Shopping Cart Application

COP4331 – Group 3

Table of Contents

Use Cases

- Benjamin Carver

User Logs In

1. User enters username and password
2. System validates credentials
3. System redirects the user to the appropriate interface

Variation #1: Invalid credentials

1. Start at step 2
2. System displays an error message and requests the user to try again

Customer Adds Items to Shopping Cart

1. Customer browses for products
2. Customer selects a product
3. Customer specifies the desired quantity (within available stock)
4. Customer clicks “Add to Cart” button
5. System adds the product to the cart
6. System updates the cart display (item count and total cost)

Variation #1: Insufficient stock

1. Start at step 4
2. System displays an error message to the user and does not add the item(s) to the cart

Variation #2: Item already in cart

1. Start at step 4
2. System displays an error to change item quantity in cart

Customer Reviews Product Details

1. Customer browses for products
2. Customer selects a product
3. Customer clicks “View Details” button
4. System displays a detailed product view (description, price, etc.)

Customer Reviews/Updates Shopping Cart

1. Customer navigates to the shopping cart page
2. System displays cart contents
3. Customer can change item quantity and remove items
4. System updates the cart display accordingly

Customer Checks Out

1. Customer proceeds to the checkout page
2. Customer reviews the order summary

3. Customer provides shipping and payment details
4. Customer confirms the order
5. System processes the order
6. System creates an order record
7. System displays order confirmation

Variation #1: Out of stock item

1. Start at step 5
2. System displays an error message and cancels the order

Seller Reviews/Updates Inventory

1. Seller navigates to the inventory management page
2. System displays the current inventory list
3. Seller selects a product and clicks the “Edit” button
4. Seller can view product details, update product information, mark products as unavailable
5. System updates inventory accordingly

Variation #1: Invalid input

1. Start at step 4
2. Seller inputs invalid information (negative price, null description)
3. System displays an error message and prompts for correction

Seller Adds New Product

1. Seller navigates to the inventory management page
2. System displays the current inventory list
3. Seller clicks the “Add Product” button
4. System displays add product form
5. Seller inputs product details and clicks the “Add Product” button
6. System validates the input
7. System adds item to the inventory

Variation #1: Invalid input

1. Start at step 6
2. System displays an error message and prompts for correction.

Seller Creates Product Bundle

1. Seller navigates to the inventory management system
2. System displays the current inventory list
3. Seller clicks the “Create Bundle” button
4. System displays the create bundle window
5. Seller selects products to add and clicks the “Create Bundle” button
6. System creates the product bundle in the inventory

Seller Applies Discount

1. Seller selects a product or bundle
2. Seller clicks the “Discount” button
3. System displays the discount window
4. Seller selects the discount type (percentage or fixed) and amount
5. System applies the discount

Variation #1: Invalid discount

1. Start at step 4
2. System displays an error message and prompts for correction

CRC Cards

- Jeremy Ladanowski

Product

- Responsibilities
    - Maintain product information
    - Provide product details
    - Update stock quantity
    - Check availability
- Collaborators
    - Inventory
    - ShoppingCart
    - Order

Inventory

- Responsibilities
    - Manage the list of products
    - Add new products to the inventory
    - Update existing product information
    - Remove products from the inventory
    - Find products by various criteria
    - Maintain the availability of products
- Collaborators
    - Product
    - Seller
    - ShoppingCart
    - Order

ShoppingCart

- Responsibilities
    - Hold items added by the customer
    - Add items to the cart

  - Remove items from the cart
  - Update item quantities in the cart
  - Calculate the total cost of items in the cart
  - Maintain the current state of the cart for each customer
- Collaborators
  - Product
  - Inventory
  - Customer
  - Discount

Order

- Responsibilities
  - Represent a completed order
  - Store order details
  - Add items to the order
  - Calculate the total order amount
  - Maintain the order history
- Collaborators
  - Customer
  - ShoppingCart

User

- Responsibilities
  - Maintain user information
  - Authenticate user credentials
  - Provide user role
- Collaborators
  - LoginController

Bundle

- Responsibilities
  - Represent a group of products sold together
  - Add products to the bundle
  - Remove products from the bundle
  - Calculate the total price of the bundle
- Collaborators
  - Product
  - Inventory
  - Seller

Discount

- Responsibilities
  - Represent a discount applied to a product or bundle

    - Apply the discount to the product or bundle
    - Calculate the discounted price
- Collaborators
    - Product
    - Bundle
    - ShoppingCart

LoginController

- Responsibilities
    - Handle user authentication
    - Validate user credentials
    - Redirect user to the appropriate dashboard based on role
- Collaborators
    - Product
    - Inventory
    - CustomerDashboard
    - SellerDashboard

ShoppingCartController

- Responsibilities
    - Manage adding items to cart
    - Manage updating item quantities in cart
    - Manage removing items from the cart
- Collaborators
    - ShoppingCart
    - Product
    - CustomerDashboard

CheckoutController

- Responsibilities
    - Handle the checkout process
    - Create order record
- Collaborators
    - ShoppingCart
    - Order
    - CustomerDashboard

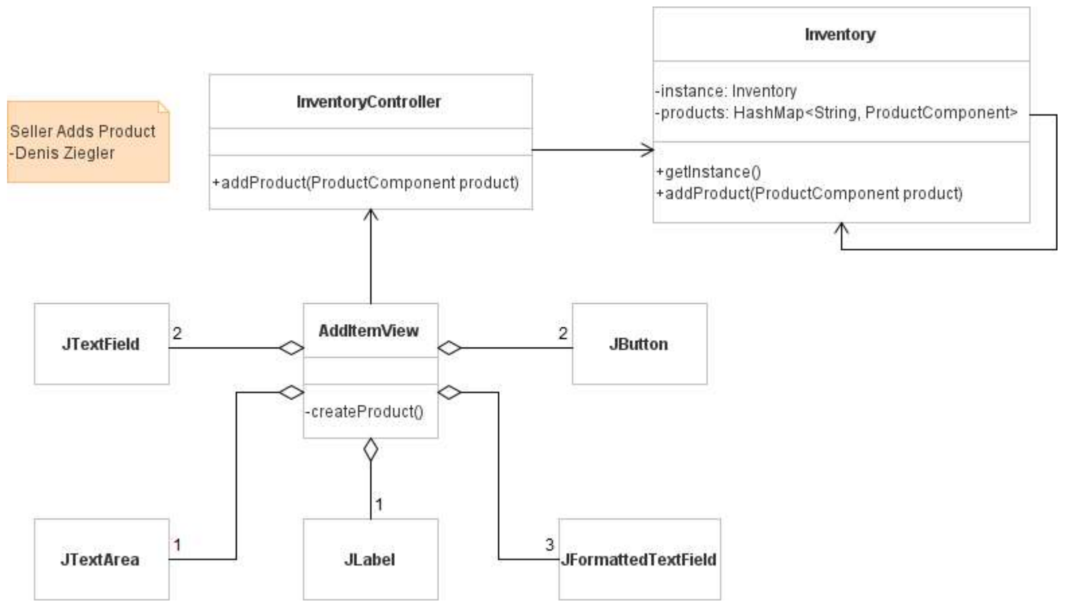
InventoryController

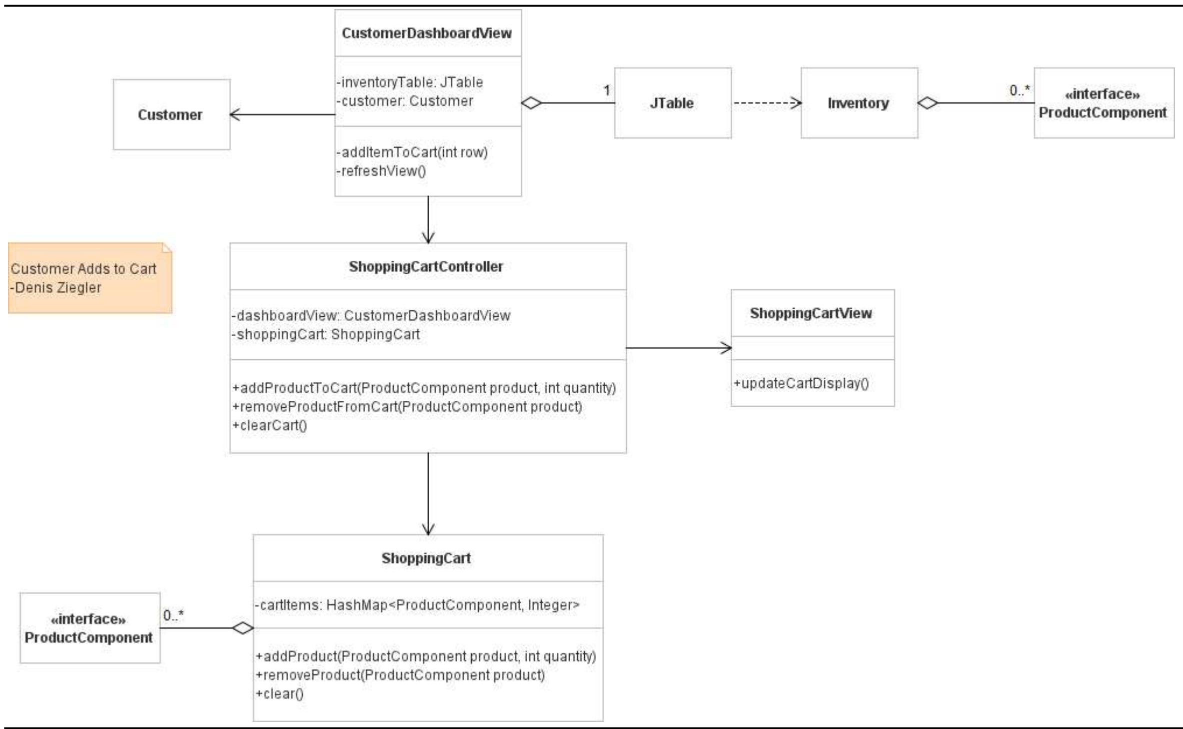
- Responsibilities
    - Manage viewing and updating inventory
    - Add new products to the inventory
    - Remove products from the inventory
- Collaborators

