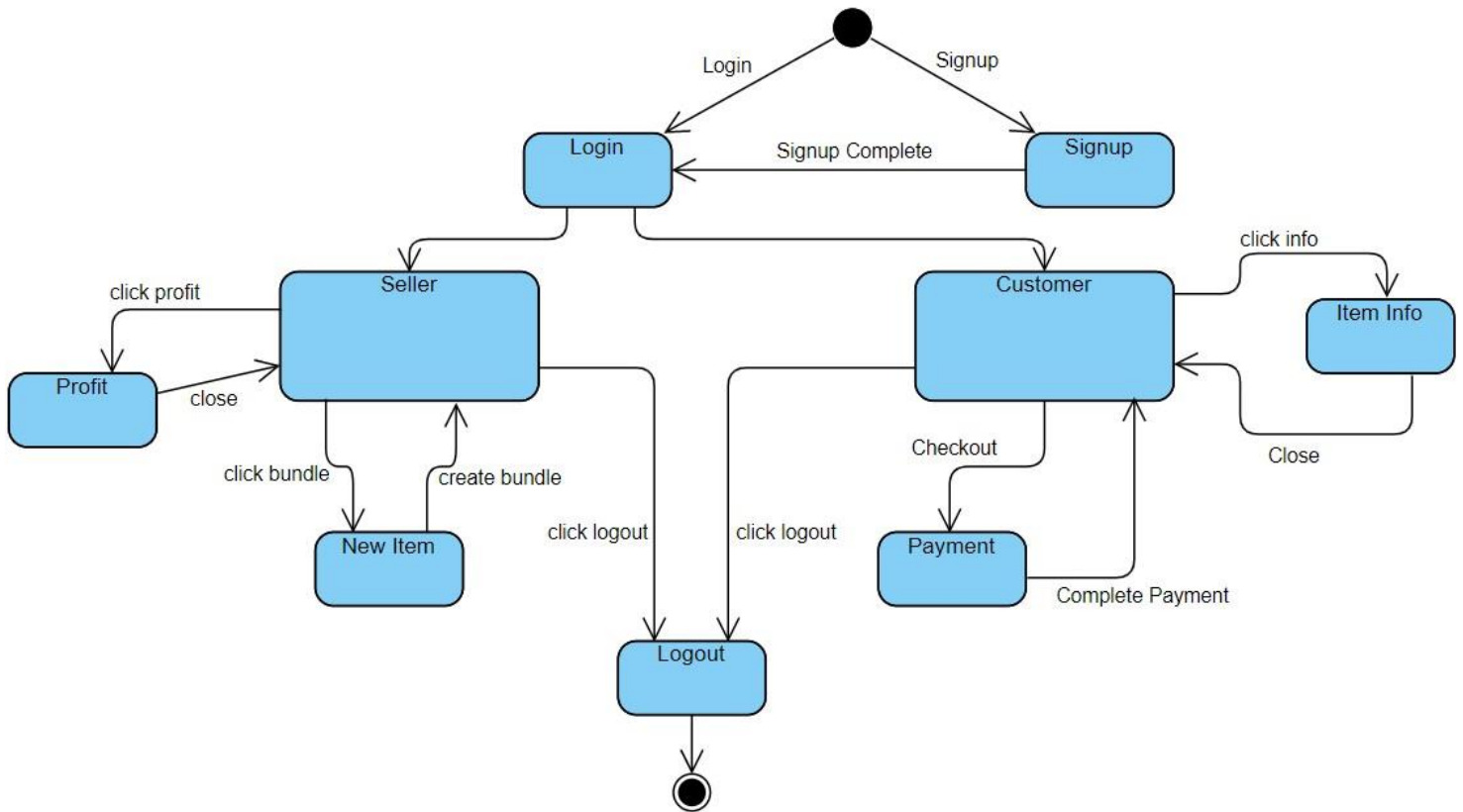
Use Case 10: Seller removes item from inventory.

Use Case 11: Seller checks revenue, costs, and profit.



State Diagrams

Contributors: Solan



Source Code

File: src/cop4331/application/Main.java

```
package cop4331.application;

import cop4331.controller.UserController;

/**
 * Main Class for JAVA Shopping App
 * @author Rabih, Gerrell, Zachary, Solan
 */
public class Main {

    public static void main(String[] args) {
        new Inventory().init();
        User.getInstance().userDataInit();
        new UserController().initLogin();
    }
}
```

File: src/cop4331/application/User.java

```
package cop4331.application;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Singleton class that handles user initialization and management.
 * @author RabiH
 */
public class User {
    private final ArrayList<String[]> users = new ArrayList<>();
    String[] out;
    String send;
    static private User instance = null;

    /**
     * Private constructor.
     */
    private User() { }

    /**
     * Get the only instance of the object.
     * @return instance
     */
    public static User getInstance() {
        if (instance == null) {
            instance = new User();
        }
        return instance;
    }

    /**
     * Initializes User.txt with preliminary usernames and passwords
     * if the file does not exist.
     */
}
```

```

    */
    public void userDataInit(){
        try {
            File f = new File("User.txt");
            if(!f.createNewFile()){} else {
                try (BufferedWriter fw = new BufferedWriter(new
FileWriter("User.txt"))) {
                    //ID,name,desc,stock,price,discount
                    fw.write("Rabih,Khatib");
                    fw.newLine();
                    fw.write("Gerrell,Bones");
                    fw.newLine();
                    fw.write("Zachary,Bundarin");
                    fw.newLine();
                    fw.write("Solan,Degefa");
                    fw.newLine();
                } catch (IOException e) {
                    Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE,
null, e);
                }
            }
        } catch (IOException e) {
            Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
        }
    }

    /**
     * Read user data from User.txt.
     */
    public void readUserData() {
        users.clear();
        try {
            String line;
            String splitBy = ",";
            //parsing a CSV file into BufferedReader class constructor
            BufferedReader br = new BufferedReader(new FileReader("User.txt"));
            while ((line = br.readLine()) != null) { //returns a Boolean
value
                String[] string = line.split(splitBy); // use comma as
separator
                users.add(string);
            }
            br.close();
        } catch (IOException e) {

```



```

        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }
}

/**
 * Updates changes to user data in User.txt.
 */
public void updateUserData() {
    try (BufferedWriter fw = new BufferedWriter(new FileWriter("User.txt")))
{
        for(int i = 0; i < users.size(); i++ ) {
            out = users.get(i);
            send = String.join(",",out);
            fw.write(send);
            fw.newLine();
        }
        fw.close();
    } catch (IOException e) {
        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }
}

/**
 * Adds new signup username and password to User.txt.
 * @param name username of new user.
 * @param pwd password of new user.
 */
public void addNewUser(String name, String pwd) {
    readUserData();
    out = new String [] {name,pwd};
    users.add(out);
    updateUserData();
}

/**
 * Checks if new signup username already exists.
 * @param name username of new user.
 * @return false if username already exists, true otherwise.
 */
public boolean checkNewUser(String name) {
    readUserData();
    for(int i = 0; i < users.size(); i++ ) {
        if (name.equals(users.get(i)[0])) {

```

```

        return false;
    }
}
return true;
}

/**
 * Checks if logging user's username and password are correct.
 * @param name username of logging user.
 * @param pwd password of logging user.
 * @return true if user is valid, false otherwise.
 */
public boolean validateUserLogin(String name, String pwd) {
    readUserData();
    for(int i = 0; i < users.size(); i++ ) {
        if (name.equals(users.get(i)[0])) {
            if (pwd.equals(users.get(i)[1])) {
                return true;
            }
        }
    }
    return false;
}
}

```

File: src/cop4331/application/Inventory.java

```
package cop4331.application;

import cop4331.model.Bundle;
import cop4331.model.DiscountedItem;
import cop4331.model.Invoice;
import cop4331.model.LineItem;
import cop4331.model.Product;
import cop4331.model.SellerBundle;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.DefaultListModel;

/**
 * Class that handles inventory initialization and management.
 * @author Rabih
 */
public class Inventory {
    Bundle bundle = new Bundle(0,0);
    DefaultListModel dlm = new DefaultListModel();
    private final ArrayList<String[]> pItems = new ArrayList<>();
    String line;
    String splitBy = ",";
    String current;
    String send;
    String[] out;
    Product product;
    DiscountedItem discount;
    private double p,d, costs;
    Integer id,s, index,currentID;
```

```

public Inventory() {}

public void init() {
    dataInventoryInit();
    dataIDInit();
}

/**
 * Initializes Inventory.txt.
 * Initializes with preliminary item data if the file does not exist.
 * Updates the current ID and calls dataUpdateFromFile().
 */
public void dataInventoryInit(){
    try {
        File f = new File("Inventory.txt");
        if(!f.createNewFile()){} else {
            try (BufferedWriter fw = new BufferedWriter(new
FileWriter("Inventory.txt"))) {
                //ID,name,desc,stock,price,discount
                fw.write("0,Strawberry Ice Cream,Delicious strawberry
flavored ice cream.,5,1.99,0");
                fw.newLine();
                fw.write("1,Hammer,Hammer to smash things.,5,19.95,0");
                fw.newLine();
                fw.write("2,Television,30 inch widescreen TV.,5,159.95,0");
                fw.newLine();
                fw.write("3,Assorted Nails,20 Assorted Nails ideal for
construction.,10,9.95,0");
                fw.newLine();
                fw.write("4,Milk,3.78L far free milk.,10,5.99,0");
                fw.newLine();
                fw.write("5,Water Bottle,reusable water bottle for
children.,5,19.95,20");
                fw.newLine();
                //ID,name,discount,stock,names...
                fw.write("6,Bundle,10,17,Hammer,Assorted Nails,Water
Bottle");
                fw.close();
            } catch (IOException e) {
                Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE,
null, e);
            }
            currentID = 7;
        }
    }
}

```

```

    } catch (IOException e) {
        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }

    dataUpdateFromFile();
}

/**
 * Initializes ID.txt.
 */
public void dataIDInit(){
    try {
        File f = new File("ID.txt");
        if(!f.createNewFile()){} else {
            try (BufferedWriter fw = new BufferedWriter(new
FileWriter("ID.txt"))) {
                fw.write(String.format("%d",currentID));
                fw.close();
            } catch (IOException e) {
                Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE,
null, e);
            }
        }
    } catch (IOException e) {
        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }
}

/**
 * Reads items from Inventory.txt into ArrayList.
 */
public void dataReadItems(){
    try {
        //parsing a CSV file into BufferedReader class constructor
        BufferedReader br = new BufferedReader(new
FileReader("Inventory.txt"));
        while ((line = br.readLine()) != null) {    //returns a Boolean
value
                String[] string = line.split(splitBy);    // use comma as
separator
                pItems.add(string);
            }
    }
}

```

```

        br.close();
    } catch (IOException e) {
        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }
}

/**
 * Reads ID from ID.txt.
 */
public void dataReadID(){
    //parsing a CSV file into BufferedReader class constructor
    Scanner sc;
    try {
        sc = new Scanner(new File("ID.txt"));
        String stringID = sc.next();
        currentID = Integer.parseInt(stringID);
        sc.close();
    } catch (FileNotFoundException e) {
        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }
}

/**
 * Writes current ID to ID.txt.
 */
public void dataUpdateID(){
    try (BufferedWriter fw = new BufferedWriter(new FileWriter("ID.txt"))) {
        fw.write(String.format("%d",currentID));
        fw.close();
    } catch (IOException e) {
        Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
    }
}

/**
 * Updates DefaultListModel with data from Inventory.txt.
 */
public void dataUpdatefromFile(){
    pItems.clear();
    dlm.removeAllElements();
    dataReadItems();
}