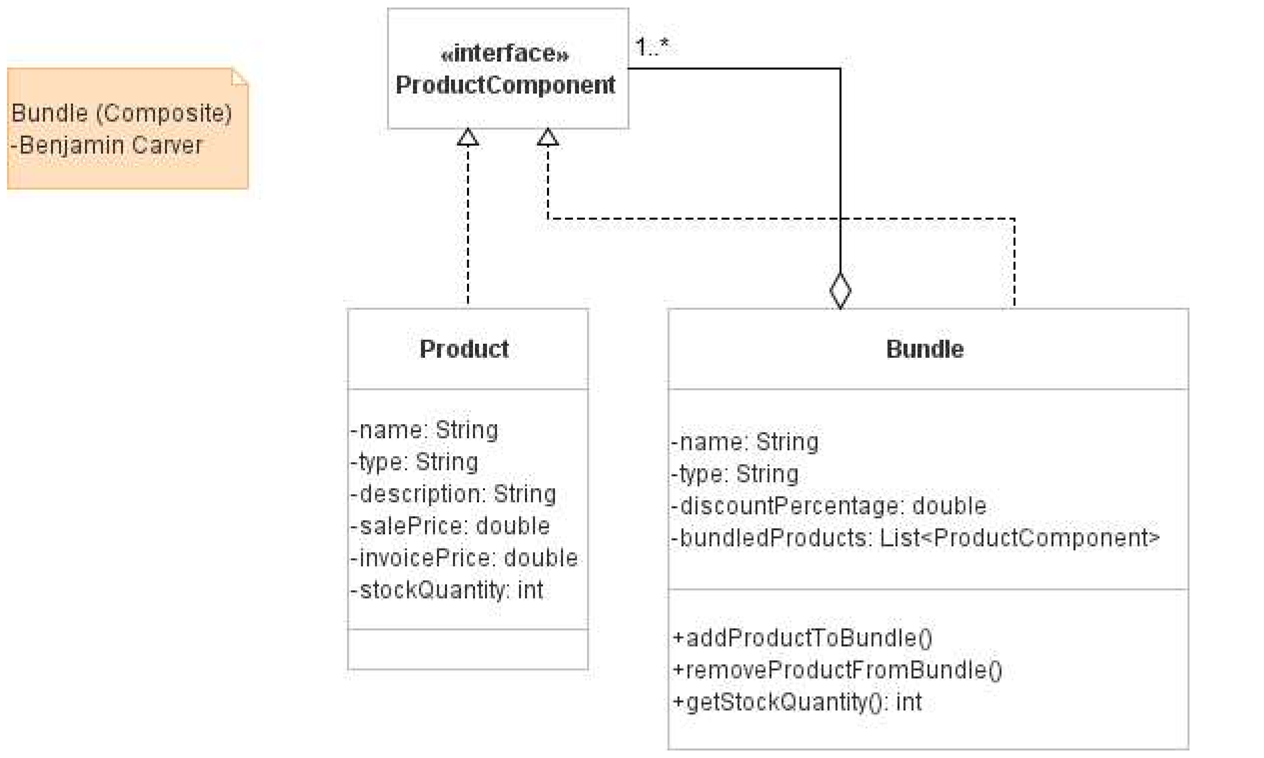
- Inventory
- Product
- SellerDashboard

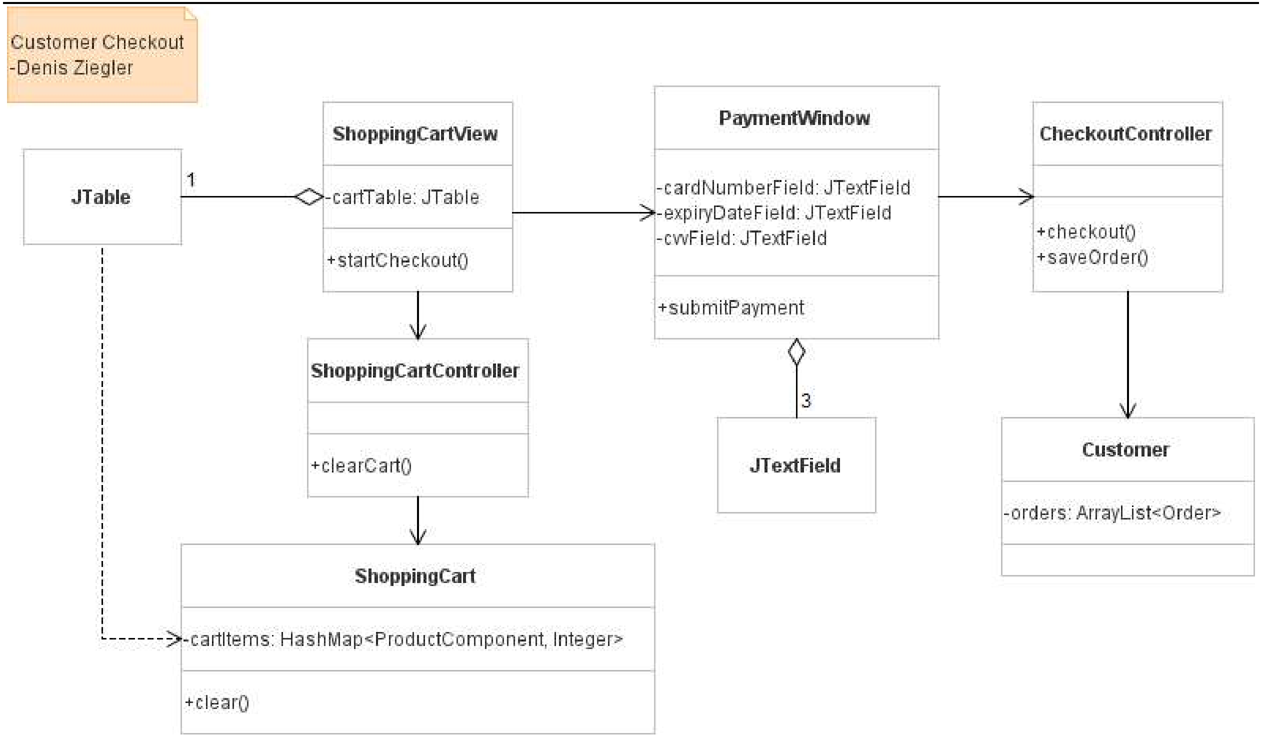
UML Diagrams

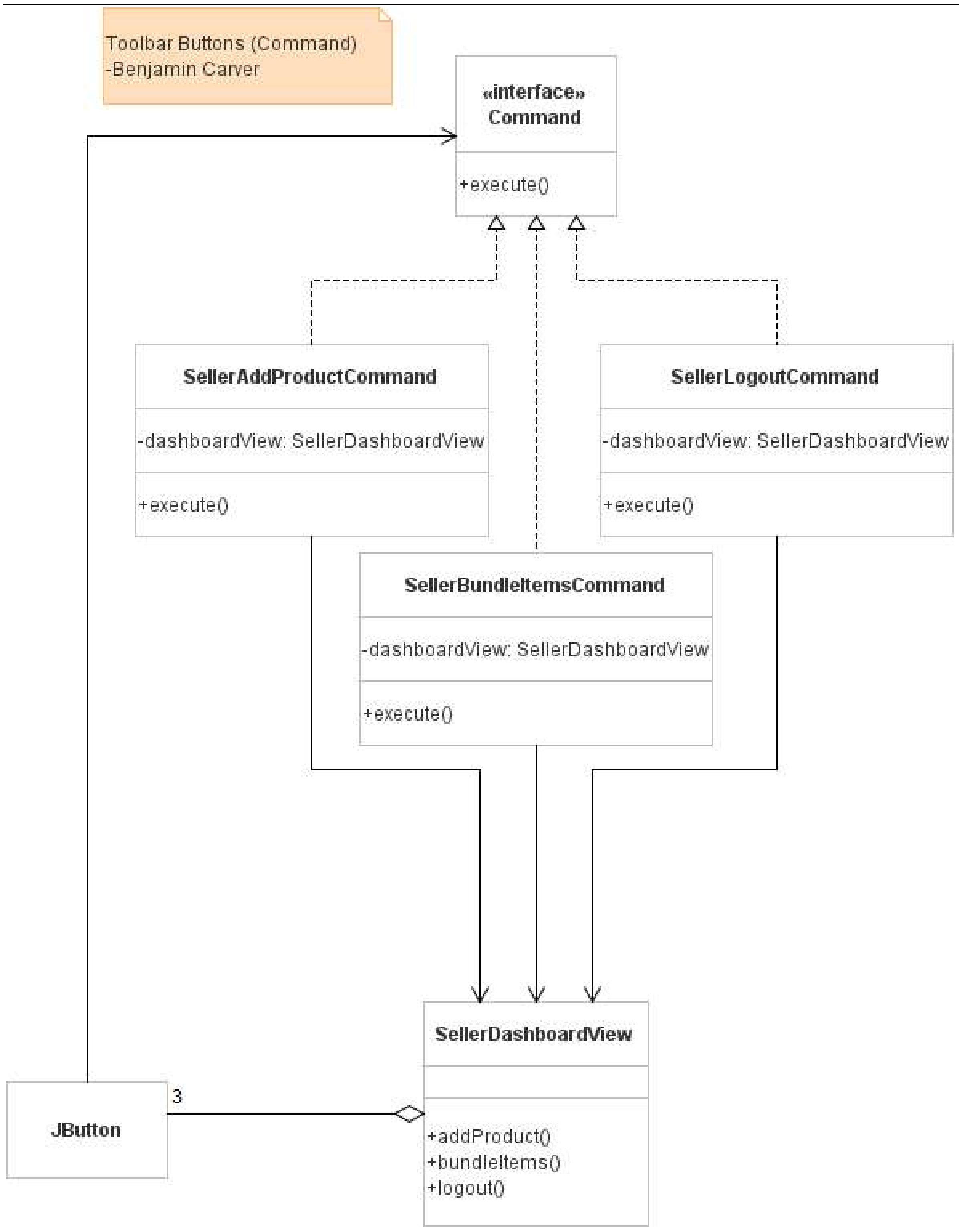
Class Diagrams

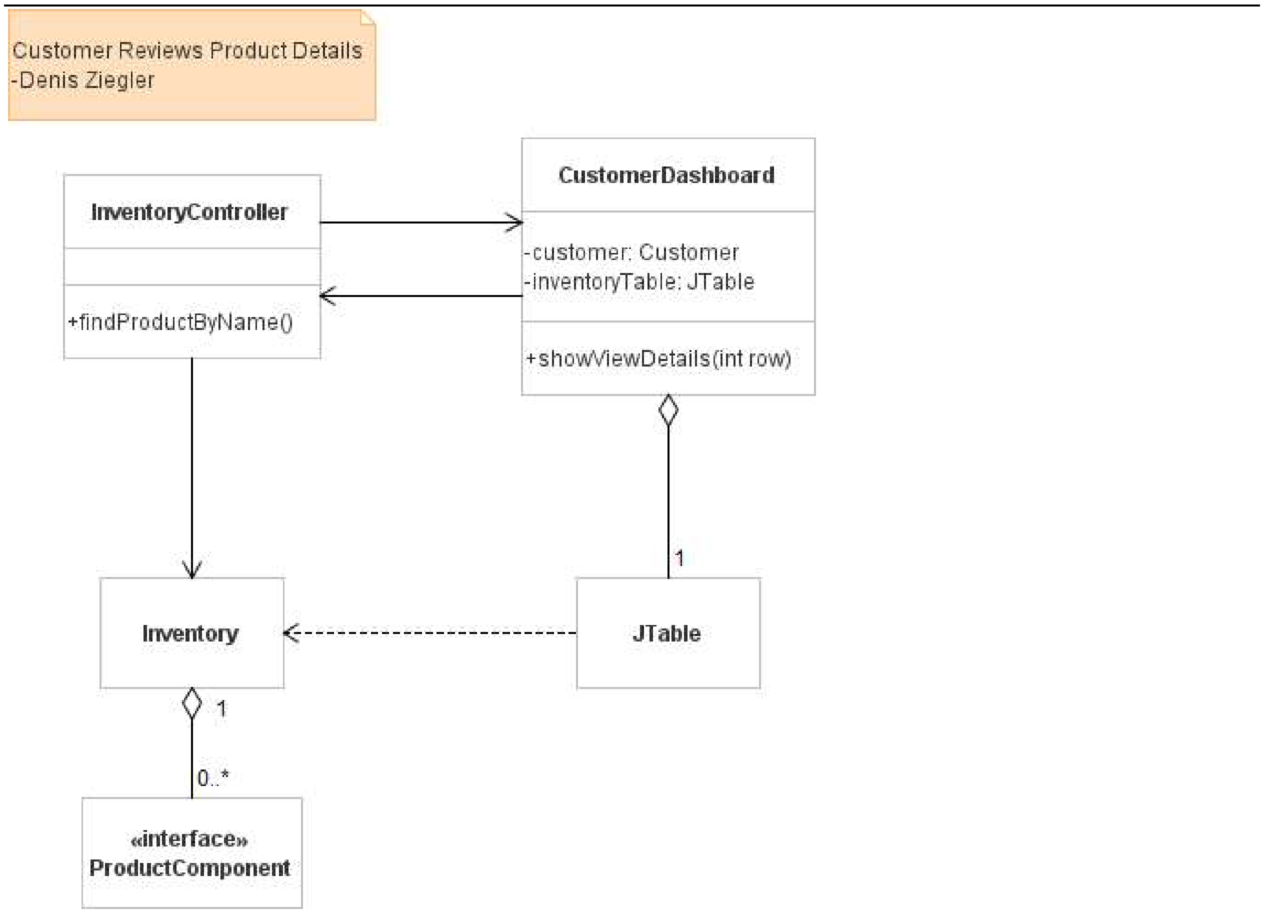


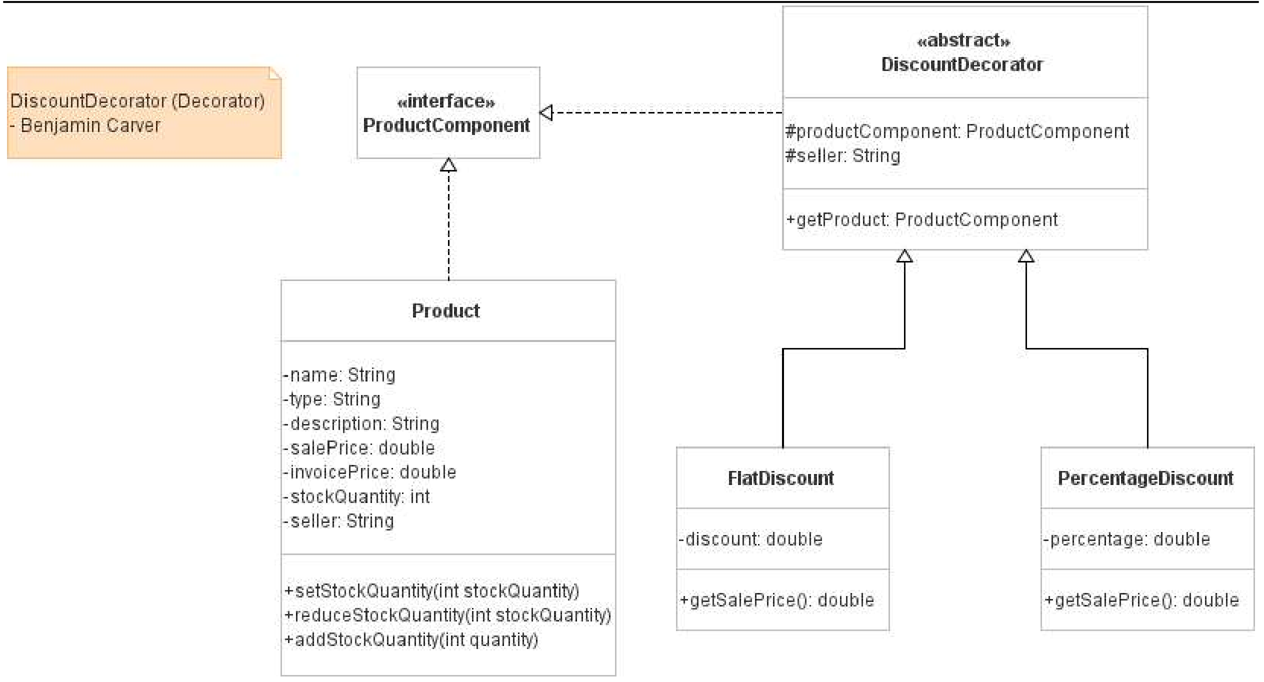


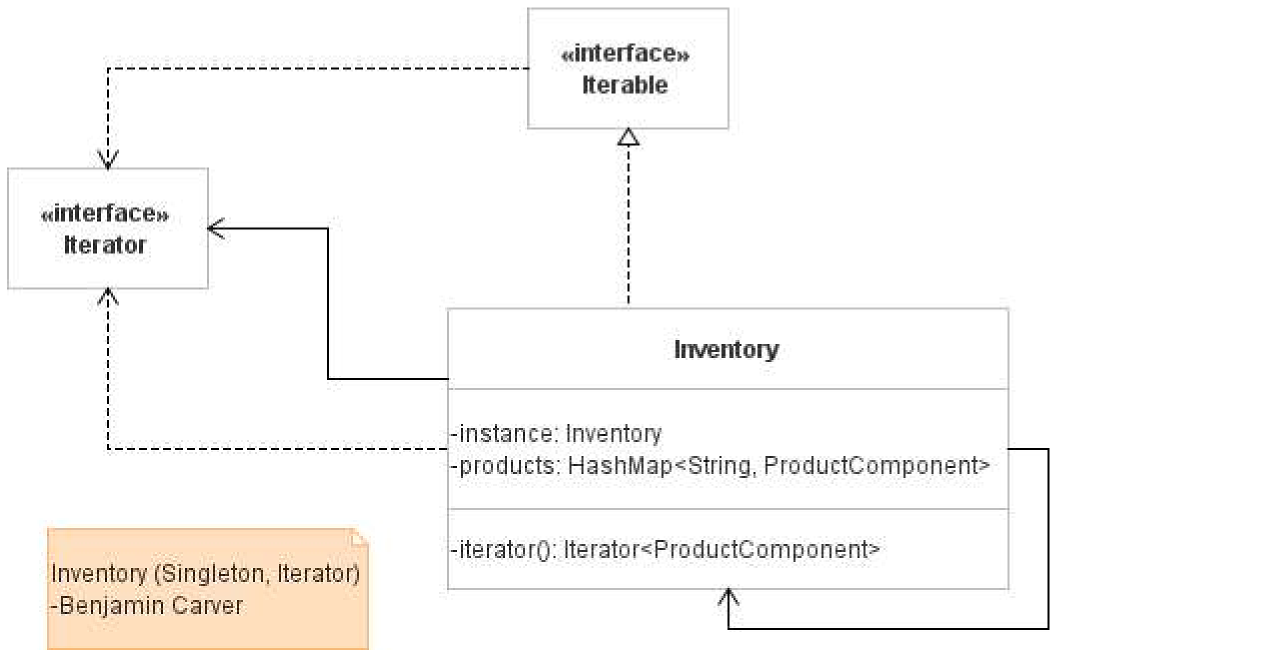

Bundle (Composite)
-Benjamin Carver
«interface»
ProductComponent
1..*
Product
-name: String
-type: String
-description: String
-salePrice: double
-invoicePrice: double
-stockQuantity: int
Bundle
-name: String
-type: String
-discountPercentage: double
-bundledProducts: List<ProductComponent>
+addProductToBundle()
+removeProductFromBundle()
+getStockQuantity(): int

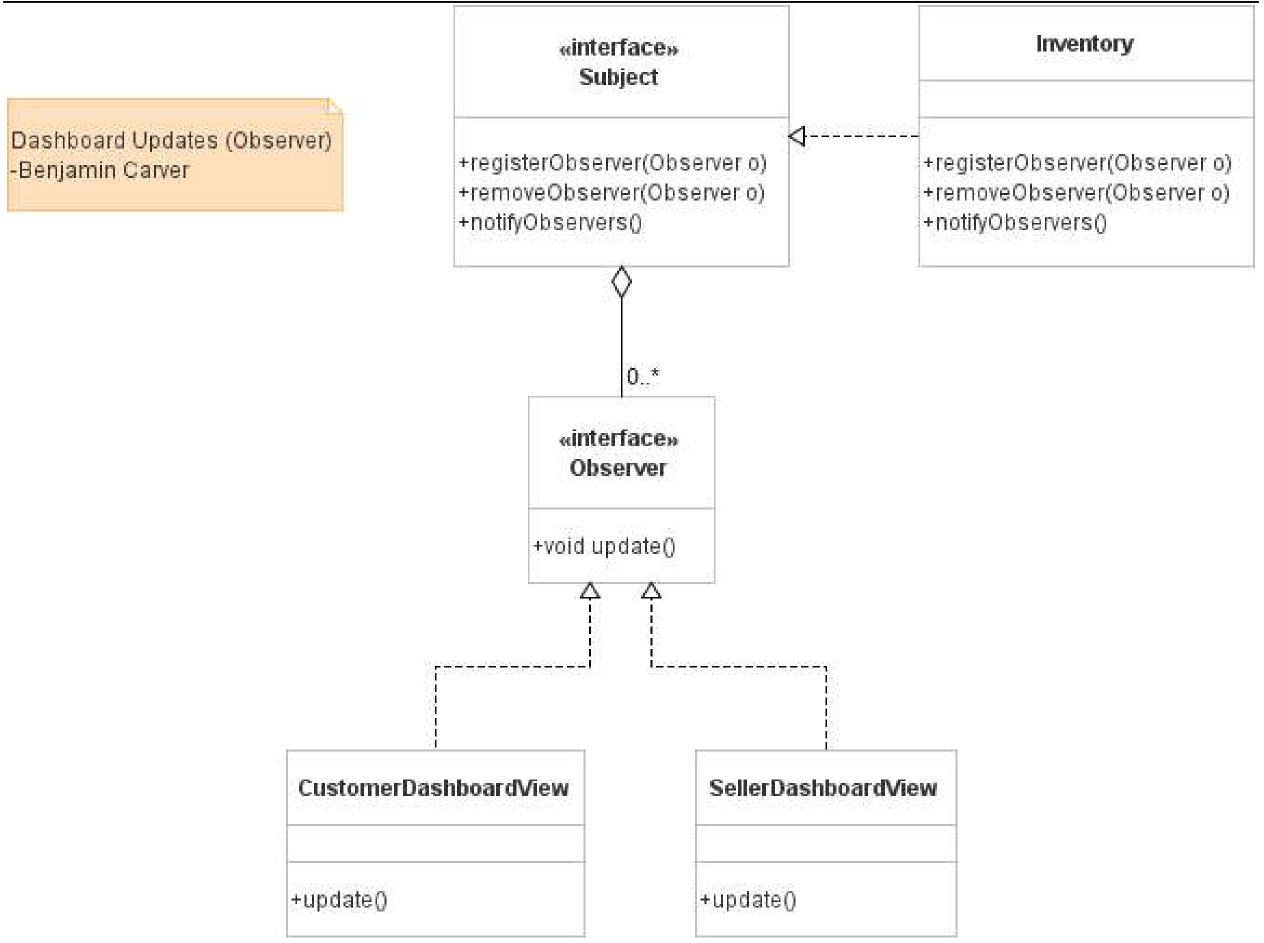

Customer Checkout
-Denis Ziegler
JTable
1
ShoppingCartView
-cartTable: JTable
+startCheckout()
PaymentWindow
-cardNumberField: JTextField
-expiryDateField: JTextField
-cvvField: JTextField
+submitPayment
CheckoutController
+checkout()
+saveOrder()
ShoppingCartController
+clearCart()
3
JTextField
Customer
-orders: ArrayList<Order>
ShoppingCart
-cartItems: HashMap<ProductComponent, Integer>
+clear()

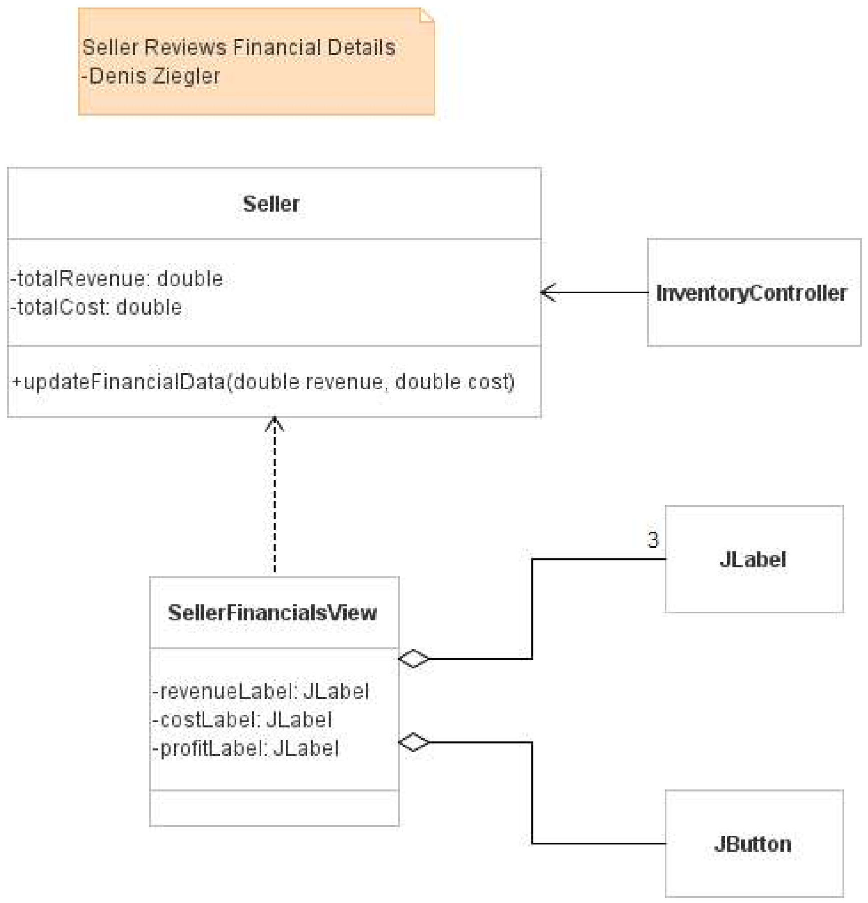

Toolbar Buttons (Command)
-Benjamin Carver
«interface»
Command
+execute()
SellerAddProductCommand
-dashboardView: SellerDashboardView
+execute()
SellerLogoutCommand
-dashboardView: SellerDashboardView
+execute()
SellerBundleItemsCommand
-dashboardView: SellerDashboardView
+execute()
SellerDashboardView
+addProduct()
+bundleItems()
+logout()
JButton
3