```

```

for(int i = 0; i < pItems.size(); i++ ) {
    // Searching for and updating bundles
    if ("Bundle".equals(pItems.get(i)[1])) {
        bundle = new Bundle(0,0);
        for (int j = 4; j <= (pItems.get(i).length) - 1; j++) {
            current = pItems.get(i)[j];
            for(int k = 0; k < pItems.size(); k++ ) {
                if (pItems.get(k)[1].equals(current)) {
                    index = k;
                    break;
                }
            }
            id = Integer.parseInt(pItems.get(index)[0]);
            p = Double.parseDouble(pItems.get(index)[4]);
            s = Integer.parseInt(pItems.get(index)[3]);
            product = new
Product(id,pItems.get(index)[1],pItems.get(index)[2],p,s);
            bundle.add(product);
        }
        id = Integer.parseInt(pItems.get(i)[0]);
        s = Integer.parseInt(pItems.get(i)[3]);
        bundle.setID(id);
        bundle.setStock(s);
        d = Double.parseDouble(pItems.get(i)[2]);
        discount = new DiscountedItem(bundle,d);
        discount.setID(id);
        dlm.addElement(discount);

        costs += (bundle.getPrice()/2) *
bundle.getStock();
    }

    else {
        id = Integer.parseInt(pItems.get(i)[0]);
        p = Double.parseDouble(pItems.get(i)[4]);
        s = Integer.parseInt(pItems.get(i)[3]);
        product = new Product(id,pItems.get(i)[1],pItems.get(i)[2],p,s);
        costs += (product.getPrice()/2) * product.getStock();

        if ("0".equals(pItems.get(i)[5])) {
            dlm.addElement(product);
        }
        else {
            d = Double.parseDouble(pItems.get(i)[5]);
            dlm.addElement(new DiscountedItem(product,d));
        }
    }
}

```

```

        }
    }
}
Profit.getInstance().dataProfitInit(costs);
}

/**
 * Updates item quantity after consumer makes payment.
 * @param invoice invoice of purchased items by consumer.
 */
public void dataUpdateStock(Invoice invoice){
    pItems.clear();
    dataReadItems();

    Iterator<LineItem> iterator = invoice.getItems();
    LineItem currentItem;
    while(iterator.hasNext()) {
        currentItem = iterator.next();
        if ("Bundle".equals(pItems.get(currentItem.getID())[1])) {
            s = Integer.parseInt(pItems.get(currentItem.getID())[3]);
            pItems.get(currentItem.getID())[3] = Integer.toString(s - 1);
        }

        else if (currentItem instanceof Product) {
            s = Integer.parseInt(pItems.get(currentItem.getID())[3]);
            pItems.get(currentItem.getID())[3] = Integer.toString(s -
1);
        }
        else if (currentItem instanceof DiscountedItem) {
            s = Integer.parseInt(pItems.get(currentItem.getID())[3]);
            pItems.get(currentItem.getID())[3] = Integer.toString(s -
1);
        }
    }
    dataInventoryUpdate();
}

/**
 * Adds new item to Inventory.txt.
 * @param name name of new item.
 * @param description description of new item.
 * @param quantity quantity of new item.
 * @param price price of new item.
 * @param discount discount percentage of new item.
 */

```



```

    public void dataSellerNewItem(String name,String description,String
quantity,String price,String discount) {
        pItems.clear();
        dataReadItems();
        dataReadID();

        double addCosts;
        Integer oldQuantity, found = 0;

        for(int i = 0; i < pItems.size(); i++ ) {
            if(name.equals(pItems.get(i)[1])) {
                found = 1;
                oldQuantity = Integer.parseInt(pItems.get(i)[3]);
                pItems.get(i)[2] = description;
                pItems.get(i)[3] = quantity;
                pItems.get(i)[4] = price;
                pItems.get(i)[5] = discount;
                if (oldQuantity < Integer.parseInt(quantity)) {
                    addCosts = Double.parseDouble(price)/2 *
(Integer.parseInt(quantity)-oldQuantity);
                    Profit.getInstance().addCosts(addCosts);
                }
                break;
            }
        }

        if (found != 1) {
            String newID = currentID.toString();
            currentID += 1;
            String[] newItem = new String []
{newID,name,description,quantity,price,discount};
            addCosts = Double.parseDouble(price)/2 * Integer.parseInt(quantity);
            Profit.getInstance().addCosts(addCosts);
            pItems.add(newItem);
            dataUpdateID();
        }
        dataInventoryUpdate();
    }

    /**
     * Adds new item to Inventory.txt.
     * @param sb items included in bundle.
     * @param quantity quantity of new bundle.
     * @param discount discount percentage of new bundle.
     */

```

```

    public void dataSellerBundleItem(SellerBundle sb,String quantity, String
discount) {
        pItems.clear();
        dataReadItems();
        dataReadID();

        Iterator<LineItem> iterator = sb.getItems();
        String bundleID = currentID.toString();
        currentID += 1;
        String currentName;
        double addCosts = 0;
        ArrayList<String> list = new ArrayList<>();

        list.add(bundleID);
        list.add("Bundle");
        list.add(discount);
        list.add(quantity);
        while(iterator.hasNext()) {
            currentName = iterator.next().getName();
            list.add(currentName);
            for(int i = 0; i < pItems.size(); i++ ) {
                if(currentName.equals(pItems.get(i)[1])) {
                    addCosts += Double.parseDouble(pItems.get(i)[4]);
                    break;
                }
            }
        }
        String[] newBundle = new String[list.size()];
        for(int i = 0; i < list.size(); i++ ) {
            newBundle[i] = list.get(i);
        }
        pItems.add(newBundle);

        addCosts = addCosts/2 * Integer.parseInt(quantity);
        Profit.getInstance().addCosts(addCosts);
        dataUpdateID();
        dataInventoryUpdate();
    }

    /**
     * Removes item from Inventory.txt.
     * @param item item to be removed
     */
    public void dataSellerRemoveItem(LineItem item){
        pItems.clear();

```

```

        dataReadItems();

        for(int i = 0; i < pItems.size(); i++ ) {
            if (item.getID() == Integer.parseInt(pItems.get(i)[0])) {
                pItems.remove(pItems.get(i));
                break;
            }
        }
        removeAdditionalBundles(item);
        dataInventoryUpdate();
    }

    /**
     * Recursively removes bundles featuring a removed item from Inventory.txt.
     * @param item item to be removed
     */
    public void removeAdditionalBundles(LineItem item) {
        for(int i = 0; i < pItems.size(); i++ ) {
            if ("Bundle".equals(pItems.get(i)[1]) &&
!"Bundle".equals(item.toString())) {
                for (int j = 4; j <= (pItems.get(i).length) - 1; j++) {
                    current = pItems.get(i)[j];
                    if (current.equals(item.getName())) {
                        pItems.remove(pItems.get(i));
                        removeAdditionalBundles(item);
                        break;
                    }
                }
            }
        }
    }

    /**
     * Updates Inventory.txt with data from ArrayList.
     */
    public void dataInventoryUpdate(){
        try (BufferedWriter fw = new BufferedWriter(new
FileWriter("Inventory.txt"))){
            for(int i = 0; i < pItems.size(); i++ ) {
                out = pItems.get(i);
                send = String.join(",",out);
                fw.write(send);
                if (i < pItems.size() - 1) { fw.newLine(); }
            }
            fw.close();
        }
    }

```

```
        } catch (IOException e) {
            Logger.getLogger(Inventory.class.getName()).log(Level.SEVERE, null,
e);
        }
    }

    public DefaultListModel getDefaultListModel(){
        dataUpdateFromFile();
        return dlm;
    }
}
```

File: src/cop4331/application/Profit.java

```
package cop4331.application;

import cop4331.gui.CardView;

import cop4331.formatter.ProfitFormatter;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Singleton class that handles profit initialization and management.
 * @author Rabih
 */
public class Profit {
    private double revenue;
    private double costs;
    static private Profit instance = null;

    /**
     * Private constructor.
     */
    private Profit() { }

    /**
     * Get the only instance of the object.
     * @return instance
     */
    public static Profit getInstance() {
        if (instance == null) {
            instance = new Profit();
        }
        return instance;
    }
}
```

/**

```

    * Initializes Profit.txt with preliminary revenue, cost,
    * and profit data if the file does not exist.
    * @param initCosts
    */
    public void dataProfitInit(double initCosts){
        try {
            File f = new File("Profit.txt");
            this.costs = initCosts;
            if(!f.createNewFile()){} else {
                try (BufferedWriter fw = new BufferedWriter(new
FileWriter("Profit.txt"))) {
                    fw.write(String.format("0,%f,%f",costs, 0 - costs));
                    fw.close();
                } catch (IOException e) {
                    Logger.getLogger(Profit.class.getName()).log(Level.SEVERE, null,
e);
                }
            }
        } catch (IOException e) {
            Logger.getLogger(Profit.class.getName()).log(Level.SEVERE, null, e);
        }
    }

    /**
    * Reads revenue and cost data from Profit.txt.
    */
    public void extractRevenueCost() {
        try {
            String line;
            String splitBy = ",";
            //parsing a CSV file into BufferedReader class constructor
            BufferedReader br = new BufferedReader(new
FileReader("Profit.txt"));
            while ((line = br.readLine()) != null) {    //returns a Boolean
value
                String[] string = line.split(splitBy);    // use comma as
separator

                this.revenue = Double.parseDouble(string[0]);
                this.costs = Double.parseDouble(string[1]);
            }
            br.close();
        } catch (IOException e) {
            Logger.getLogger(Profit.class.getName()).log(Level.SEVERE, null, e);
        }
    }
}

```

```

/**
 * Adds revenue to Profit.txt after customer makes a purchase.
 * @param cardview customer's card data
 */
public void addRevenue(CardView cardview) {
    extractRevenueCost();
    this.revenue += cardview.getFormatter().getTotal();
    updateProfitData();
}

/**
 * Adds costs to Profit.txt after seller adds new items or bundles.
 * @param addCosts costs of new item or bundle.
 */
public void addCosts(double addCosts) {
    extractRevenueCost();
    this.costs += addCosts;
    updateProfitData();
}

/**
 * Updates Profit.txt with any changes made.
 */
public void updateProfitData() {
    try (BufferedWriter fw = new BufferedWriter(new
FileWriter("Profit.txt"))) {
        fw.write(String.format("%f,%f,%f",this.revenue, this.costs,
this.revenue - this.costs));
        fw.close();
    } catch (IOException e) {
        Logger.getLogger(Profit.class.getName()).log(Level.SEVERE, null, e);
    }
}

public String format(ProfitFormatter formatter) {
    String r = formatter.formatHeader();
    r += formatter.formatLineItem(new Profit());
    return r + formatter.formatFooter();
}

public double getRevenue() {extractRevenueCost(); return revenue;}

```

```
    public double getCosts() {extractRevenueCost(); return costs;}  
    public double getProfit() {extractRevenueCost(); return this.revenue -  
this.costs;}  
}
```