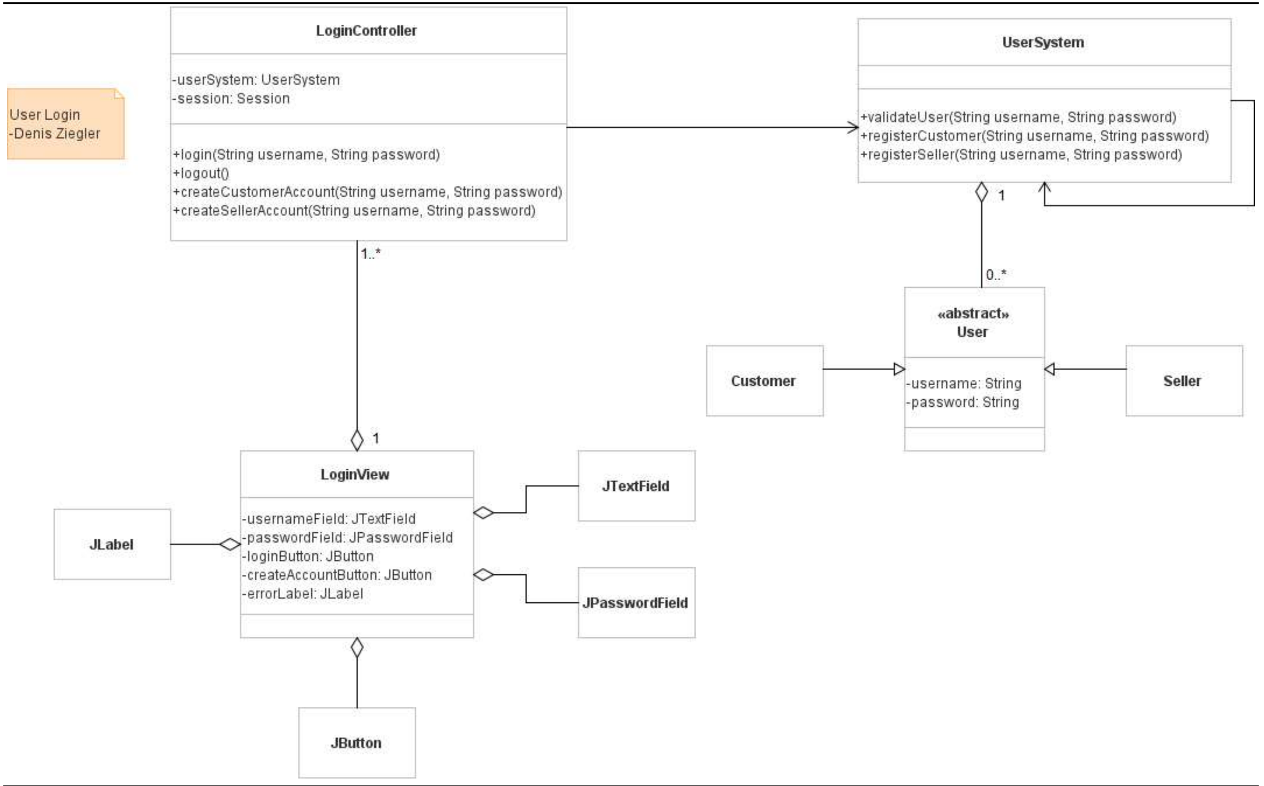

Customer Reviews Product Details
-Denis Ziegler
InventoryController
+findProductByName()
CustomerDashboard
-customer: Customer
-inventoryTable: JTable
+showViewDetails(int row)
1
Inventory
JTable
1
0..*
«interface»
ProductComponent

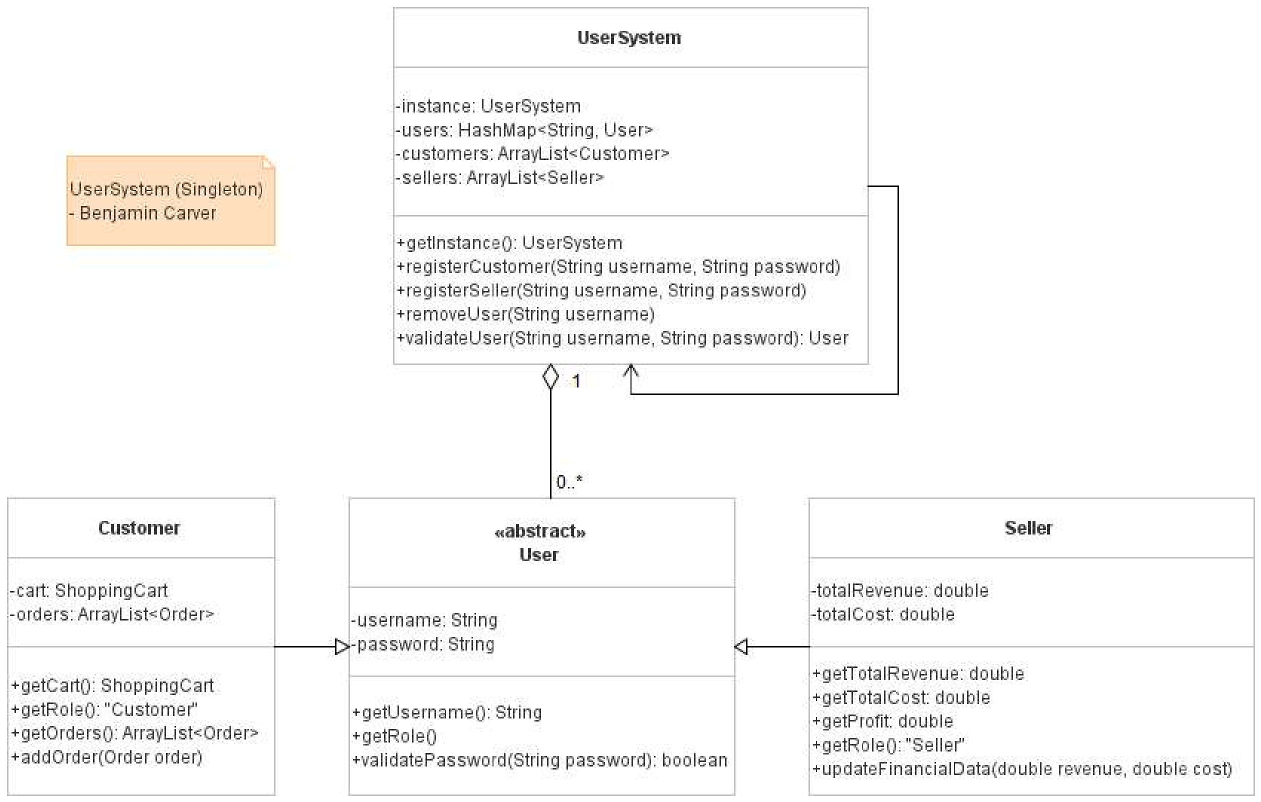

DiscountDecorator (Decorator)
- Benjamin Carver
«interface»
ProductComponent
«abstract»
DiscountDecorator
#productComponent: ProductComponent
#seller: String
+getProduct: ProductComponent
Product
-name: String
-type: String
-description: String
-salePrice: double
-invoicePrice: double
-stockQuantity: int
-seller: String
+setStockQuantity(int stockQuantity)
+reduceStockQuantity(int stockQuantity)
+addStockQuantity(int quantity)
FlatDiscount
-discount: double
+getSalePrice(): double
PercentageDiscount
-percentage: double
+getSalePrice(): double


«interface»
Iterable
«interface»
Iterator
Inventory
-instance: Inventory
-products: HashMap<String, ProductComponent>
-iterator(): Iterator<ProductComponent>
Inventory (Singleton, Iterator)
-Benjamin Carver

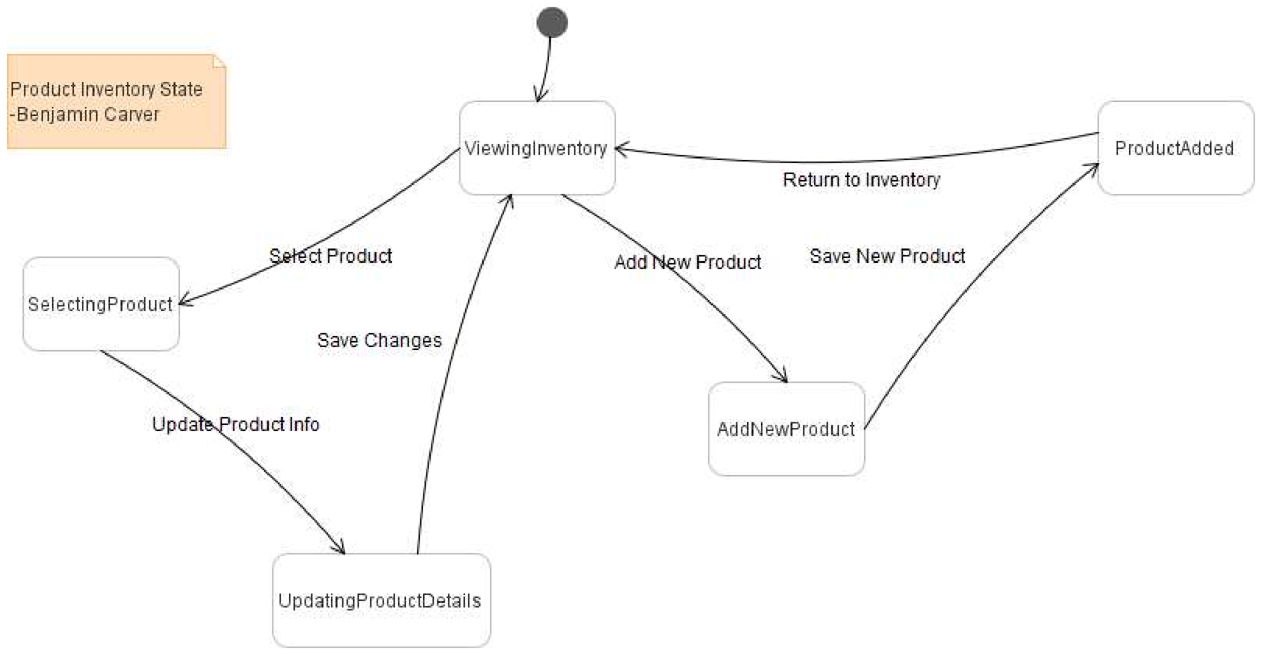

«interface»
Subject
+registerObserver(Observer o)
+removeObserver(Observer o)
+notifyObservers()
Inventory
+registerObserver(Observer o)
+removeObserver(Observer o)
+notifyObservers()
Dashboard Updates (Observer)
-Benjamin Carver
0..*
«interface»
Observer
+void update()
CustomerDashboardView
+update()
SellerDashboardView
+update()

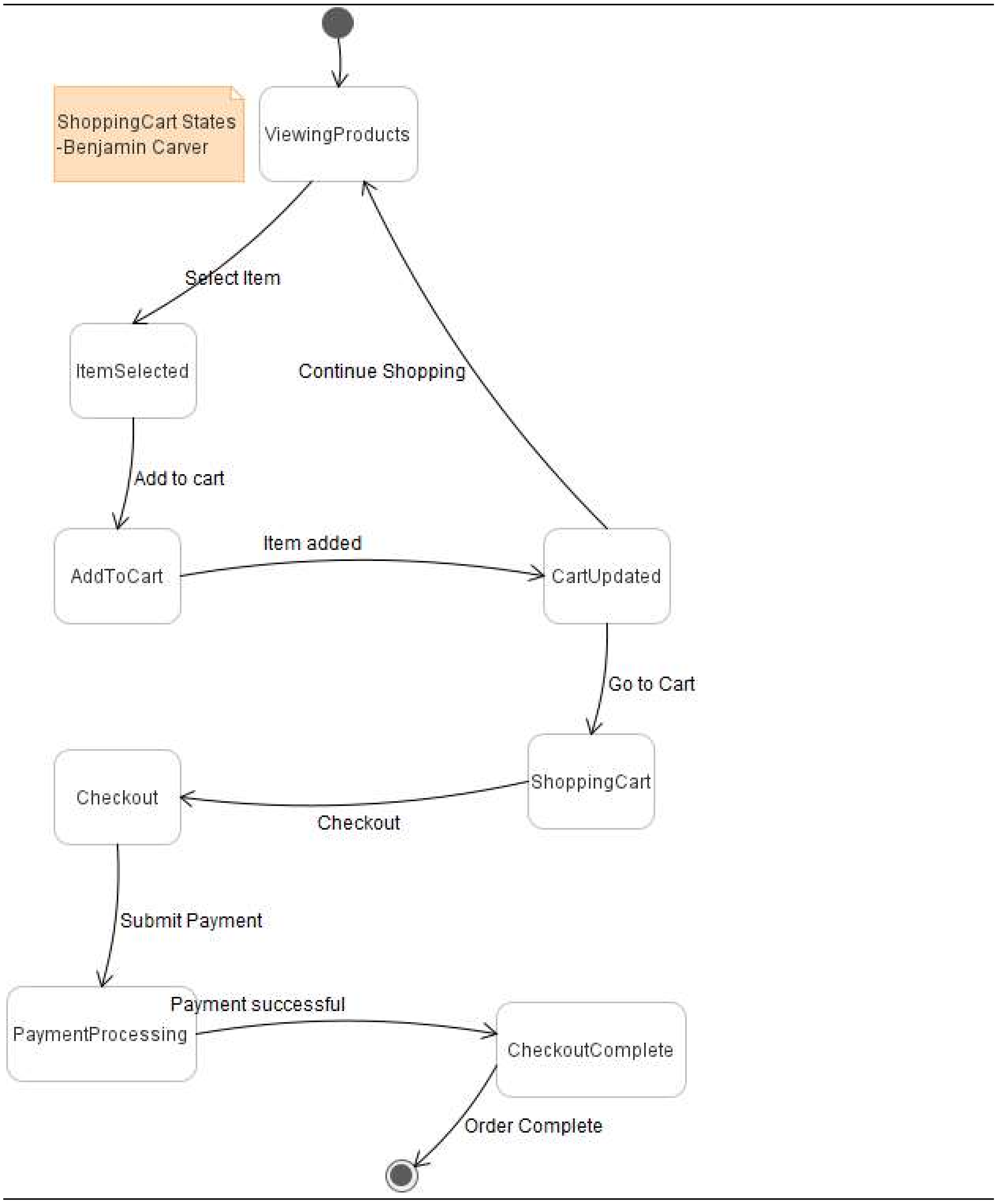

Seller Reviews Financial Details
-Denis Ziegler
Seller
-totalRevenue: double
-totalCost: double
+updateFinancialData(double revenue, double cost)
InventoryController
SellerFinancialsView
-revenueLabel: JLabel
-costLabel: JLabel
-profitLabel: JLabel
3
JLabel
JButton