File: src/cop4331/model/LineItem.java

```
package cop4331.model;

/**
 * A line item in an invoice.
 * @author modified by Rabi
 */
public interface LineItem
{
    /**
     * Gets the id of this line item.
     * @return the price
     */
    int getID();

    /**
     * Gets the price of this line item.
     * @return the price
     */
    double getPrice();

    /**
     * Gets the name of this line item.
     * @return the description
     */
    @Override
    String toString();

    /**
     * Gets the name of this line item without any additions.
     * @return the description
     */
    String getName();

    /**
     * Gets the description of this line item.
     * @return the description
     */
    String getDescription();

    /**
     * Gets the stock of this line item.
     * @return the description
     */
}
```

```
    */  
    int getStock();  
  
    /**  
     * Sets the stock of this line item.  
     * @param stock  
     * @return the description  
     */  
    int setStock(int stock);  
  
    /**  
     * Gets the full info of this line item.  
     * @return the description  
     */  
    String getInfo();  
  
    /**  
     * Gets the full seller info of this line item.  
     * @return the description  
     */  
    String getSellerInfo();  
}
```

File: src/cop4331/model/Product.java

```
package cop4331.model;

/**
 * A product with a price, quantity and description.
 * @author modified by Zachary
 */
public class Product implements LineItem {
    /**
     * Constructs a product.
     * @param id the id of the item
     * @param name the name
     * @param description the description
     * @param price the price
     * @param stock the stock
     */
    public Product(int id, String name, String description, double price, int
stock) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
        this.stock = stock;
    }

    @Override
    public int getID() { return id; }

    @Override
    public double getPrice() { return price; }

    public double setPrice(double price) { return this.price = price; }

    @Override
    public String toString() { return name; }

    @Override
    public String getName() { return name; }

    @Override
    public String getDescription() { return description; }

    @Override
```

```
public int getStock() { return stock; }

@Override
public int setStock(int stock) { return this.stock = stock; }

@Override
public String getInfo() { return (String.format(
    "Name: %s\nDescription: %s\nPrice: $%.2f\nQuantity: %d\n", name,
description, price, stock)); }

@Override
public String getSellerInfo() { return (String.format(
    "ID: %d\nName: %s\nDescription: %s\nInvoice Price: $%.2f\nSell Price:
$%.2f\nQuantity: %d\n", id, name, description, price/2, price, stock)); }

private final int id;
private final String name;
private final String description;
private double price;
public int stock;
}
```

File: src/cop4331/model/DiscountedItem.java

```
package cop4331.model;

/**
 * A decorator for an item that applies a discount.
 * @author modified by Zachary
 */
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.id = item.getID();
        this.discount = discount;
        this.stock = item.getStock();
    }

    @Override
    public int getID() { return item.getID(); }

    public int setID(int id) { return this.id = id; }

    @Override
    public double getPrice() {
        return item.getPrice() * (1 - discount / 100);
    }

    @Override
    public String toString() {
        return item.toString() + " (Discount " + discount + "%)";
    }

    @Override
    public String getName() {
        return item.toString();
    }

    @Override
```

```

    public String getDescription() {
        return item.getDescription();
    }

    @Override
    public int getStock() { return stock; }

    @Override
    public int setStock(int stock) { return this.stock = stock; }

    @Override
    public String getInfo() { return (String.format(
        "Name: %s\nDescription: %s\nPrice: $%.2f\nQuantity: %d\nDiscount:
        %.2f%%\n", item.toString(), getDescription(), getPrice(), getStock(), discount));
    }

    @Override
    public String getSellerInfo() { return (String.format(
        "ID: %d\nName: %s\nDescription: %s\nInvoice Price: $%.2f\nSell Price:
        $%.2f\nQuantity: %d\nDiscount: %.2f%%\n", id, item.toString(), getDescription(),
        item.getPrice()/2, getPrice(), getStock(), discount)); }

    private final LineItem item;
    private final double discount;
    private int id;
    private int stock;
}

```

File: src/cop4331/model/Bundle.java

```
package cop4331.model;

import java.util.*;

/**
 * A bundle of line items that is again a line item.
 * @author modified by Zachary
 */
public class Bundle implements LineItem {
    /**
     * Constructs a bundle with no items.
     * @param id
     * @param stock
     */
    public Bundle(int id, int stock) {
        this.items = new ArrayList<>();
        this.id = id;
        this.stock = stock;
    }

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

    @Override
    public int getID() { return id; }

    public int setID(int id) { return this.id = id; }

    @Override
    public double getPrice()
    {
        double price = 0;

        for (LineItem item : items)
            price += item.getPrice();
        return price;
    }
}
```

```

@Override
public String toString()
{
    String name = "Bundle of ";
    for (int i = 0; i < items.size(); i++) {
        if (i > 0) name += ", ";
        name += items.get(i).toString();
    }
    return name;
}

@Override
public String getName() {
    return "Bundle";
}

@Override
public String getDescription() {
    String description = "Bundle";
    return description;
}

@Override
public int getStock() {
    return stock;
}

@Override
public int setStock(int stock) { return this.stock = stock; }

@Override
public String getInfo() {
    String info = "Bundle of ";
    for (int i = 0; i < items.size(); i++) {
        if (i > 0) info += ", ";
        info += items.get(i).toString();
    }
    info += ".\n";
    info += String.format("Price: $%.2f\nQuantity:
%d\n", getPrice(), getStock());
    return info;
}

@Override
public String getSellerInfo() {

```



```
String info = "Bundle of ";
for (int i = 0; i < items.size(); i++)
{
    if (i > 0) info += ", ";
    info += items.get(i).toString();
}
info += ".\n";
info += String.format("Price: $%.2f\nQuantity:
%d\n",getPrice(),getStock());
return info;
}

private final ArrayList<LineItem> items;
private int id;
private int stock;
}
```

File: src/cop4331/model/Invoice.java

```
package cop4331.model;

import cop4331.formatter.InvoiceFormatter;

import java.util.*;

import javax.swing.JOptionPane;
import javax.swing.event.*;

/**
 * An invoice for a sale, consisting of line items.
 * @author modified by Rabih
 */
public class Invoice {
    /**
     * Constructs a blank invoice.
     */
    public Invoice() {
        items = new ArrayList<>();
        listeners = new ArrayList<>();
    }

    /**
     * Resets invoice.
     */
    public void reset() {
        items = new ArrayList<>();
        listeners = new ArrayList<>();

        // Notify all observers of the change to the invoice.
        ChangeEvent event = new ChangeEvent(this);
        for (ChangeListener listener : listeners)
            listener.stateChanged(event);
    }

    /**
     * Adds an item to the invoice.
     * @param item the item to add
     */
    public void addItem(LineItem item) {
        if (item.getStock() > 0) {
            items.add(item);
        }
    }
}
```

```

        item.setStock(item.getStock() - 1);
    }
    else {
        JOptionPane.showMessageDialog(null, item + " is out of stock.",
"Info", JOptionPane.INFORMATION_MESSAGE);
    }
    // Notify all observers of the change to the invoice.
    ChangeEvent event = new ChangeEvent(this);
    for (ChangeListener listener : listeners)
        listener.stateChanged(event);
}

/**
 * Removes an item to the invoice.
 * @param item the item to add
 */
public void removeItem(LineItem item) {
    if(items.remove(item)) {
        item.setStock(item.getStock() + 1);
    }
    // Notify all observers of the change to the invoice.
    ChangeEvent event = new ChangeEvent(this);
    for (ChangeListener listener : listeners)
        listener.stateChanged(event);
}

/**
 * Adds a change listener to the invoice.
 * @param listener the change listener to add
 */
public void addChangeListener(ChangeListener listener)
{
    listeners.add(listener);
}

/**
 * Gets an iterator that iterates through the items.
 * @return an iterator for the items
 */
public Iterator<LineItem> getItems()
{
    return new Iterator<LineItem>() {
        @Override
        public boolean hasNext() {
            return current < items.size();

```

```

    }

    @Override
    public LineItem next() {
        return items.get(current++);
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }

    private int current = 0;
};
}

/**
 * Formats invoice string.
 * @param formatter
 * @return formatted invoice string
 */
public String format(InvoiceFormatter formatter)
{
    String r = formatter.formatHeader();
    Iterator<LineItem> iter = getItems();
    while (iter.hasNext())
        r += formatter.formatLineItem(iter.next());
    return r + formatter.formatFooter();
}

/**
 * Clears invoice format.
 * @return empty string
 */
public String clearFormat()
{
    return "";
}

private ArrayList<LineItem> items;
private ArrayList<ChangeListener> listeners;
}

```

File: src/cop4331/model/SellerBundle.java

```
package cop4331.model;

import cop4331.formatter.SellerBundleFormatter;

import java.util.*;

import javax.swing.event.*;

/**
 * An item bundle consisting of line items.
 * @author Rabih
 */
public class SellerBundle {
    /**
     * Constructs a blank item bundle.
     */
    public SellerBundle() {
        items = new ArrayList<>();
        listeners = new ArrayList<>();
    }

    /**
     * Adds a new item to the bundle.
     * @param item the item to add
     */
    public void addItem(LineItem item) {
        items.add(item);

        // Notify all observers of the change to the invoice
        ChangeEvent event = new ChangeEvent(this);
        for (ChangeListener listener : listeners)
            listener.stateChanged(event);
    }

    /**
     * Adds a change listener to the item bundle.
     * @param listener the change listener to add
     */
    public void addChangeListener(ChangeListener listener) {
        listeners.add(listener);
    }
}
```

```

/**
 * Gets an iterator that iterates through the items.
 * @return an iterator for the items
 */
public Iterator<LineItem> getItems() {
    return new
        Iterator<LineItem>() {
            @Override
            public boolean hasNext() {
                return current < items.size();
            }

            @Override
            public LineItem next() {
                return items.get(current++);
            }

            @Override
            public void remove() {
                throw new UnsupportedOperationException();
            }

            private int current = 0;
        };
}

/**
 * Formats item bundle string.
 * @param formatter
 * @return formatted item bundle string
 */
public String format(SellerBundleFormatter formatter)
{
    String r = formatter.formatHeader();
    Iterator<LineItem> iter = getItems();
    while (iter.hasNext())
        r += formatter.formatLineItem(iter.next());
    return r + formatter.formatFooter();
}

private final ArrayList<LineItem> items;
private final ArrayList<ChangeListener> listeners;
}

```

File: src/cop4331/controller/UserController.java

```
package cop4331.controller;

import cop4331.gui.LoginView;
import cop4331.gui.SignupView;

import cop4331.application.User;
import cop4331.application.Inventory;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * User view pages controller.
 * @author Gerrell
 */
public class UserController {
    private final Inventory data;
    private final SignupView signup;
    private final LoginView login;

    /**
     * Constructor.
     */
    public UserController() {
        this.login = new LoginView();
        this.signup = new SignupView();
        this.data = new Inventory();
    }

    /**
     * Initializes LoginView and adds listeners for buttons.
     */
    public void initLogin() {
        login.view();

        login.getSignupButton().addActionListener(e -> {
            login.getFrame().dispose();
            initSignup();
        });
        login.getLoginButton().addActionListener(new VerifyListenerLogin());
    }
}
```

```

/**
 * Initializes SignupView and adds listeners for buttons.
 */
public void initSignup() {
    signup.view();

    signup.getBackButton().addActionListener(a -> {
        signup.getFrame().dispose();
        initLogin();
    });
    signup.getNewAccountButton().addActionListener(new
VerifyListenerSignup());
}

/**
 * Listener object that implements action listener and sets the label's
 * text to show whether the login data is valid.
 */
public class VerifyListenerLogin implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {
        String name = login.getUserNameField().getText();
        String pwd = login.getPWDField().getText();

        if (name.isEmpty()) {
            login.getValidLabel().setText("Empty username field.");
        }
        else if (pwd.isEmpty()) {
            login.getValidLabel().setText("Empty password field.");
        }
        else if (!User.getInstance().validateUserLogin(name,pwd)){
            login.getValidLabel().setText("<html>Incorrect username<br>or
password</html>");
        }
        else {
            login.getFrame().dispose();
            if("customer".equals(login.getButtonGroup().getSelection().getAct
ionCommand())) {
                data.dataInventoryInit();
                data.dataIDInit();
                CustomerController shop = new
CustomerController(data.getDefaultListModel());
                shop.initController();
            }
        }
    }
}