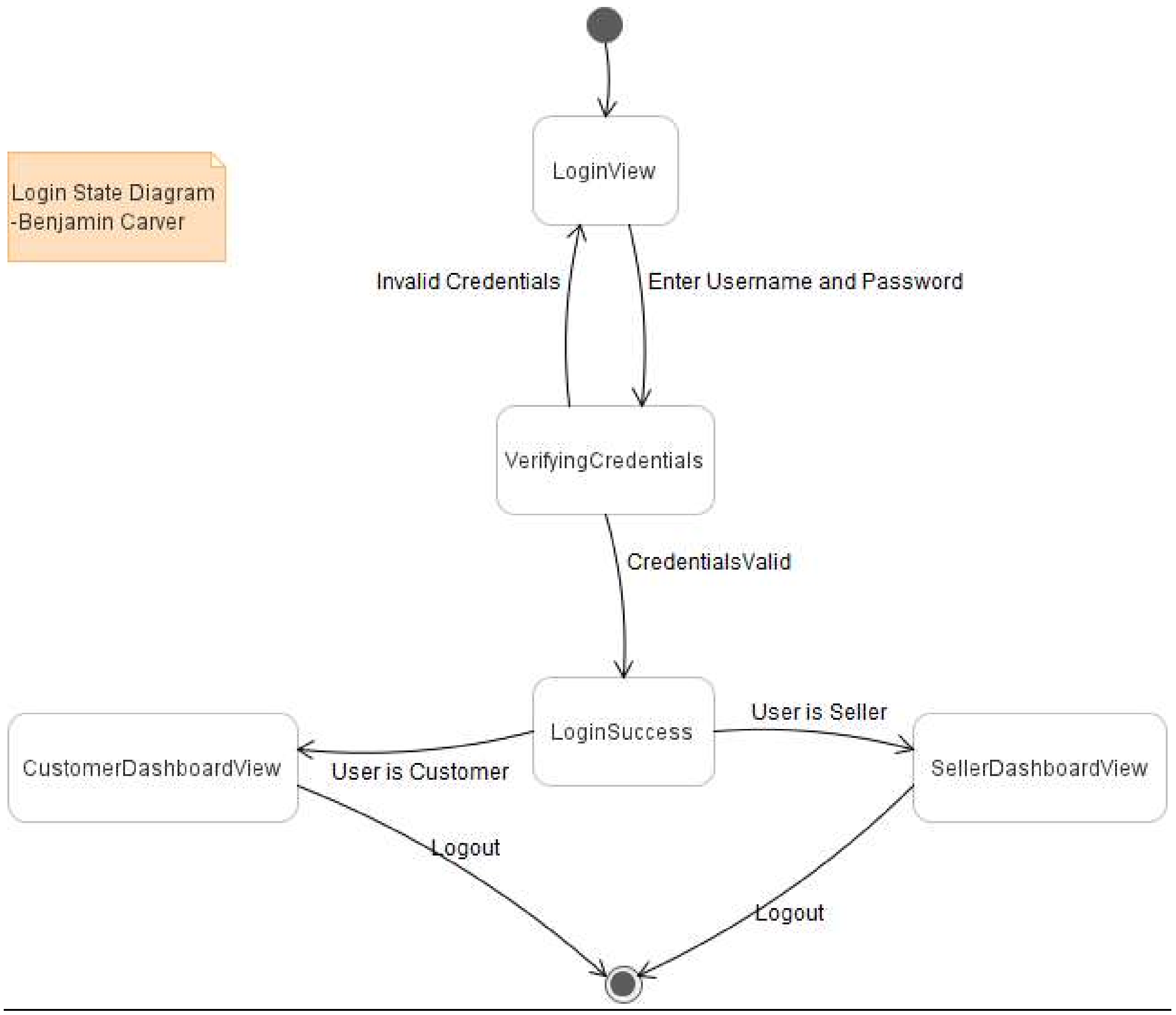

User Login
-Denis Ziegler
LoginController
-userSystem: UserSystem
-session: Session
+login(String username, String password)
+logout()
+createCustomerAccount(String username, String password)
+createSellerAccount(String username, String password)
UserSystem
+validateUser(String username, String password)
+registerCustomer(String username, String password)
+registerSeller(String username, String password)
1
0..*
«abstract»
User
-username: String
-password: String
Customer
Seller
1..*
1
LoginView
-usernameField: JTextField
-passwordField: JPasswordField
-loginButton: JButton
-createAccountButton: JButton
-errorLabel: JLabel
JTextField
JPasswordField
JLabel
JButton


UserSystem (Singleton)
- Benjamin Carver
UserSystem
-instance: UserSystem
-users: HashMap<String, User>
-customers: ArrayList<Customer>
-sellers: ArrayList<Seller>
+getInstance(): UserSystem
+registerCustomer(String username, String password)
+registerSeller(String username, String password)
+removeUser(String username)
+validateUser(String username, String password): User
1
0..*
Customer
-cart: ShoppingCart
-orders: ArrayList<Order>
+getCart(): ShoppingCart
+getRole(): "Customer"
+getOrders(): ArrayList<Order>
+addOrder(Order order)
«abstract»
User
-username: String
-password: String
+getUsername(): String
+getRole()
+validatePassword(String password): boolean
Seller
-totalRevenue: double
-totalCost: double
+getTotalRevenue: double
+getTotalCost: double
+getProfit: double
+getRole(): "Seller"
+updateFinancialData(double revenue, double cost)

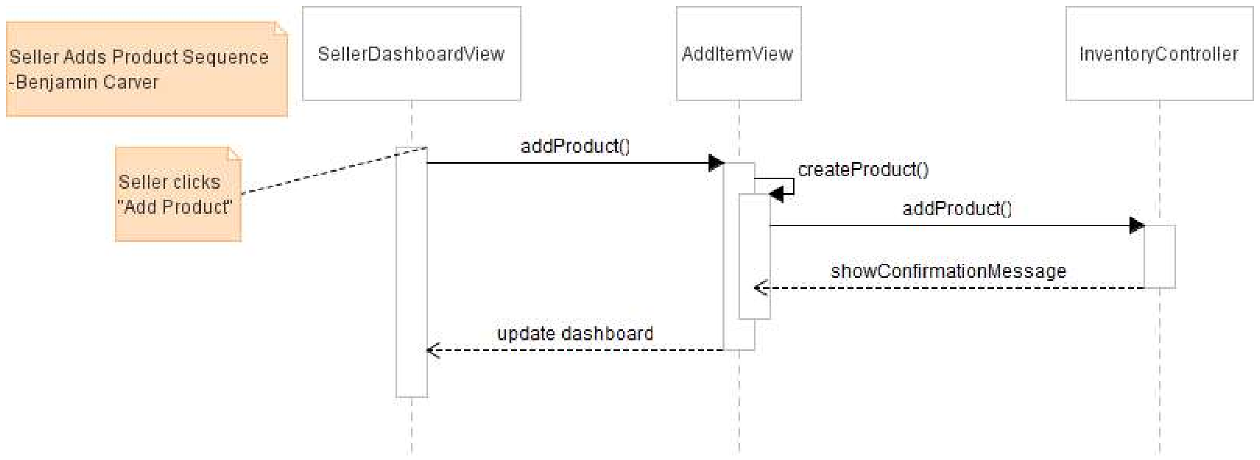
State Diagrams

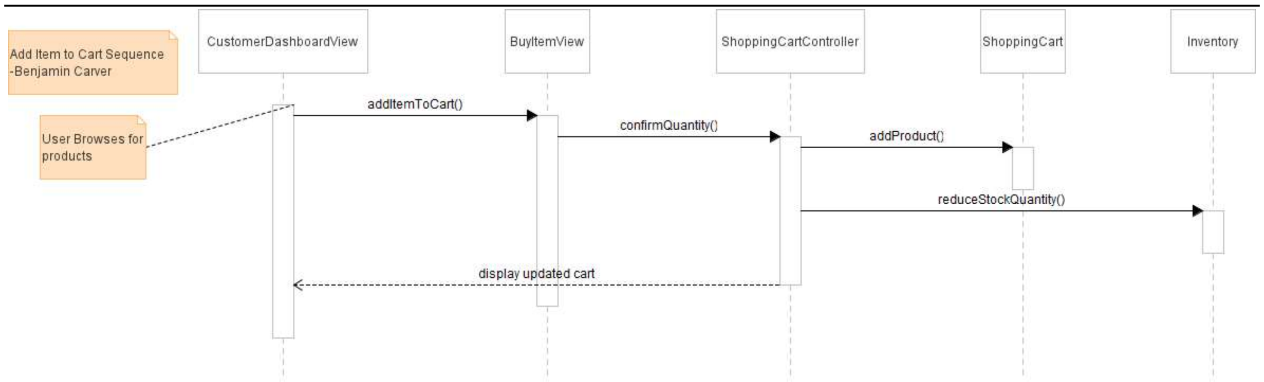


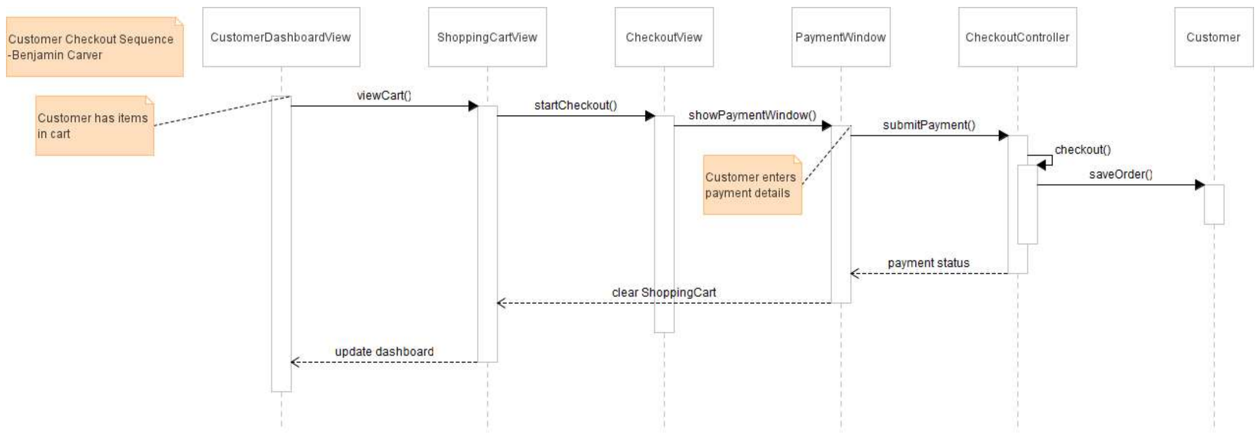

ShoppingCart States
-Benjamin Carver
ViewingProducts
Select Item
ItemSelected
Continue Shopping
Add to cart
AddToCart
Item added
CartUpdated
Go to Cart
ShoppingCart
Checkout
Checkout
Submit Payment
PaymentProcessing
Payment successful
CheckoutComplete
Order Complete

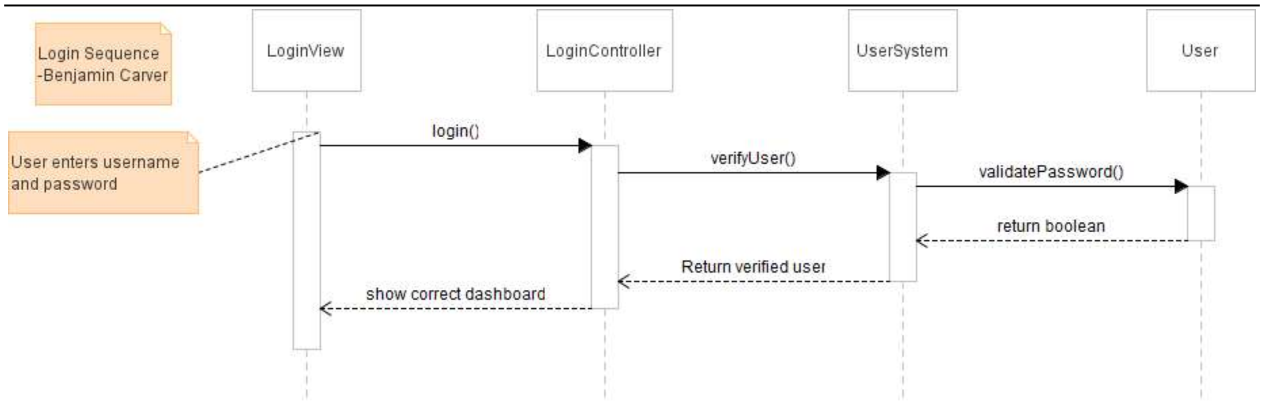

Login State Diagram
-Benjamin Carver
LoginView
Invalid Credentials
Enter Username and Password
VerifyingCredentials
CredentialsValid
LoginSuccess
User is Customer
User is Seller
CustomerDashboardView
SellerDashboardView
Logout
Logout

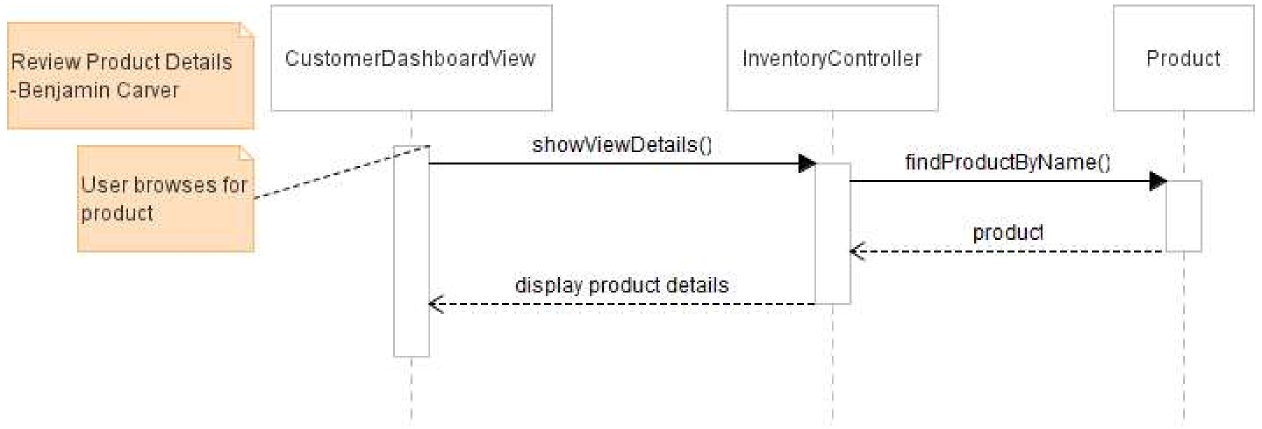
Sequence Diagrams











Source Code

cop4331.controller

CheckoutController.java:

```
package cop4331.controller;

import cop4331.model.ProductComponent;
import cop4331.model.customer.Customer;
import cop4331.model.customer.Order;
import cop4331.model.customer.ShoppingCart;
import cop4331.view.customer.CustomerDashboardView;

import javax.swing.*;
import java.util.HashMap;

/**
 * <p>Handles the checkout process for a customer including generating an
invoice,
 * processing payment, and saving an order.</p>
 * @author Benjamin Carver
 */
public class CheckoutController {
    private CustomerDashboardView dashboardView;

    /**
```

```
     * <p>Constructs a CheckoutController object.</p>
     *
     * @param dashboardView The customer dashboard view.
     */
    public CheckoutController(CustomerDashboardView dashboardView) {
        this.dashboardView = dashboardView;
    }

    /**
     * <p>Takes a User's cart and produces an invoice with product details,
quantities,
     * and the total cost.</p>
     * @param cart The User's cart.
     * @return The invoice in String form.
     */
    public String generateInvoice(ShoppingCart cart) {
        StringBuilder invoice = new StringBuilder();
        invoice.append("Order:\n");
        HashMap<ProductComponent, Integer> cartItems = cart.getCartItems();

        for (ProductComponent product : cartItems.keySet()) {
            double price = product.getSalePrice();
            int quantity = cartItems.get(product);
            invoice.append(product.getName()).append(" x").append(quantity)
                    .append(" = $").append(String.format("%.2f", price *
quantity)).append("\n");

        }
        invoice.append("Total: $").append(String.format("%.2f",
cart.calculateTotal()));
        return invoice.toString();
    }

    /**
     * <p>Simulates payment processing using provided information.</p>
     * @param cardNumber The card's number.
     * @param expiryDate The card's expiry date.
     * @param cvv The card's security code.
     * @return true, always simulates successful payment.
     */
    public boolean processPayment(String cardNumber, String expiryDate,
String cvv) {
        System.out.println("Processing payment...");
        return true;
    }

    /**
     * <p>Saves the order to the User's account.</p>
     * @param customer The User who made the order.
     * @param cart The User's cart.
     * @param invoice The invoice generated for the order.
     */
    public void saveOrder(Customer customer, ShoppingCart cart, String
invoice) {
        Order order = new Order(customer, cart.getCartItems(),
cart.calculateTotal(), invoice);
        customer.addOrder(order);
    }

    /**
     * <p>Performs the checkout process for the customer.</p>
     *
     * @param customer   The customer performing the checkout.
     * @param cart       The customer's shopping cart.
     * @param cardNumber The card number for payment.
     * @param expiryDate The expiry date of the card.
     * @param cvv        The CVV security code of the card.
     */
    public void checkout(Customer customer, ShoppingCart cart,
                         String cardNumber, String expiryDate, String cvv) {
        String invoice = generateInvoice(cart);
        boolean paymentSuccessful = processPayment(cardNumber, expiryDate,
cvv);
        System.out.println(invoice);

        if (paymentSuccessful) {
            saveOrder(customer, cart, invoice);

            cart.getCartItems().clear();
        } else {
            JOptionPane.showMessageDialog(null, "Payment failed");
        }
    }
}
```

InventoryController.java

```
package cop4331.controller;

import cop4331.model.Inventory;
import cop4331.model.ProductComponent;
import cop4331.view.customer.CustomerDashboardView;
import cop4331.view.seller.SellerDashboardView;

import java.util.Iterator;

/**
 * <p>Handles interactions with the product inventory such as adding and
removing products,
 * viewing products, and searching for products.</p>
 * @author Benjamin Carver
 */
public class InventoryController {
    private Inventory inventory;
    private CustomerDashboardView customerDashboardView;
    private SellerDashboardView sellerDashboardView;

    /**
     * <p>Constructs an InventoryController object.</p>
     */
    public InventoryController() {
        this.inventory = Inventory.getInstance();
    }
```

```java
    /**
     * <p>Constructs an InventoryController object.</p>
     * @param customerDashboardView The customer dashboard view.
     */
    public InventoryController(CustomerDashboardView customerDashboardView) {
        this.inventory = Inventory.getInstance();
        this.customerDashboardView = customerDashboardView;
    }

    /**
     * <p>Constructs an InventoryController object.</p>
     * @param sellerDashboardView The seller dashboard view.
     */
    public InventoryController(SellerDashboardView sellerDashboardView) {
        this.inventory = Inventory.getInstance();
        this.sellerDashboardView = sellerDashboardView;
    }

    /**
     * <p>Adds a product to the inventory.</p>
     * @param product The product to be added.
     */
    public boolean addProduct(ProductComponent product) {
        try {
            inventory.addProduct(product);
        } catch (IllegalArgumentException e) {
            System.err.println(e.getMessage());
            return false;
        }

        sellerDashboardView.refreshView();
        return true;
    }

    /**
     * <p>Removes a product from the inventory.</p>
     * @param product The product to be removed.
     */
    public boolean removeProduct(ProductComponent product) {
        try {
            inventory.removeProduct(product);
        } catch (IllegalArgumentException e) {
            System.err.println(e.getMessage());
            return false;
        }

        sellerDashboardView.refreshView();
        return true;
    }

    /**
     * <p>Finds and returns a product from the inventory by its name.</p>
     * @param productName The name of the product to search for.
     * @return The ProductComponent with the matching name,
     * or null if not found.
     */
    public ProductComponent findProductByName(String productName) {
        Iterator<ProductComponent> it = inventory.iterator();
        while (it.hasNext()) {
            ProductComponent product = it.next();
            if (product.getName().equals(productName)) {
                return product;
            }
        }
        // Product not found
        return null;
    }

    /**
     * <p>Updates a product by replacing it in the Inventory</p>
     * @param newProduct The edited product.
     * @throws IllegalArgumentException If product is null or the product did
     * not previously exist in the Inventory.
     */
    public void updateProduct(ProductComponent newProduct) {
        if (newProduct == null) {
            throw new IllegalArgumentException("Product cannot be null");
        }

        ProductComponent oldProduct =
findProductByName(newProduct.getName());
        if (oldProduct == null) {
            throw new IllegalArgumentException(newProduct.getName() + " does
not exist");
        }

        try {
            Inventory.getInstance().setProduct(newProduct);
        } catch (IllegalArgumentException e) {
            System.err.println(e.getMessage());
        }

        inventory.notifyObservers();
    }

    /**
     * <p>Reduces the stock of a product by a set amount.</p>
     * @param product The product to subtract from.
     * @param quantity The amount to remove from the stock.
     * @return true if successful, false otherwise
     */
    public boolean reduceStock(ProductComponent product, int quantity) {
        try {
            product.reduceStockQuantity(quantity);
        } catch (IllegalArgumentException e) {
            System.err.println(e.getMessage());
            return false;
        }

        return true;
    }
}
```

LoginController.java:

```
package cop4331.controller;

import cop4331.model.Session;
import cop4331.model.User;
import cop4331.model.UserSystem;
import cop4331.model.customer.Customer;
import cop4331.model.seller.Seller;
import cop4331.view.customer.CustomerDashboardView;
import cop4331.view.seller.SellerDashboardView;

/**
 * <p>Handles the core logic for logging in, logging out, and creating new
accounts.</p>
 * @author Benjamin Carver
 */
public class LoginController {
    private UserSystem userSystem;
    private Session session;

    /**
     * <p>Constructs a LoginController object.</p>
     */
    public LoginController() {
        this.userSystem = UserSystem.getInstance();
        this.session = Session.getInstance();
    }

    /**
     * <p>Handles login by passing username and password to UserSystem for
verification
     * before redirecting back to the login page on a failed authentication
or to the
     * correct dashboard on success. Sets the Session.currentUser to
validated users.</p>
     * @param username The username provided by the user.
     * @param password The password provided by the user.
     */
    public void login(String username, String password) {
        User user = userSystem.verifyUser(username, password);

        if (user == null) {
            throw new IllegalArgumentException("Invalid username or
password");
        }

        session.setCurrentUser(user);
        if (user instanceof Customer) {
            new CustomerDashboardView((Customer) user);
        } else if (user instanceof Seller) {
            new SellerDashboardView((Seller) user);
        }
    }

    /**
     * <p>Logs the current user out and clears the session.</p>
     */
    public void logout() {
        session.invalidate();
    }

    /**
     * <p>Creates a new Customer with the provided information.</p>
     * @param username The new account's username.
     * @param password The new account's password.
     */
    public void createCustomerAccount(String username, String password) {
        userSystem.registerCustomer(username, password);
    }

    /**
     * <p>Creates a new Seller with the provided information.</p>
     * @param username The new account's username.
     * @param password The new account's password.
     */
    public void createSellerAccount(String username, String password) {
        userSystem.registerSeller(username, password);
    }
}
```