```



```

        else
if("seller".equals(login.getButtonGroup().getSelection().getActionCommand())) {
    data.dataInventoryInit();
    data.dataIDInit();
    SellerController shop = new
SellerController(data.getDefaultListModel());
    shop.initController();
    }
    }
}

/**
 * Listener object that implements action listener and sets the label's
 * text to show whether the signup data is valid.
 */
public class VerifyListenerSignup implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {
        String name = signup.getUserNameField().getText();
        String pwd = signup.getPWDField().getText();
        String pwdConfirm = signup.getPWDConfirmField().getText();

        if (name.isEmpty()) {
            signup.getValidLabel().setText("Empty username field.");
        }
        else if (!User.getInstance().checkNewUser(name)) {
            signup.getValidLabel().setText("This name is already in Use.");
        }
        else if (pwd.isEmpty()) {
            signup.getValidLabel().setText("Empty password field.");
        }
        else if (!pwdConfirm.equals(pwd)){
            signup.getValidLabel().setText("Passwords don't match.");
        }
        else {
            User.getInstance().addNewUser(name,pwd);
            User.getInstance().updateUserData();
            signup.getFrame().dispose();
            initLogin();
        }
    }
}
}
}

```

File: src/cop4331/controller/CustomerController.java

```
package cop4331.controller;

import cop4331.gui.CardView;
import cop4331.gui.CustomerView;

import cop4331.model.Invoice;
import cop4331.model.LineItem;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.JOptionPane;

/**
 * Customer view page controller.
 * @author Gerrell
 */
public class CustomerController {
    private final CustomerView customer;
    private final Invoice invoice;
    private CardView cardview;

    /**
     * Constructor.
     * @param dlm
     */
    public CustomerController(DefaultListModel dlm){
        this.invoice = new Invoice();
        this.customer = new CustomerView(dlm);
    }

    /**
     * Initializes CustomerView and adds listeners for buttons, text area,
     * and other UI elements.
     */
    public void initController() {
        customer.view();

        // Changelister to update textarea with invoice data.
    }
}
```

```

        ChangeListener listener = e ->
customer.getTextArea().setText(invoice.format(customer.getFormatter()));

invoice.addChangeListener(listener);

customer.getAdd().addActionListener(e -> {
    LineItem item = (LineItem) customer.getList().getSelectedValue();
    invoice.addItem(item);
});

customer.getRemove().addActionListener(e -> {
    LineItem item = (LineItem) customer.getList().getSelectedValue();
    invoice.removeItem(item);
});

customer.getInfo().addActionListener(e -> {
    LineItem item = (LineItem) customer.getList().getSelectedValue();
    JOptionPane.showMessageDialog(null, item.getInfo(), "Info",
JOptionPane.INFORMATION_MESSAGE);
});

cardview = new CardView(invoice);

customer.getCheckout().addActionListener(e -> {
new CardController(cardview,invoice);
    cardview.getFrame().addWindowListener(new WindowAdapter() {
@Override
public void windowClosed(WindowEvent e) {
        invoice.reset();
        customer.getTextArea().setText(invoice.clearFormat());
        invoice.addChangeListener(listener);
    }
});
});

customer.getLogout().addActionListener(e -> {
    customer.getFrame().dispose();
    new UserController().initLogin();
});

    }
}

```

File: src/cop4331/controller/SellerController.java

```
package cop4331.controller;

import cop4331.model.LineItem;
import cop4331.model.SellerBundle;

import cop4331.gui.ProfitView;
import cop4331.gui.SellerBundleView;
import cop4331.gui.SellerNewView;
import cop4331.gui.SellerView;

import cop4331.application.Profit;
import cop4331.application.Inventory;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.DefaultListModel;
import javax.swing.event.ChangeListener;

/**
 * Seller view pages controller.
 * @author Rabih
 */
public class SellerController
{
    private final SellerView sellerView;
    private final SellerNewView sellerNew;
    private final SellerBundleView sellerBundle;

    private final Inventory data;
    private final DefaultListModel dlm;
    private final SellerBundle sb;

    /**
     * Constructor.
     * @param dlm
     */
    public SellerController(DefaultListModel dlm){
        this.sellerView = new SellerView(dlm);
    }
}
```

```

        this.sellerNew = new SellerNewView();
        this.sellerBundle = new SellerBundleView();
        this.data = new Inventory();
        this.dlm = dlm;
        this.sb = new SellerBundle();
    }

    /**
     * Initializes SellerView and adds listeners for buttons, text area,
     * and other UI elements.
     */
    public void initController() {
        sellerView.view();

        ChangeListener listener = e ->
sellerBundle.getBundleArea().setText(sb.format(sellerBundle.getFormatter()));

        sb.addChangeListener(listener);

        // MouseListener for highlighted element in JList.
        MouseListener mouseListener = new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() == 1) {
                    if (sellerView.getList().getSelectedValue() == null) {
                        sellerView.getList();
                    }
                    ListItem item = (ListItem)
sellerView.getList().getSelectedValue();
                    sellerView.getTextArea().setText(item.getSellerInfo());

                }
            }
        };
        sellerView.getList().addMouseListener(mouseListener);

        sellerView.getNew().addActionListener(e -> {
            sellerView.getTextArea().setText(" ");
            sellerView.getTextArea().setText("");
            sellerNew.view();
            sellerNew.getNewButton().addActionListener(new
VerifyListenerNew());
            sellerNew.getBackButton().addActionListener(a ->
sellerNew.getFrame().dispose());

```

```

    });

    sellerView.getRemove().addActionListener(e -> {
        sellerView.getTextArea().setText(" ");
        sellerView.getTextArea().setText("");
        LineItem item = (LineItem) sellerView.getList().getSelectedValue();
        sellerView.getFrame().dispose();
        data.dataSellerRemoveItem(item);
        data.init();
        SellerController shop = new
SellerController(data.getDefaultListModel());
        shop.initController();
    });

    // Initialize profit view.
    sellerView.getProfit().addActionListener(e -> {
        new ProfitView(Profit.getInstance());
    });

    sellerView.getBundle().addActionListener(e -> {

        sellerView.getTextArea().setText(" ");
        sellerView.getTextArea().setText("");
        sellerBundle.view(dlm);
        sellerBundle.getBundleButton().addActionListener(new
VerifyListenerBundle());
        sellerBundle.getBackButton().addActionListener(a ->
sellerBundle.getFrame().dispose());
        sellerBundle.getAddButton().addActionListener(a -> {
            LineItem item = (LineItem)
sellerBundle.getComboBox().getSelectedItem();
            sb.addItem(item);
        });
    });

    sellerView.getLogOut().addActionListener(e -> {
        sellerView.getFrame().dispose();
        new UserController().initLogin();
    });
}

/**
 * Checks if new item name is "Bundle".
 * @param n
 * @return true if n is "Bundle", false otherwise.

```

```

    */
    public boolean isBundle(String n) {
        return "Bundle".equals(n);
    }

    /**
     * Validates if String is a double.
     * @param n
     * @return false if n has a double format, true otherwise.
     */
    public boolean isValidDouble(String n) {
        return !n.matches("(\\d+\\.?\\d+)");
    }

    /**
     * Validates if String is a integer.
     * @param n
     * @return false if n has an integer format, true otherwise.
     */
    public boolean isValidInteger(String n) {
        return !n.matches("(0|[1-9]\\d*)");
    }

    /**
     * Validates if String is a percentage.
     * @param n
     * @return false if n has a double format and is between
     * 0 and 100 (excluding 100), true otherwise.
     */
    public boolean isValidDiscountNumber(String n) {
        return !n.matches("^(([0-9][0-9]?(\\. [0-9]+)?))$");
    }

    /**
     * Listener object that implements action listener and sets the label's
     * text to show whether the new item data is valid.
     */
    public class VerifyListenerNew implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent event) {
            String name = sellerNew.getNameField().getText();
            String description = sellerNew.getDescriptionArea().getText();
            String price = sellerNew.getPriceField().getText();
            String quantity = sellerNew.getQuantityField().getText();
            String discount = sellerNew.getDiscountField().getText();

```

```

        if (name.isEmpty()) {
            sellerNew.getValidLabel().setText("Empty name field.");
        }
        else if (isBundle(name)) {
            sellerNew.getValidLabel().setText("Click Bundle to add new
Bundle.");
        }
        else if (description.isEmpty()) {
            sellerNew.getValidLabel().setText("Empty description field.");
        }
        else if (isBundle(description)) {
            sellerNew.getValidLabel().setText("Click Bundle to add new
Bundle.");
        }
        else if (isValidDouble(price)) {
            sellerNew.getValidLabel().setText("Price must be a number.");
        }
        else if (isValidInteger(quantity)) {
            sellerNew.getValidLabel().setText("Quantity must be a integer.");
        }
        else if (isValidDiscountNumber(discount)) {
            sellerNew.getValidLabel().setText("Discount must be 0-99.");
        }
        else {
            sellerView.getFrame().dispose();
            sellerNew.getFrame().dispose();

            data.dataSellerNewItem(name, description, quantity, price,
discount);

            data.init();
            SellerController shop = new
SellerController(data.getDefaultListModel());
            shop.initController();
        }
    }
}

/**
 * Listener object that implements action listener and sets the label's
 * text to show whether new Bundle data is valid.
 */
public class VerifyListenerBundle implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {

```



```

String bundle = sellerBundle.getBundleArea().getText();
String quantity = sellerBundle.getQuantityField().getText();
String discount = sellerBundle.getDiscountField().getText();

if (bundle.isEmpty()) {
    sellerBundle.getValidLabel().setText("Add items to bundle.");
}
else if (isValidInteger(quantity)) {
    sellerBundle.getValidLabel().setText("Quantity must be a
integer.");
}
else if (isValidDiscountNumber(discount)) {
    sellerBundle.getValidLabel().setText("Discount must be 0-99.");
}
else {
    sellerView.getFrame().dispose();
    sellerBundle.getFrame().dispose();

    data.dataSellerBundleItem(sb, quantity, discount);
    data.init();
    SellerController shop = new
SellerController(data.getDefaultListModel());
    shop.initController();
}
}
}
}