ShoppingCartController.java

```
package cop4331.controller;

import cop4331.model.ProductComponent;
import cop4331.model.customer.ShoppingCart;
import cop4331.view.customer.CustomerDashboardView;

import javax.swing.*;

/**
 * <p>Controller class for facilitating user interaction and updating the
display
 * of the shopping cart.</p>
 *
 * @author Benjamin Carver
 */
public class ShoppingCartController {
    private CustomerDashboardView dashboardView;
    private ShoppingCart shoppingCart;

    /**
     * <p>Constructs a ShoppingCartController object.</p>
     * @param shoppingCart The ShoppingCart model object.
     * @param dashboardView The CustomerDashboard view object.
     */
    public ShoppingCartController(CustomerDashboardView dashboardView,
ShoppingCart shoppingCart) {
        this.dashboardView = dashboardView;
        this.shoppingCart = shoppingCart;
    }
```

```java
    /**
     * <p>Adds a product to the shopping cart and updates the view.</p>
     * @param product The product to be added.
     * @param quantity The amount of the product to add.
     */
    public void addProductToCart(ProductComponent product, int quantity) {
        if (product.getStockQuantity() >= quantity) {
            shoppingCart.addProduct(product, quantity);
            product.reduceStockQuantity(quantity);
            dashboardView.refreshView();
        } else {
            JOptionPane.showMessageDialog(null, "Insufficient stock for "
                    + product.getName(),"Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * <p>Removes a product from the shopping cart and updates the view.</p>
     * @param product The product to be removed.
     */
    public void removeProductFromCart(ProductComponent product) {
        product.setStockQuantity(product.getStockQuantity()
                + shoppingCart.getCartItems().get(product));
        shoppingCart.removeProduct(product);
        dashboardView.refreshView();
    }

    /**
     * <p>Clears the cart of all items and adds the held stock back to the
inventory.</p>
     */
    public void clearCart() {
        for (ProductComponent product : shoppingCart.getCartItems().keySet())
{
            product.setStockQuantity(product.getStockQuantity()
                    + shoppingCart.getCartItems().get(product));
        }
        shoppingCart.clear();
        dashboardView.refreshView();
    }

}
```

cop4331.model

Bundle.java

```
package cop4331.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * <p>Represents a collection of products bundled together and sold as a
single unit.
 * Implements the {@code ProductComponent} interface. Provides functionality
to manage
 * the bundled products.</p>
 * @author Denry Ormejuste
 * @author Benjamin Carver
 */
public class Bundle implements ProductComponent {
    private String name;
    private String type;
    private double discountPercentage;
    private List<ProductComponent> bundledProducts;
    private final String seller;

    /**
     * <p>Constructs a Bundle object with an empty list.</p>
     * @param name The name of the Bundle.
     * @param discountPercentage The discount applied to the Bundle.
     */
    public Bundle(String name, double discountPercentage) {
        this.name = name;
        this.type = "Bundle";
        this.discountPercentage = discountPercentage;
        bundledProducts = new ArrayList<>();
        // ProductComponents will only be created by the current session
user.
        // We can use the getCurrentUser method to implicitly assign the
product,
        // bundle, or discount to the Seller who created it.
        this.seller = Session.getInstance().getCurrentUser().getUsername();
    }

    /**
     * <p>Constructs a Bundle object with the provided list.</p>
     * @param name The name of the Bundle.
     * @param discountPercentage The discount applied to the Bundle.
     * @param productList A list of products to be bundled.
     */
    public Bundle(String name, double discountPercentage,
List<ProductComponent> productList) {
        this.name = name;
        this.type = "Bundle";
        this.discountPercentage = discountPercentage;
        bundledProducts = productList;
        this.seller = Session.getInstance().getCurrentUser().getUsername();
    }

    /**
     * <p>Gets the name of the bundle.</p>
     * @return The name of the bundle.
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * <p>Constructs the description of the bundle.</p>
     * @return The description of the bundle.
     */
    @Override
    public String getDescription() {
        StringBuilder description = new StringBuilder(name + " includes:\n");
        for (ProductComponent product : bundledProducts) {
            description.append("- ").append(product.getName()).append(": ")
                    .append(product.getDescription()).append("\n");
        }
        return description.toString();
    }

    /**
     * <p>Gets the type of the bundle. Always "Bundle".</p>
     * @return "Bundle"
     */
    public String getType() {
        return type;
    }

    /**
     * <p>Gets the minimum stock for the bundle by checking the stock values
of the items inside.</p>
     * @return The minimum stock value in the bundle.
     */
    @Override
    public int getStockQuantity() {
        int minStock = Integer.MAX_VALUE;
        for (ProductComponent product : bundledProducts) {
            if (product.getStockQuantity() < minStock) {
                minStock = product.getStockQuantity();
            }
        }

        return minStock;
    }

    /**
     * <p>Gets the sale price of the bundle by totaling the prices of its
constituent products
     * and applying the Bundle's discount.</p>
     * @return The sale price of the Bundle.
     */
    @Override
    public double getSalePrice() {
        double salePrice = 0;
        for (ProductComponent product : bundledProducts) {
            salePrice += product.getSalePrice();
        }

        // Apply the Bundle's discount by multiplying it by the reciprocal of
the percentage.
        salePrice *= (1 - discountPercentage);
        return salePrice;
    }

    /**
     * <p>Gets the invoice price of the bundle by totaling the invoice prices
of its
     * constituent products.</p>
     * @return The total invoice price of the Bundle.
     */
    @Override
    public double getInvoicePrice() {
        double invoicePrice = 0;
        for (ProductComponent product : bundledProducts) {
            invoicePrice += product.getInvoicePrice();
        }

        return invoicePrice;
    }

    /**
     * <p>Gets the seller of the bundle's username.</p>
     * @return The bundle seller's username.
     */
    @Override
    public String getSeller() {
        return seller;
    }

    /**
     * <p>Adds a product to the Bundle.</p>
     * @param product The product to be added.
     * @throws IllegalArgumentException If product is null or if product is
already in the Bundle.
     */
    public void addProductToBundle(Product product) {
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null");
        }

        if (bundledProducts.contains(product)) {
            throw new IllegalArgumentException("Product is already in the
bundle");
        }

        bundledProducts.add(product);
    }

    /**
```

```
     * <p>Removes a product from the Bundle.</p>
     * @param product The product to be removed.
     * @throws IllegalArgumentException If product is null or if product is
not found in the Bundle.
     */
    public void removeProductFromBundle(Product product) {
        if (product == null) {
            throw new IllegalArgumentException("Cannot remove null
product.");
        }
        if (!bundledProducts.remove(product)) {
            throw new IllegalArgumentException("Product is not in the
bundle");
        }
    }

    /**
     * <p>Sets the stock quantity of the bundle to a specific value.</p>
     * @param stockQuantity The new stock quantity.
     */
    @Override
    public void setStockQuantity(int stockQuantity) {
        if (stockQuantity < 0) {
            throw new IllegalArgumentException("Quantity cannot be
negative");
        }

        int difference = this.getStockQuantity() - stockQuantity;

        for (ProductComponent product : bundledProducts) {
            product.reduceStockQuantity(difference);
        }
    }

    /**
     * <p>Removes stock quantity from the bundle.</p>
     * @param stockQuantity The new stock quantity.
     */
    @Override
    public void reduceStockQuantity(int stockQuantity) {
        if (stockQuantity < 0) {
            throw new IllegalArgumentException("Quantity cannot be
negative");
        }

        for (ProductComponent product : bundledProducts) {
            product.reduceStockQuantity(stockQuantity);
        }
    }
}
```

Command.java

```
package cop4331.model;

/**
 * <p>Interface for the Command design pattern used in the
SellerDashboardView's toolbar.</p>
 * @author Benjamin Carver
 */
public interface Command {
    void execute();
}
```

DiscountDecorator.java

```
package cop4331.model;

import java.io.Serializable;

/**
 * <p>An abstract class that serves as a decorator for applying discounts
onto
 * {@code ProductComponent} objects. Implements the {@code ProductComponent}
interface.</p>
 * @author Denis Ziegler
 * @author Benjamin Carver
 */
public abstract class DiscountDecorator implements ProductComponent {
    protected ProductComponent productComponent;
    protected String seller;

    /**
     * <p>Constructs a DiscountDecorator object.</p>
     * @param productComponent The product to be discounted.
     * @throws IllegalArgumentException If product is null.
     */
    public DiscountDecorator(ProductComponent productComponent) {
        if (productComponent == null) {
            throw new IllegalArgumentException("Product must not be null.");
        }
        this.productComponent = productComponent;
        // ProductComponents will only be created by the current session
user.
        // We can use the getCurrentUser method to implicitly assign the
product,
        // bundle, or discount to the Seller who created it.
        this.seller = Session.getInstance().getCurrentUser().getUsername();
    }

    /**
     * <p>Gets the base product from the discount.</p>
     * @return The base product.
     */
    public ProductComponent getProduct() {
        return productComponent;
    }
```

```java
    /**
     * <p>Gets the name of the base product.</p>
     * @return The name of the base product.
     */
    @Override
    public String getName() {
        return productComponent.getName();
    }

    /**
     * <p>Gets the description of the base product.</p>
     * @return The description of the base product.
     */
    @Override
    public String getDescription() {
        return productComponent.getDescription();
    }

    /**
     * <p>Gets the type of the base product.</p>
     * @return The type of the base product.
     */
    @Override
    public String getType() {
        return "Discounted " + productComponent.getType();
    }

    /**
     * <p>Gets the invoice price of the base product.</p>
     * @return The invoice price of the base product.
     */
    @Override
    public double getInvoicePrice() {
        return productComponent.getInvoicePrice();
    }

    /**
     * <p>Gets the stock quantity of the base product.</p>
     * @return The stock quantity of the base product.
     */
    @Override
    public int getStockQuantity() {
        return productComponent.getStockQuantity();
    }

    /**
     * <p>Gets the seller of the base product's username.</p>
     * @return The base product seller's username.
     */
    @Override
    public String getSeller() {
        return seller;
    }

    /**
     * <p>Sets the stock quantity of the item to a specified value.</p>
     * @param stockQuantity The quantity value to be set.
     * @throws IllegalArgumentException If stockQuantity is negative.
     */
    @Override
    public void setStockQuantity(int stockQuantity) {
        if (stockQuantity >= 0) {
            this.getProduct().setStockQuantity(stockQuantity);
        } else {
            throw new IllegalArgumentException("Stock quantity cannot be
negative.");
        }
    }

    /**
     * <p>Reduces the stock quantity by the specified amount.</p>
     * @param stockQuantity The amount to reduce the stock by.
     * @throws IllegalArgumentException If stockQuantity is negative.
     */
    @Override
    public void reduceStockQuantity(int stockQuantity) {
        if (stockQuantity >= 0) {
            this.getProduct().reduceStockQuantity(stockQuantity);
        } else {
            throw new IllegalArgumentException("Quantity cannot be
negative.");
        }
    }
}
```

FlatDiscount.java

```
package cop4331.model;

import java.io.Serializable;

/**
 * <p>Applies a flat discount to the price of a product. Extends the {@code
DiscountDecorator}
 * class.</p>
 * @author Denis Ziegler
 * @author Benjamin Carver
 */
public class FlatDiscount extends DiscountDecorator {
    private double discount;

    /**
     * <p>Constructs a FlatDiscount object.</p>
     * @param productComponent The product to be discounted.
     * @param discount The discount amount.
     * @throws IllegalArgumentException If discount is negative or exceeds
product's price.
     */
    public FlatDiscount(ProductComponent productComponent, double discount) {
        super(productComponent);
        if (discount <= 0) {
            throw new IllegalArgumentException("Discount cannot be
negative.");
        } else if (discount > productComponent.getSalePrice()) {
            throw new IllegalArgumentException("Discount must be less than
the product's sale price.");
        }
        this.discount = discount;
    }

    /**
     * <p>Gets the discount amount for the discount.</p>
     * @return The discount amount.
     */
    public double getDiscount() {
        return discount;
    }

    /**
     * <p>Sets a new discount amount.</p>
     * @param discount The new discount amount.
     * @throws IllegalArgumentException If discount is negative or exceeds
product's price.
     */
    public void setDiscount(double discount) {
        if (discount <= 0) {
            throw new IllegalArgumentException("Discount cannot be
negative.");
        } else if (discount > productComponent.getSalePrice()) {
            throw new IllegalArgumentException("Discount must be less than
the product's sale price.");
        }
        this.discount = discount;
    }

    /**
     * <p>Gets the discounted sale price of the product.</p>
     * @return The discounted sale price of the product.
     */
    @Override
    public double getSalePrice() {
        return productComponent.getSalePrice() - discount;
    }
}
```