```

File: src/cop4331/controller/CardController.java

```
package cop4331.controller;

import cop4331.gui.CardView;

import cop4331.model.Invoice;

import cop4331.application.Profit;
import cop4331.application.Inventory;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JOptionPane;

/**
 * Credit Card payment page controller.
 * @author Solan
 */
public class CardController {
    private Inventory data;
    private final CardView cardview;
    private final Invoice invoice;

    /**
     * Constructor.
     * @param cv
     * @param invoice
     */
    public CardController(CardView cv, Invoice invoice) {
        this.cardview = cv;
        this.invoice = invoice;
        initCardController();
    }

    /**
     * Initializes CardView and add action listeners for buttons.
     */
    private void initCardController() {
        cardview.view();
        cardview.getPayButton().addActionListener(new VerifyListener());
        cardview.getBackButton().addActionListener(e ->
cardview.getFrame().hide());
    }
}
```

```

}

/**
 * Validates Credit Card number using regular expressions.
 * @param number
 * @return true if Credit Card number is in the correct format,
 * false otherwise.
 */
public boolean isValidCreditCardNumber(String number) {
    return number.matches("(^(?:(<visa>4[0-9]{12}(?:[0-9]{3})?)|))$");
}

/**
 * Checks if Credit Card name is empty.
 * @param name
 * @return true if Credit Card name is empty, false otherwise.
 */
public boolean isValidCreditCardName(String name) {
    return name.isEmpty();
}

/**
 * Validates Credit Card date using regular expressions.
 * @param date
 * @return true if Credit Card date is in the correct format,
 * false otherwise.
 */
public boolean isValidCreditCardDate(String date) {
    return date.matches("(^(0[1-9]|1[0-2])\\/(?[0-9]{2})$)");
}

/**
 * Validates Credit Card cvv number using regular expressions.
 * @param cvv
 * @return true if Credit Card cvv number is in the correct format,
 * false otherwise.
 */
public boolean isValidCreditCardCVV(String cvv) {
    return cvv.matches("[0-9]{3}$");
}

/**
 * Listener object that implements action listener and sets the label's
 * text to show whether the credit card data is valid or not.
 */

```

```

public class VerifyListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent event) {
        String number = cardview.getNumberField().getText();
        String name = cardview.getNameField().getText();
        String date = cardview.getDateField().getText();
        String cvv = cardview.getCVVField().getText();

        if (!isValidCreditCardNumber(number)) {
            cardview.getValidLabel().setText("<html>Invalid number. Must
start<br>with 4 and is 16 digits long.</html>");
        }
        else if (isValidCreditCardName(name)) {
            cardview.getValidLabel().setText("Empty name field.");
        }
        else if (!isValidCreditCardDate(date)) {
            cardview.getValidLabel().setText("<html>Date must be in<br>the
form mm/yy.</html>");
        }
        else if (!isValidCreditCardCVV(cvv)) {
            cardview.getValidLabel().setText("<html>Invalid cvv.<br>must be 3
digits long.</html>");
        }
        else {
            cardview.getFrame().dispose();
            data = new Inventory();
            data.dataUpdateStock(invoice);
            Profit.getInstance().addRevenue(cardview);

            JOptionPane.showMessageDialog(null, "Thank You For Purchasing!",
"Purchase Complete", JOptionPane.INFORMATION_MESSAGE);
        }
    }
}

```

File: src/cop4331/gui/LoginView.java

```
package cop4331.gui;

import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

/**
 * Login UI view.
 * @author Rabih
 */
public class LoginView{
    private JFrame frame;
    private JRadioButton customer, seller;
    private JTextField userName;
    private JPasswordField pwd;
    private JPanel panel;
    private JLabel userNameLabel, pwdLabel, validLabel;
    private JButton signupButton,loginButton;
    private ButtonGroup bg;

    /**
     * Constructor.
     */
    public LoginView() { }

    /**
     * Sets up all UI components.
     */
    public void view() {
        // set up components
        panel = new JPanel();
        customer = new JRadioButton("customer");
        customer.setActionCommand("customer");
        seller = new JRadioButton("seller");
    }
}
```

```

seller.setActionCommand("seller");
bg = new ButtonGroup();
bg.add(customer);
bg.add(seller);
customer.setSelected(true);
userName = new JTextField(17);
pwd = new JPasswordField(17);
userNameLabel = new JLabel("UserName");
pwdLabel = new JLabel("Password");
signupButton = new JButton("Create Account");
loginButton = new JButton("Login");
validLabel = new JLabel("");
panel.add(customer);
panel.add(seller);
frame = new JFrame("Login");

frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
frame.setSize(new Dimension(230, 240));
frame.setResizable(false);
frame.setLayout(new FlowLayout());
frame.add(panel);
frame.add(userNameLabel);
frame.add(userName);
frame.add(pwdLabel);
frame.add(pwd);
frame.add(signupButton);
frame.add(loginButton);
frame.add(validLabel);
frame.setLocationRelativeTo(null);
frame.setVisible(true);
}

public JFrame getFrame() { return frame; }
public ButtonGroup getButtonGroup() { return bg; }
public JButton getSignupButton() { return signupButton; }
public JButton getLoginButton() { return loginButton; }
public JTextField getUserTextField() { return userName; }
public JPasswordField getPWDField() { return pwd; }
public JLabel getValidLabel() { return validLabel; }
}

```

File: src/cop4331/gui/SignupView.java

```
package cop4331.gui;

import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

/**
 * Sign up UI view.
 * @author RabiH
 */
public class SignupView{
    private JFrame frame;
    private JTextField userName;
    private JPasswordField pwd, pwdConfirm;
    private JLabel userNameLabel, pwdLabel, pwdConfirmLabel, validLabel;
    private JButton backButton,newAccountButton;

    /**
     * Constructor.
     */
    public SignupView() { }

    /**
     * Sets up all UI components.
     */
    public void view() {
        // set up components
        userName = new JTextField(25);
        pwd = new JPasswordField(25);
        pwdConfirm = new JPasswordField(25);
        userNameLabel = new JLabel("Choose UserName");
        pwdLabel = new JLabel("Choose Password");
        pwdConfirmLabel = new JLabel("Confirm Password");
        backButton = new JButton("Back to Login");
        newAccountButton = new JButton("Create Account");
        validLabel = new JLabel("");
        frame = new JFrame("Signup");
    }
}
```

```

        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(new Dimension(300, 250));
        frame.setResizable(false);
        frame.setLayout(new FlowLayout());
        frame.add(userNameLabel);
        frame.add(userName);
        frame.add(pwdLabel);
        frame.add(pwd);
        frame.add(pwdConfirmLabel);
        frame.add(pwdConfirm);
        frame.add(backButton);
        frame.add(newAccountButton);
        frame.add(validLabel);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public JFrame getFrame() { return frame; }
    public JButton getNewAccountButton() { return newAccountButton; }
    public JButton getBackButton() { return backButton; }
    public JTextField getUserNameField() { return userName; }
    public JTextField getPWDField() { return pwd; }
    public JTextField getPWDConfirmField() { return pwdConfirm; }
    public JLabel getValidLabel() { return validLabel; }
}

```


File: src/cop4331/gui/CustomerView.java

```
package cop4331.gui;

import cop4331.formatter.InvoiceFormatter;
import cop4331.formatter.SimpleFormatter;

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ListSelectionModel;

/**
 * Customer UI view.
 * @author Rabih
 */
public class CustomerView {

    private final DefaultListModel plist;
    private InvoiceFormatter formatter;
    private JTextArea textArea;
    private JPanel panel, panelBtn;
    private JList list;
    private JScrollPane listScrollPane;
    private JFrame frame;
    private JButton addButton;
    private JButton removeButton;
    private JButton infoButton;
    private JButton checkoutButton;
    private JButton logout;

    /**
     * Constructor.
     * @param plist
     */
    public CustomerView(DefaultListModel plist) {
        this.plist = plist;
    }
}
```

```

}

/**
 * Sets up all UI components.
 */
public void view() {
    formatter = new SimpleFormatter();
    textArea = new JTextArea(20, 40);
    textArea.setEditable(false);
    list = new JList(plist);
    listScrollPane = new JScrollPane(list);
    addButton = new JButton("Add");
    removeButton = new JButton("Remove");
    infoButton = new JButton("Info");
    checkoutButton = new JButton("Pay");
    logout = new JButton("Log Out");

    // Put the combo box and the add button into a panel
    panel = new JPanel();
    panelBtn = new JPanel();

    // Create the list and put it in a scroll pane.
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    list.setSelectedIndex(0);
    list.setVisibleRowCount(10);

    listScrollPane.setPreferredSize(new Dimension(360, 240));
    panel.add(listScrollPane, BorderLayout.CENTER);

    panelBtn.add(addButton);
    panelBtn.add(removeButton);
    panelBtn.add(infoButton);
    panelBtn.add(checkoutButton);
    panelBtn.add(logout);

    // Add the text area and panel to the frame
    frame = new JFrame("JAVA Shopping App");
    frame.add(new JScrollPane(textArea), BorderLayout.NORTH);
    frame.add(new JScrollPane(panel), BorderLayout.CENTER);
    frame.add(panel, BorderLayout.CENTER);
    frame.add(panelBtn, BorderLayout.SOUTH);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}

```

```
}

public InvoiceFormatter getFormatter() { return formatter; }
public JTextArea getTextArea() { return textArea; }
public JPanel getPanel() { return panel; }
public JList getList() { return list; }
public JFrame getFrame() { return frame; }
public JButton getAdd() { return addButton; }
public JButton getRemove() { return removeButton; }
public JButton getInfo() { return infoButton; }
public JButton getCheckout() { return checkoutButton; }
public JButton getLogOut() { return logout; }
}
```

File: src/cop4331/gui/CardView.java

```
package cop4331.gui;

import cop4331.model.Invoice;

import cop4331.formatter.InvoiceFormatter;
import cop4331.formatter.SimpleFormatter;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

/**
 * Credit Card UI view.
 * @author Solan
 */
public class CardView {

    private final Invoice invoice;
    private JFrame frame;
    private InvoiceFormatter formatter;
    private JTextArea checkoutCart;
    private JTextField numberField;
    private JLabel numberLabel;
    private JTextField nameField;
    private JLabel nameLabel;
    private JTextField dateField;
    private JLabel dateLabel;
    private JTextField cvvField;
    private JLabel cvvLabel;
    private JLabel validLabel;
    private JButton backButton;
    private JButton payButton;

    /**
     * Constructor.

```

```

    * @param invoice
    */
    public CardView(Invoice invoice) {
        this.invoice = invoice;
    }

    /**
     * Sets up all UI components.
     */
    public void view() {
        // set up components
        formatter = new SimpleFormatter();
        checkoutCart = new JTextArea(10, 16);
        checkoutCart.setEditable(false);
        checkoutCart.setText(this.invoice.format(formatter));

        numberField = new JTextField(17);
        numberLabel = new JLabel("Card Number:");
        nameField = new JTextField(17);
        nameLabel = new JLabel("Card Name:");
        dateField = new JTextField(5);
        dateLabel = new JLabel("Date:");
        cvvField = new JTextField(5);
        cvvLabel = new JLabel("cvv:");

        validLabel = new JLabel("");
        backButton = new JButton("Back");
        payButton = new JButton("Pay");

        frame = new JFrame("Cart Checkout");
        frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        frame.setSize(new Dimension(210, 420));
        frame.setResizable(false);
        frame.setLayout(new FlowLayout());
        frame.add(new JScrollPane(checkoutCart), BorderLayout.CENTER);
        frame.add(numberLabel);
        frame.add(numberField);
        frame.add(nameLabel);
        frame.add(nameField);
        frame.add(dateLabel);
        frame.add(dateField);
        frame.add(cvvLabel);
        frame.add(cvvField);
        frame.add(backButton);
        frame.add(payButton);
    }

```

```
        frame.add(validLabel);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }

    public InvoiceFormatter getFormatter() { return formatter; }
    public JFrame getFrame() { return frame; }
    public JButton getPayButton() { return payButton; }
    public JButton getBackButton() { return backButton; }
    public JTextField getNumberField() { return numberField; }
    public JTextField getNameField() { return nameField; }
    public JTextField getDateField() { return dateField; }
    public JTextField getCVVField() { return cvvField; }
    public JLabel getValidLabel() { return validLabel; }
}
```

File: src/cop4331/gui/SellerView.java

```
package cop4331.gui;

import cop4331.formatter.InvoiceFormatter;
import cop4331.formatter.SimpleFormatter;

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ListSelectionModel;

/**
 * Seller UI view.
 * @author Rabih
 */
public class SellerView {
    private DefaultListModel plist;
    private InvoiceFormatter formatter;
    private JTextArea textArea;
    private JPanel panel, panelBtn;
    private JList list;
    private JScrollPane listScrollPane;
    private JFrame frame;
    private JButton newButton;
    private JButton removeButton;
    private JButton profitButton;
    private JButton bundleButton;
    private JButton logout;

    /**
     * Constructor.
     * @param plist
     */
    public SellerView(DefaultListModel plist) {
        this.plist = plist;
    }
}
```

```

/**
 * Sets up all UI components.
 */
public void view() {
    formatter = new SimpleFormatter();
    textArea = new JTextArea(10, 40);
    textArea.setEditable(false);
    textArea.setLineWrap(true);
    textArea.setWrapStyleWord(true);
    list = new JList(plist);
    listScrollPane = new JScrollPane(list);
    newButton = new JButton("New");
    removeButton = new JButton("Remove");
    bundleButton = new JButton("Bundle");
    profitButton = new JButton("Profit");
    logout = new JButton("Log Out");

    // Put the combo box and the add button into a panel
    panel = new JPanel();
    panelBtn = new JPanel();

    // Create the list and put it in a scroll pane.
    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    list.setSelectedIndex(0);
    list.setVisibleRowCount(10);
    listScrollPane.setPreferredSize(new Dimension(360, 240));

    panel.add(listScrollPane, BorderLayout.CENTER);
    panelBtn.add(newButton);
    panelBtn.add(bundleButton);
    panelBtn.add(removeButton);
    panelBtn.add(profitButton);
    panelBtn.add(logout);

    // Add the text area and panel to the frame
    frame = new JFrame("JAVA Shopping App");
    frame.add(new JScrollPane(textArea), BorderLayout.NORTH);
    frame.add(panel, BorderLayout.CENTER);
    frame.add(panelBtn, BorderLayout.SOUTH);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}

```



```
public InvoiceFormatter getFormatter() { return formatter; }
public JTextArea getTextArea() { return textArea; }
public JPanel getPanel() { return panel; }
public JList getList() { return list; }
public JFrame getFrame() { return frame; }
public JButton getNew() { return newButton; }
public JButton getRemove() { return removeButton; }
public JButton getProfit() { return profitButton; }
public JButton getBundle() { return bundleButton; }
public JButton getLogOut() { return logout; }
}
```

File: src/cop4331/gui/SellerNewView.java

```
package cop4331.gui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.Border;

/**
 * Seller new item UI view.
 * @author Rabih
 */
public class SellerNewView {

    private JFrame frame;
    private JTextField nameField;
    private JLabel nameLabel;
    private JTextArea descriptionArea;
    private JScrollPane descriptionPane;
    private JLabel descriptionLabel;
    private JTextField priceField;
    private JLabel priceLabel;
    private JTextField quantityField;
    private JLabel quantityLabel;
    private JTextField discountField;
    private JLabel discountLabel;
    private JLabel validLabel;
    private JButton backButton;
    private JButton newButton;
    Border border = BorderFactory.createLineBorder(Color.GRAY);

    /**
     * Constructor.
     */
    public SellerNewView() {}
}
```

```

/**
 * Sets up all UI components.
 */
public void view() {
    // set up components
    nameField = new JTextField(17);
    nameLabel = new JLabel("Item Name:");
    descriptionArea = new JTextArea(5, 17);
    descriptionArea.setBorder(BorderFactory.createCompoundBorder(border,
        BorderFactory.createEmptyBorder(0, 0, 0, 0)));
    descriptionArea.setLineWrap(true);
    descriptionArea.setWrapStyleWord(true);
    descriptionPane = new JScrollPane(descriptionArea);
    descriptionLabel = new JLabel("Item Description:");
    priceField = new JTextField(17);
    priceLabel = new JLabel("Item Price:");
    quantityField = new JTextField(17);
    quantityLabel = new JLabel("Item Quantity:");
    discountField = new JTextField(17);
    discountLabel = new JLabel("Discount:");

    validLabel = new JLabel("");
    backButton = new JButton("Back");
    newButton = new JButton("New");

    frame = new JFrame("Add New Item");
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.setSize(new Dimension(210, 400));
    frame.setResizable(false);
    frame.setLayout(new FlowLayout());
    frame.add(nameLabel);
    frame.add(nameField);
    frame.add(descriptionLabel);
    frame.add(descriptionPane);
    frame.add(priceLabel);
    frame.add(priceField);
    frame.add(quantityLabel);
    frame.add(quantityField);
    frame.add(discountLabel);
    frame.add(discountField);
    frame.add(backButton);
    frame.add(newButton);
    frame.add(validLabel);
    frame.setLocationRelativeTo(null);

```

```
        frame.setVisible(true);
    }

    public JFrame getFrame() { return frame; }
    public JButton getNewButton() { return newButton; }
    public JButton getBackButton() { return backButton; }
    public JTextField getNameField() { return nameField; }
    public JTextArea getDescriptionArea() { return descriptionArea; }
    public JTextField getPriceField() { return priceField; }
    public JTextField getQuantityField() { return quantityField; }
    public JTextField getDiscountField() { return discountField; }
    public JLabel getValidLabel() { return validLabel; }
}
```

File: src/cop4331/gui/SellerBundleView.java

```
package cop4331.gui;

import cop4331.model.LineItem;

import cop4331.formatter.SellerBundleFormatter;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.BorderFactory;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.Border;

/**
 * Seller new bundle UI view.
 * @author Rabih
 */
public class SellerBundleView {

    private JFrame frame;
    private SellerBundleFormatter formatter;
    private JComboBox combo;
    private JTextArea bundleArea;
    private JScrollPane bundlePane;
    private JLabel bundleLabel;
    private JTextField quantityField;
    private JLabel quantityLabel;
    private JTextField discountField;
    private JLabel discountLabel;
    private JLabel validLabel;
    private JButton backButton;
    private JButton addButton;
    private JButton bundleButton;
    Border border = BorderFactory.createLineBorder(Color.GRAY);
```

```

/**
 * Constructor.
 */
public SellerBundleView() {}

/**
 * Sets up all UI components.
 * @param plist
 */
public void view(DefaultListModel plist) {
    // set up components
    formatter = new SellerBundleFormatter();
    combo = new JComboBox();
    LineItem item;
    for(int i = 0; i< plist.getSize();i++){
        item = (LineItem) plist.getElementAt(i);
        if (!"Bundle".equals(item.getDescription())) {
            combo.addItem(item);
        }
    }

    combo.setPreferredSize(new Dimension(160, 20));
    bundleArea = new JTextArea(7, 17);
    bundleArea.setLineWrap(true);
    bundleArea.setWrapStyleWord(true);
    bundleArea.setEditable(false);
    bundleArea.setBorder(BorderFactory.createCompoundBorder(border,
        BorderFactory.createEmptyBorder(0, 0, 0, 0)));
    bundlePane = new JScrollPane(bundleArea);
    bundleLabel = new JLabel("Items to Bundle:");
    quantityField = new JTextField(17);
    quantityLabel = new JLabel("Item Quantity:");
    discountField = new JTextField(17);
    discountLabel = new JLabel("Discount:");
    validLabel = new JLabel("");
    backButton = new JButton("Back");
    addButton = new JButton("Add");
    bundleButton = new JButton("Bundle");

    frame = new JFrame("Add New Bundle");
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.setSize(new Dimension(210, 400));
    frame.setResizable(false);
    frame.setLayout(new FlowLayout());

```

```
    frame.add(bundleLabel);
    frame.add(combo);
    frame.add(bundlePane);
    frame.add(quantityLabel);
    frame.add(quantityField);
    frame.add(discountLabel);
    frame.add(discountField);
    frame.add(backButton);
    frame.add(addButton);
    frame.add(bundleButton);
    frame.add(validLabel);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
```

```
public JFrame getFrame() { return frame; }
public SellerBundleFormatter getFormatter() { return formatter; }
public JComboBox getComboBox() { return combo; }
public JButton getAddButton() { return addButton; }
public JButton getBundleButton() { return bundleButton; }
public JButton getBackButton() { return backButton; }
public JTextArea getBundleArea() { return bundleArea; }
public JTextField getQuantityField() { return quantityField; }
public JTextField getDiscountField() { return discountField; }
public JLabel getValidLabel() { return validLabel; }
}
```

File: test/cop4331/gui/ProfitView.java

```
package cop4331.gui;

import cop4331.formatter.ProfitFormatter;
import cop4331.application.Profit;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

/**
 * Profit UI view.
 * @author Rabih
 */
public class ProfitView {
    private final JFrame frame;
    private final JTextArea profitText;
    private final ProfitFormatter formatter;

    /**
     * Constructor sets up all UI components.
     * @param profit
     */
    public ProfitView(Profit profit) {
        // set up components
        formatter = new ProfitFormatter();
        profitText = new JTextArea(10, 17);
        profitText.setText(profit.format(formatter));
        profitText.setEditable(false);

        frame = new JFrame("Profit Page");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(new Dimension(210, 210));
        frame.setResizable(false);
        frame.setLayout(new FlowLayout());
        frame.add(new JScrollPane(profitText), BorderLayout.CENTER);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```



```
    }  
  
    public JFrame getFrame() { return frame; }  
}
```

File: src/cop4331/formatter/InvoiceFormatter.java

```
package cop4331.formatter;

/**
 * This interface describes the tasks that an invoice
 * formatter needs to carry out.
 * @param <T>
 * @author Modified by Zachary
 */
public interface InvoiceFormatter<T>
{
    /**
     * Formats the header of the invoice.
     * @return the invoice header
     */
    String formatHeader();

    /**
     * Formats a line item of the invoice.
     * @param item
     * @return the formatted line item
     */
    String formatLineItem(T item);

    /**
     * Formats the footer of the invoice.
     * @return the invoice footer
     */
    String formatFooter();

    /**
     * Get the total revenue
     * @return the invoice footer
     */
    double getTotal();
}
```

File: src/cop4331/formatter/SimpleFormatter.java

```
package cop4331.formatter;

import cop4331.model.LineItem;

/**
 * A simple invoice formatter that implements InvoiceFormatter.
 * @author Modified by Zachary
 */
public class SimpleFormatter implements InvoiceFormatter<LineItem>
{
    @Override
    public String formatHeader()
    {
        total = 0;
        return "                I N V O I C E\n\n\n";
    }

    @Override
    public String formatLineItem(LineItem item)
    {
        total += item.getPrice();
        return (String.format(
            "%s: $%.2f\n", item, item.getPrice()));
    }

    @Override
    public String formatFooter()
    {
        return (String.format("\n\nTOTAL DUE: $%.2f\n", total));
    }

    @Override
    public double getTotal() {return total;}

    private double total;
}
```

File: src/cop4331/formatter/ProfitFormatter.java

```
package cop4331.formatter;

import cop4331.application.Profit;

/**
 * A profit formatter that implements InvoiceFormatter.
 * @author Rabih
 */
public class ProfitFormatter implements InvoiceFormatter<Profit>
{
    @Override
    public String formatHeader()
    {
        return "          JAVA Shopping App\n\n\n";
    }

    @Override
    public String formatLineItem(Profit profit)
    {
        profitValue = profit.getProfit();
        return (String.format(
            "Revenue: $%.2f\nCosts:
$%.2f\n",profit.getRevenue(),profit.getCosts()));

    }

    @Override
    public String formatFooter()
    {
        return (String.format("\n\nProfit: $%.2f\n", profitValue));
    }

    @Override
    public double getTotal() {return profitValue;}

    private double profitValue = 0;
}
```

File: src/cop4331/formatter/SellerBundleFormatter.java

```
package cop4331.formatter;

import cop4331.model.LineItem;

/**
 * A Seller bundle formatter that implements InvoiceFormatter.
 * @author Gerrell
 */
public class SellerBundleFormatter implements InvoiceFormatter<LineItem>
{
    @Override
    public String formatHeader()
    {
        total = 0;
        return String.format("\n");
    }

    @Override
    public String formatLineItem(LineItem item)
    {
        total += item.getPrice();
        return (String.format("%s\n", item));
    }

    @Override
    public String formatFooter()
    {
        return "";
    }

    @Override
    public double getTotal() {return total;}

    private double total;
}
```

File: test/cop4331/application/ProfitTest.java

```
package cop4331.application;

import cop4331.formatter.ProfitFormatter;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Rabih
 */
public class ProfitTest {
    /**
     * Test of format method, of class Profit.
     */
    @Test
    public void testFormat() {
        System.out.println("format");
        ProfitFormatter formatter = new ProfitFormatter();
        String expResult = String.format("          JAVA Shopping
App\n\n\nRevenue: $%.2f\nCosts: $%.2f\n\n\nProfit:
$%.2f\n", Profit.getInstance().getRevenue(), Profit.getInstance().getCosts(), Profit
.getInstance().getProfit());
        String result = Profit.getInstance().format(formatter);
        assertEquals(expResult, result);
    }

    /**
     * Test of getRevenue method, of class Profit.
     */
    @Test
    public void testGetRevenue() {
        System.out.println("getRevenue");
        double expResult = 0.0;
        double result = Profit.getInstance().getRevenue();
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of getCosts method, of class Profit.
     */
    @Test
    public void testGetCosts() {
        System.out.println("getCosts");
    }
}
```

```
        double expResult = Profit.getInstance().getCosts();
        double result = Profit.getInstance().getCosts();
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of getProfit method, of class Profit.
     */
    @Test
    public void testGetProfit() {
        System.out.println("getProfit");
        double expResult = Profit.getInstance().getRevenue() -
Profit.getInstance().getCosts();
        double result = Profit.getInstance().getProfit();
        assertEquals(expResult, result, 0);
    }
}
```

File: test/cop4331/application/UserTest.java

```
package cop4331.application;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author RabiH
 */
public class UserTest {
    /**
     * Test of checkNewUser method, of class User.
     */
    @Test
    public void testCheckNewUser() {
        System.out.println("checkNewUser");
        String name = "Tester";
        boolean expResult = true;
        boolean result = User.getInstance().checkNewUser(name);
        assertEquals(expResult, result);
    }

    /**
     * Test of validateUserLogin method, of class User.
     */
    @Test
    public void testValidateUserLogin() {
        System.out.println("validateUserLogin");
        String name = "RabiH";
        String pwd = "Khatib";
        boolean expResult = true;
        boolean result = User.getInstance().validateUserLogin(name, pwd);
        assertEquals(expResult, result);
    }
}
```


File: test/cop4331/controller/CardControllerTest.java

```
package cop4331.controller;

import cop4331.gui.CardView;
import cop4331.model.Invoice;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Solan
 */
public class CardControllerTest {
    /**
     * Test of isValidCreditCardNumber method, of class CardController.
     */
    @Test
    public void testIsValidCreditCardNumber() {
        System.out.println("isValidCreditCardNumber");
        String number = "4312123412341234";
        CardController instance = new CardController(new CardView(new
Invoice(),new Invoice()));
        boolean expResult = true;
        boolean result = instance.isValidCreditCardNumber(number);
        assertEquals(expResult, result);
    }

    /**
     * Test of isValidCreditCardName method, of class CardController.
     */
    @Test
    public void testIsValidCreditCardName() {
        System.out.println("isValidCreditCardName");
        String name = "";
        CardController instance = new CardController(new CardView(new
Invoice(),new Invoice()));
        boolean expResult = true;
        boolean result = instance.isValidCreditCardName(name);
        assertEquals(expResult, result);
    }

    /**
     * Test of isValidCreditCardDate method, of class CardController.
     */
}
```

```

    */
@Test
public void testIsValidCreditCardDate() {
    System.out.println("isValidCreditCardDate");
    String date = "11/24";
    CardController instance = new CardController(new CardView(new
Invoice()),new Invoice());
    boolean expResult = true;
    boolean result = instance.isValidCreditCardDate(date);
    assertEquals(expResult, result);
}

/**
 * Test of isValidCreditCardCVV method, of class CardController.
 */
@Test
public void testIsValidCreditCardCVV() {
    System.out.println("isValidCreditCardCVV");
    String cvv = "123";
    CardController instance = new CardController(new CardView(new
Invoice()),new Invoice());
    boolean expResult = true;
    boolean result = instance.isValidCreditCardCVV(cvv);
    assertEquals(expResult, result);
}
}

```

File: test/cop4331/controller/SellerControllerTest.java

```
package cop4331.controller;

import javax.swing.DefaultListModel;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Rabih
 */
public class SellerControllerTest {

    public SellerControllerTest() {

    }

    /**
     * Test of isBundle method, of class SellerController.
     */
    @Test
    public void testIsBundle() {
        System.out.println("isBundle");
        String n = "Bundle";
        SellerController instance = new SellerController(new DefaultListModel());
        boolean expResult = true;
        boolean result = instance.isBundle(n);
        assertEquals(expResult, result);
    }

    /**
     * Test of isValidDouble method, of class SellerController.
     */
    @Test
    public void testIsValidDouble() {
        System.out.println("isValidDouble");
        String n = "12.42";
        SellerController instance = new SellerController(new DefaultListModel());
        boolean expResult = false;
        boolean result = instance.isValidDouble(n);
        assertEquals(expResult, result);
    }

    /**
```

```

    * Test of isValidInteger method, of class SellerController.
    */
@Test
public void testIsValidInteger() {
    System.out.println("isValidInteger");
    String n = "15.";
    SellerController instance = new SellerController(new DefaultListModel());
    boolean expectedResult = true;
    boolean result = instance.isValidInteger(n);
    assertEquals(expectedResult, result);
}

/**
 * Test of isValidDiscountNumber method, of class SellerController.
 */
@Test
public void testIsValidDiscountNumber() {
    System.out.println("isValidDiscountNumber");
    String n = "100";
    SellerController instance = new SellerController(new DefaultListModel());
    boolean expectedResult = true;
    boolean result = instance.isValidDiscountNumber(n);
    assertEquals(expectedResult, result);
}
}

```

File: test/cop4331/model/BundleTest.java

```
package cop4331.model;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Rabih
 */
public class BundleTest {
    /**
     * Test of getID method, of class Bundle.
     */
    @Test
    public void testGetID() {
        System.out.println("getID");
        LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
        Bundle instance = new Bundle(10, 20);
        instance.add(item);
        int expResult = 10;
        int result = instance.getID();
        assertEquals(expResult, result);
    }

    /**
     * Test of setID method, of class Bundle.
     */
    @Test
    public void testSetID() {
        System.out.println("setID");
        int id = 50;
        LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
        Bundle instance = new Bundle(10, 20);
        instance.add(item);
        int expResult = 50;
        int result = instance.setID(id);
        assertEquals(expResult, result);
    }

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

```

@Test
public void testGetPrice() {
    System.out.println("getPrice");
    LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
    Bundle instance = new Bundle(10, 20);
    instance.add(item);
    double expectedResult = 10.0;
    double result = instance.getPrice();
    assertEquals(expectedResult, result, 0);
}

/**
 * Test of toString method, of class Bundle.
 */
@Test
public void testToString() {
    System.out.println("toString");
    LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
    Bundle instance = new Bundle(10, 20);
    instance.add(item);
    String expectedResult = "Bundle of Test";
    String result = instance.toString();
    assertEquals(expectedResult, result);
}

/**
 * Test of getName method, of class Bundle.
 */
@Test
public void testGetName() {
    System.out.println("getName");
    LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
    Bundle instance = new Bundle(10, 20);
    instance.add(item);
    String expectedResult = "Bundle";
    String result = instance.getName();
    assertEquals(expectedResult, result);
}

/**
 * Test of getDescription method, of class Bundle.
 */
@Test
public void testGetDescription() {
    System.out.println("getDescription");

```

```

        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        Bundle instance = new Bundle(10,20);
        instance.add(item);
        String expResult = "Bundle";
        String result = instance.getDescription();
        assertEquals(expResult, result);
    }

    /**
     * Test of getStock method, of class Bundle.
     */
    @Test
    public void testGetStock() {
        System.out.println("getStock");
        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        Bundle instance = new Bundle(10,20);
        instance.add(item);
        int expResult = 20;
        int result = instance.getStock();
        assertEquals(expResult, result);
    }

    /**
     * Test of setStock method, of class Bundle.
     */
    @Test
    public void testSetStock() {
        System.out.println("setStock");
        int stock = 50;
        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        Bundle instance = new Bundle(10,20);
        instance.add(item);
        instance.setStock(50);
        int expResult = 50;
        int result = instance.setStock(stock);
        assertEquals(expResult, result);
    }

    /**
     * Test of getInfo method, of class Bundle.
     */
    @Test
    public void testGetInfo() {
        System.out.println("getInfo");
        LineItem item = new Product(0,"Test","Test Item",10.00,10);

```

```

        Bundle instance = new Bundle(10,20);
        instance.add(item);
        String expResult = "Bundle of Test.\nPrice: $10.00\nQuantity: 20\n";
        String result = instance.getInfo();
        assertEquals(expResult, result);
    }

    /**
     * Test of getSellerInfo method, of class Bundle.
     */
    @Test
    public void testGetSellerInfo() {
        System.out.println("getSellerInfo");
        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        Bundle instance = new Bundle(10,20);
        instance.add(item);
        String expResult = "Bundle of Test.\nPrice: $10.00\nQuantity: 20\n";
        String result = instance.getSellerInfo();
        assertEquals(expResult, result);
    }
}

```


File: test/cop4331/model/DiscountedItemTest.java

```
package cop4331.model;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Rabih
 */
public class DiscountedItemTest {

    public DiscountedItemTest() {

    }

    /**
     * Test of getID method, of class DiscountedItem.
     */
    @Test
    public void testGetID() {
        System.out.println("getID");
        LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
        DiscountedItem instance = new DiscountedItem(item, 30);
        int expResult = 0;
        int result = instance.getID();
        assertEquals(expResult, result);
    }

    /**
     * Test of setID method, of class DiscountedItem.
     */
    @Test
    public void testSetID() {
        System.out.println("setID");
        int id = 10;
        LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
        DiscountedItem instance = new DiscountedItem(item, 30);
        int expResult = 10;
        int result = instance.setID(id);
        assertEquals(expResult, result);
    }

    /**
```

```

    * Test of getPrice method, of class DiscountedItem.
    */
@Test
public void testGetPrice() {
    System.out.println("getPrice");
    LineItem item = new Product(0,"Test","Test Item",10.00,10);
    DiscountedItem instance = new DiscountedItem(item,30);
    double expResult = 10.00 * (1.0 - (30.0 / 100.0));
    double result = instance.getPrice();
    assertEquals(expResult, result, 0);
}

/**
 * Test of toString method, of class DiscountedItem.
 */
@Test
public void testToString() {
    System.out.println("toString");
    LineItem item = new Product(0,"Test","Test Item",10.00,10);
    DiscountedItem instance = new DiscountedItem(item,30);
    String expResult = "Test (Discount 30.0%)";
    String result = instance.toString();
    assertEquals(expResult, result);
}

/**
 * Test of getName method, of class DiscountedItem.
 */
@Test
public void testGetName() {
    System.out.println("getName");
    LineItem item = new Product(0,"Test","Test Item",10.00,10);
    DiscountedItem instance = new DiscountedItem(item,30);
    String expResult = "Test";
    String result = instance.getName();
    assertEquals(expResult, result);
}

/**
 * Test of getDescription method, of class DiscountedItem.
 */
@Test
public void testGetDescription() {
    System.out.println("getDescription");
    LineItem item = new Product(0,"Test","Test Item",10.00,10);

```

```

        DiscountedItem instance = new DiscountedItem(item,30);
        String expResult = "Test Item";
        String result = instance.getDescription();
        assertEquals(expResult, result);
    }

    /**
     * Test of getStock method, of class DiscountedItem.
     */
    @Test
    public void testGetStock() {
        System.out.println("getStock");
        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        DiscountedItem instance = new DiscountedItem(item,30);
        int expResult = 10;
        int result = instance.getStock();
        assertEquals(expResult, result);
    }

    /**
     * Test of setStock method, of class DiscountedItem.
     */
    @Test
    public void testSetStock() {
        System.out.println("setStock");
        int stock = 50;
        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        DiscountedItem instance = new DiscountedItem(item,30);
        int expResult = 50;
        int result = instance.setStock(stock);
        assertEquals(expResult, result);
    }

    /**
     * Test of getInfo method, of class DiscountedItem.
     */
    @Test
    public void testGetInfo() {
        System.out.println("getInfo");
        LineItem item = new Product(0,"Test","Test Item",10.00,10);
        DiscountedItem instance = new DiscountedItem(item,30);
        String expResult = "Name: Test\nDescription: Test Item\nPrice:
$7.00\nQuantity: 10\nDiscount: 30.00%\n";
        String result = instance.getInfo();
        assertEquals(expResult, result);
    }

```

```

    }

    /**
     * Test of getSellerInfo method, of class DiscountedItem.
     */
    @Test
    public void testGetSellerInfo() {
        System.out.println("getSellerInfo");
        LineItem item = new Product(0, "Test", "Test Item", 10.00, 10);
        DiscountedItem instance = new DiscountedItem(item, 30);
        String expectedResult = "ID: 0\nName: Test\nDescription: Test Item\nInvoice  
Price: $5.00\nSell Price: $7.00\nQuantity: 10\nDiscount: 30.00%\n";
        String result = instance.getSellerInfo();
        assertEquals(expectedResult, result);
    }
}

```

File: test/cop4331/model/ProductTest.java

```
package cop4331.model;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Rabih
 */
public class ProductTest {
    /**
     * Test of getID method, of class Product.
     */
    @Test
    public void testGetID() {
        System.out.println("getID");
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        int expResult = 0;
        int result = instance.getID();
        assertEquals(expResult, result);
    }

    /**
     * Test of getPrice method, of class Product.
     */
    @Test
    public void testGetPrice() {
        System.out.println("getPrice");
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        double expResult = 10.00;
        double result = instance.getPrice();
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of setPrice method, of class Product.
     */
    @Test
    public void testSetPrice() {
        System.out.println("setPrice");
        double price = 50.0;
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        double expResult = 50.0;
```

```

        double result = instance.setPrice(price);
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of toString method, of class Product.
     */
    @Test
    public void testToString() {
        System.out.println("toString");
        Product instance = new Product(0, "Test", "Test Item", 10.00, 10);
        String expResult = "Test";
        String result = instance.toString();
        assertEquals(expResult, result);
    }

    /**
     * Test of getName method, of class Product.
     */
    @Test
    public void testGetName() {
        System.out.println("getName");
        Product instance = new Product(0, "Test", "Test Item", 10.00, 10);
        String expResult = "Test";
        String result = instance.getName();
        assertEquals(expResult, result);
    }

    /**
     * Test of getDescription method, of class Product.
     */
    @Test
    public void testGetDescription() {
        System.out.println("getDescription");
        Product instance = new Product(0, "Test", "Test Item", 10.00, 10);
        String expResult = "Test Item";
        String result = instance.getDescription();
        assertEquals(expResult, result);
    }

    /**
     * Test of getStock method, of class Product.
     */
    @Test
    public void testGetStock() {

```

```

        System.out.println("getStock");
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        int expResult = 10;
        int result = instance.getStock();
        assertEquals(expResult, result);
    }

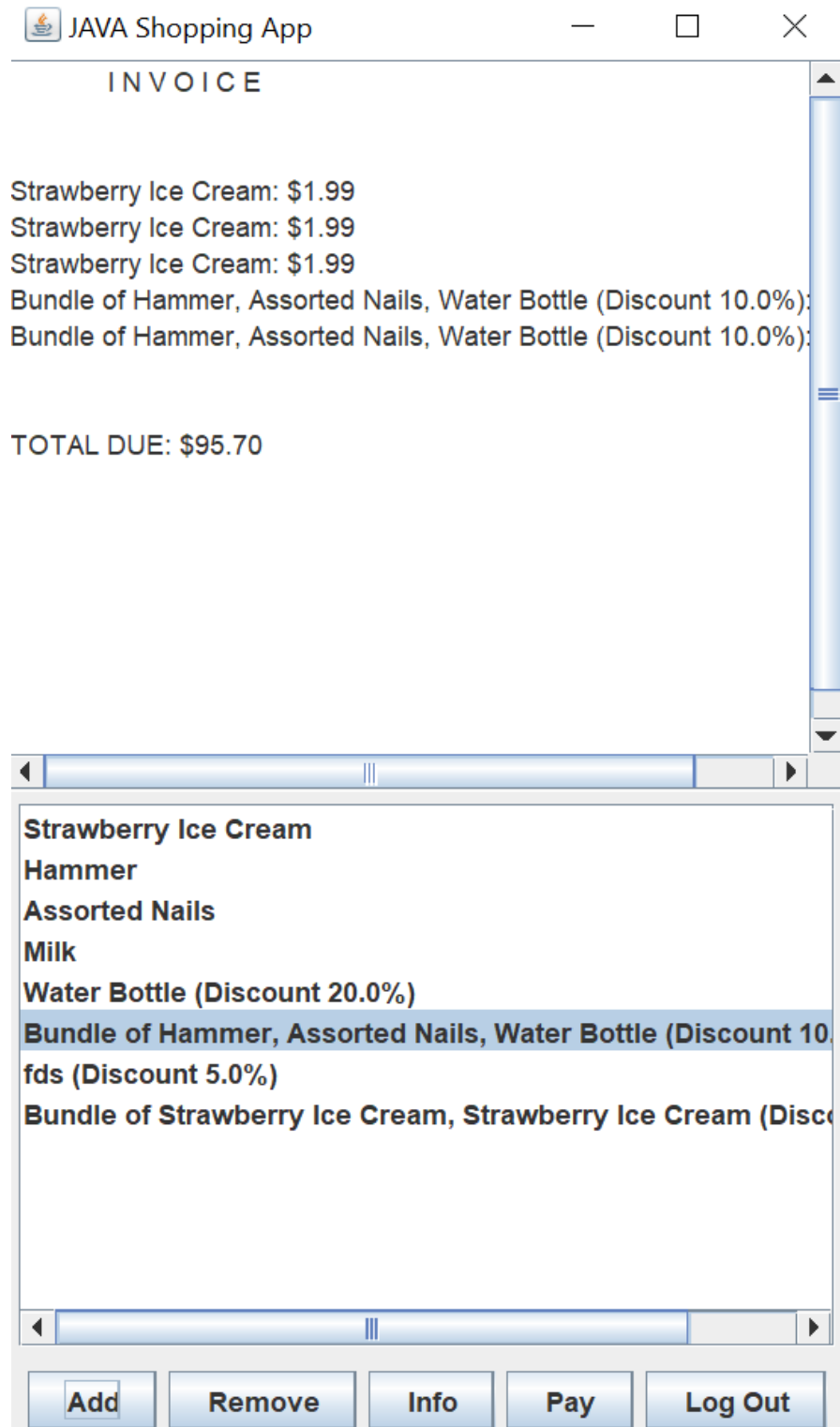
    /**
     * Test of setStock method, of class Product.
     */
    @Test
    public void testSetStock() {
        System.out.println("setStock");
        int stock = 40;
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        int expResult = 40;
        int result = instance.setStock(stock);
        assertEquals(expResult, result);
    }

    /**
     * Test of getInfo method, of class Product.
     */
    @Test
    public void testGetInfo() {
        System.out.println("getInfo");
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        String expResult = "Name: Test\nDescription: Test Item\nPrice:
$10.00\nQuantity: 10\n";
        String result = instance.getInfo();
        assertEquals(expResult, result);
    }

    /**
     * Test of getSellerInfo method, of class Product.
     */
    @Test
    public void testGetSellerInfo() {
        System.out.println("getSellerInfo");
        Product instance = new Product(0,"Test","Test Item",10.00,10);
        String expResult = "ID: 0\nName: Test\nDescription: Test Item\nInvoice
Price: $5.00\nSell Price: $10.00\nQuantity: 10\n";
        String result = instance.getSellerInfo();
        assertEquals(expResult, result);
    }
}

```

Screenshots



JAVA Shopping App

INVOICE

Strawberry Ice Cream: \$1.99
Strawberry Ice Cream: \$1.99
Strawberry Ice Cream: \$1.99
Bundle of Hammer, Assorted Nail: \$10.00 (Discount 10.0%)
Bundle of Hammer, Assorted Nail: \$10.00 (Discount 10.0%)
TOTAL DUE: \$95.70


Strawberry Ice Cream: \$1.99
Strawberry Ice Cream: \$1.99
Strawberry Ice Cream: \$1.99
Bundle of Hammer, Assorted Nail: \$10.00 (Discount 10.0%)
Bundle of Hammer, Assorted Nail: \$10.00 (Discount 10.0%)
TOTAL DUE: \$95.70

Card Number:

Card Name:

Date: cvv:

Invalid number. Must start with 4 and is 16 digits long.

 JAVA Shopping App

ID: 3
Name: Assorted Nails
Description: 20 Assorted Nails ideal for construction.
Invoice Price: \$4.98
Sell Price: \$9.95
Quantity: 10

Strawberry Ice Cream

Hammer

Assorted Nails

Milk

Water Bottle (Discount 20.0%)

Bundle of Hammer, Assorted Nails, Water Bottle (Discount 10.0%)

Bundle of Strawberry Ice Cream, Strawberry Ice Cream (Discount 5.0%)

New

Bundle

Remove

Profit

Log Out

Java Shopping App

Ad...

Items to Bundle:

Milk

Strawberry Ice Cream
Strawberry Ice Cream
Milk
Milk

Item Quantity:

20

Discount:

30

Back Add

Bundle

Strawberry Ice Cream
Hammer
Assorted Nails
Milk
Water Bottle (Discount 10)
Bundle of Hammer
Tools (Discount 10)
Bundle of Strawberry Ice Cream (Discount 10)

New Bundle Remove Profit Log Out