PercentageDiscount.java

```java
package cop4331.model;

import java.io.Serializable;

/**
 * <p>Applies a percentage discount to the price of the product. Extends the
 * {@code DiscountDecorator} class.</p>
 * @author Denis Ziegler
 * @author Benjamin Carver
 */
public class PercentageDiscount extends DiscountDecorator {
    private double percentage;

    /**
     * <p>Constructs a PercentageDiscount object.</p>
     * @param productComponent The product to be discounted.
     * @param percentage The discount percentage.
     * @throws IllegalArgumentException If percentage is negative or
     *                                  if percentage is greater than 1.
     */
    public PercentageDiscount(ProductComponent productComponent, double
percentage) {
        super(productComponent);
        if (percentage < 0) {
            throw new IllegalArgumentException("Percentage cannot be
negative.");
        }
        if (percentage > 1) {
            throw new IllegalArgumentException("Percentage cannot be greater
than 1.");
        }
        this.percentage = percentage;
    }

    /**
     * <p>Gets the percentage of the discount.</p>
     * @return The discount percentage.
     */
    public double getPercentage() {
        return percentage;
    }

    /**
     * <p>Sets the percentage of the discount to a new value.</p>
     * @param percentage The new discount percentage.
     * @throws IllegalArgumentException If percentage is negative or
     *                                  if percentage is greater than 1.
     */
    public void setPercentage(double percentage) {
        if (percentage < 0) {
            throw new IllegalArgumentException("Percentage cannot be
negative.");
        }
        if (percentage > 1) {
            throw new IllegalArgumentException("Percentage cannot be greater
than 1.");
        }
        this.percentage = percentage;
    }

    /**
     * <p>Gets the discounted sale price of the Product.</p>
     * @return The discounted sale price.
     */
    @Override
    public double getSalePrice() {
        return productComponent.getSalePrice() * (1 - percentage);
    }
}
```

Inventory.java

```java
package cop4331.model;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

/**
 * <p>Represents the inventory of products managed by the system. Provides
functionality to add,
 * remove, update, and retrieve products. Implements the {@code Iterable} and
{@code Subject} interfaces.</p>
 * @author Benjamin Carver
 */
public class Inventory implements Iterable<ProductComponent>, Subject {
    private static Inventory instance;
    private HashMap<String, ProductComponent> products;
    private ArrayList<Observer> observers;

    /**
     * <p>Private constructor for the Inventory.</p>
     */
    private Inventory() {
        this.products = new HashMap<>();
        this.observers = new ArrayList<>();
    }

    /**
     * <p>Gets the current Inventory instance or creates one if
necessary.</p>
     * @return The Inventory instance.
     */
    public static synchronized Inventory getInstance() {
        if (instance == null) {
            instance = new Inventory();
        }

        return instance;
    }
    /**
     * <p>Sets the current Inventory instance for use in Serialization.</p>
     * @param inventory The Inventory instance to be loaded.
     */
    public static void setInstance(Inventory inventory) {
        instance = inventory;
    }

    /**
     * <p>Adds a product to the Inventory.</p>
     * @param product The product to be added.
     * @throws IllegalArgumentException If the product is null or already
exists in the Inventory.
     */
    public void addProduct(ProductComponent product) {
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null.");
        }
        if (products.containsKey(product.getName())) {
            throw new IllegalArgumentException("Product already exists.");
        }
        this.products.put(product.getName(), product);

        notifyObservers();
    }

    /**
     * <p>Removes a product from the Inventory.</p>
     * @param product The product to be removed.
     * @throws IllegalArgumentException If the product is null or does not
exist in the Inventory.
     */
    public void removeProduct(ProductComponent product) {
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null.");
        }
        if (!products.containsKey(product.getName())) {
            throw new IllegalArgumentException("Product does not exist.");
        }

        products.remove(product.getName());

        notifyObservers();
    }

    /**
     * <p>Sets a product in the Inventory to a new value allowing for editing
of product details
     * like prices and type.</p>
     * @param product The product to be put into the inventory.
     */
    public void setProduct(ProductComponent product) {
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null.");
        }
        if (!products.containsKey(product.getName())) {
```

```java
            throw new IllegalArgumentException("Product does not exist.");
        }

        products.remove(product.getName());
        products.put(product.getName(), product);

        notifyObservers();
    }

    /**
     * <p>Creates an {@code iterator} for use when searching through the
inventory.</p>
     * @return A new {@code Iterator}.
     */
    @Override
    public Iterator<ProductComponent> iterator() {
        return products.values().iterator();
    }

    /**
     * <p>Registers a new observer with the inventory.</p>
     * @param observer The new observer
     */
    @Override
    public void registerObserver(Observer observer) {
        observers.add(observer);
    }

    /**
     * <p>Removes the observer from the observers list.</p>
     * @param observer The observer to be removed.
     */
    @Override
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    /**
     * <p>Notifies all current observers of changes to the inventory.</p>
     */
    @Override
    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update();
        }
    }
}
```

Observer.java

```java
package cop4331.model;

/**
 * <p>Observer interface to facilitate inventory notifications in the
system.</p>
 */
```

```java
public interface Observer {
    void update();
}
```

Product.java

```java
package cop4331.model;

import java.io.Serializable;

/**
 * <p>Represents a product with attributes like name, type, description,
prices and quantity.</p>
 * <p>Implements the {@code ProductComponent} interface.</p>
 *
 * @author Denry Ormejuste
 * @author Benjamin Carver
 */
public class Product implements ProductComponent {
    private String name;
    private String type;
    private String description;
    private double salePrice;
    private double invoicePrice;
    private int stockQuantity;
    private String seller;

    /**
     * <p>Constructs a Product object.</p>
     * @param name The name of the Product.
     * @param type The type/category the Product belongs in.
     * @param description A description of the Product.
     * @param salePrice How much the Product is being sold for.
     * @param invoicePrice How much the Product cost the seller.
     * @param stockQuantity How much Product is in Inventory.
     * @throws IllegalArgumentException If salePrice, invoicePrice, or
stockQuantity is negative.
     */
    public Product(String name, String type, String description,
                   double salePrice, double invoicePrice, int stockQuantity)
{
        this.name = name;
        this.type = type;
        this.description = description;
        if (salePrice >= 0) {
            this.salePrice = salePrice;
        } else {
            throw new IllegalArgumentException("Sale price cannot be
negative.");
        }
        if (invoicePrice >= 0) {
            this.invoicePrice = invoicePrice;
        } else {
            throw new IllegalArgumentException("Invoice price cannot be
negative.");
        }
        if (stockQuantity >= 0) {
            this.stockQuantity = stockQuantity;
        } else {
            throw new IllegalArgumentException("Stock quantity cannot be
negative.");
        }
        // ProductComponents will only be created by the current session
user.
        // We can use the getCurrentUser method to implicitly assign the
product,
        // bundle, or discount to the Seller who created it.
        this.seller = Session.getInstance().getCurrentUser().getUsername();
    }

    // Getters

    /**
     * <p>Gets the name of the product.</p>
     * @return The name of the product.
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * <p>Gets the type of the product.</p>
     * @return The type of the product.
     */
    @Override
    public String getType() {
        return type;
    }

    /**
     * <p>Gets the description of the product.</p>
     * @return The description of the product.
     */
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * <p>Gets the sale price of the product.</p>
     * @return The sale price of the product.
     */
    @Override
    public double getSalePrice() {
        return salePrice;
    }

    /**
     * <p>Gets the invoice price of the product.</p>
     * @return The invoice price of the product.
     */
    public double getInvoicePrice() {
```

```java
        return invoicePrice;
    }

    /**
     * <p>Gets the stock quantity of the product.</p>
     * @return The stock quantity of the product.
     */
    @Override
    public int getStockQuantity() {
        return stockQuantity;
    }

    /**
     * <p>Gets the seller of the product's username.</p>
     * @return The product seller's username.
     */
    @Override
    public String getSeller() {
        return seller;
    }

    // Setters

    /**
     * <p>Sets the product's name to a new value.</p>
     * @param name The new name.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * <p>Sets the product's type to a new value.</p>
     * @param type The new type.
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     * <p>Sets the product's description to a new value.</p>
     * @param description The new description.
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * <p>Sets the sale price of the Product.</p>
     * @param salePrice The new sale price.
     * @throws IllegalArgumentException If salePrice is negative.
     */
    public void setSalePrice(double salePrice) {
        if (salePrice >= 0) {
            this.salePrice = salePrice;
        } else {
            throw new IllegalArgumentException("Sale price cannot be
negative.");
        }
    }

    /**
     * <p>Sets the invoice price of the Product.</p>
     * @param invoicePrice The new invoice price.
     * @throws IllegalArgumentException If invoicePrice is negative.
     */
    public void setInvoicePrice(double invoicePrice) {
        if (invoicePrice >= 0) {
            this.invoicePrice = invoicePrice;
        } else {
            throw new IllegalArgumentException("Invoice price cannot be 
negative.");
        }
    }

    /**
     * <p>Sets the stockQuantity of the Product.</p>
     * @param stockQuantity The new value for stockQuantity.
     * @throws IllegalArgumentException If stockQuantity is negative.
     */
    @Override
    public void setStockQuantity(int stockQuantity) {
        if (stockQuantity >= 0) {
            this.stockQuantity = stockQuantity;
        } else {
            throw new IllegalArgumentException("Stock quantity cannot be 
negative.");
        }
    }

    /**
     * <p>Removes the specified amount from the product stock quantity.</p>
     * @param stockQuantity The amount to remove from the stock quantity.
     * @throws IllegalArgumentException If quantity is invalid or greater 
than stock quantity.
     */
    @Override
    public void reduceStockQuantity(int stockQuantity) {
        if (stockQuantity >= 0 && stockQuantity <= this.stockQuantity) {
            this.stockQuantity -= stockQuantity;
        } else if (stockQuantity > this.stockQuantity) {
            throw new IllegalArgumentException("Quantity cannot be greater 
than the stock quantity.");
        } else {
            throw new IllegalArgumentException("Invalid stock quantity.");
        }
    }

    /**
     * <p>Checks if the product is in stock.</p>
     * @return true if one or more items are in stock, false otherwise.
     */
    public boolean isInStock() {
        return stockQuantity > 0;
    }

    /**
     * <p>Adds the specified quantity to the stockQuantity attribute of the
Product.</p>
     * @param quantity The amount to be added.
     * @throws IllegalArgumentException If quantity is negative or would
cause an integer overflow.
     */
    public void addStockQuantity(int quantity) {
        if (quantity < 0) {
            throw new IllegalArgumentException("Quantity to add cannot be
negative.");
        }

        if (Integer.MAX_VALUE - quantity < this.stockQuantity) {
            throw new IllegalArgumentException("Adding this quantity would
cause an integer overflow.");
        }

        this.stockQuantity += quantity;
    }
}
```

ProductComponent.java

```
package cop4331.model;

/**
 * <p>Interface for storing information regarding products, bundles, and
discounts.</p>
 * @author Denry Ormejuste
 * @author Benjamin Carver
 */
public interface ProductComponent {
    String getName();
    String getDescription();
    String getType();
    int getStockQuantity();
    double getSalePrice();
    double getInvoicePrice();
    String getSeller();

    void setStockQuantity(int quantity);
    void reduceStockQuantity(int quantity);
}
```

Session.java

```
package cop4331.model;

import java.io.Serializable;

/**
 * <p>Stores the current user.</p>
```

```java
 * @author Benjamin Carver
 */
public class Session implements Serializable {
    private static Session instance;
    private User currentUser;

    private Session() {
        currentUser = null;
    }

    /**
     * <p>Gets the current Session instance or creates one if it does not
exist.</p>
     * @return The Session object.
     */
    public static synchronized Session getInstance() {
        if (instance == null) {
            instance = new Session();
        }
        return instance;
    }

    /**
     * <p>Gets the current user.</p>
     * @return The current user.
     */
    public User getCurrentUser() {
        return currentUser;
    }

    /**
     * <p>Sets the current user.</p>
     * @param user The new current user.
     */
    public void setCurrentUser(User user) {
        currentUser = user;
    }

    /**
     * <p>Invalidates the current session.</p>
     */
    public void invalidate() {
        currentUser = null;
    }
}
```

Subject.java

```java
package cop4331.model;

/**
 * <p>Subject interface for publishers like the {@code Inventory}</p>
 * @code Benjamin Carver
 */
public interface Subject {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();

}
```

User.java

```
package cop4331.model;


import java.io.Serializable;

/**
 * <p>Abstract class for storing User information in the system.</p>
 * @author Jeremy Ladanowski
 * @author Benjamin Carver
 */
public abstract class User implements Serializable {
    private String username;
    private String password;

    /**
     * <p>Constructs a new User object.</p>
     * @param username The username of the new User.
     * @param password The password of the new User.
     */
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    /**
     * <p>Gets the username of the user.</p>
     * @return The username of the user.
     */
    public String getUsername() {
        return username;
    }

    public abstract String getRole();

    /**
     * <p>Validates the password for the current user.</p>
     * @param password The password to check.
     * @return true if password is valid, false otherwise.
     */
    public boolean validatePassword(String password) {
        return this.password.equals(password);
    }
}
```

UserSystem.java

```
package cop4331.model;

import cop4331.model.customer.Customer;
import cop4331.model.seller.Seller;

import java.util.ArrayList;
import java.util.HashMap;

/**
 * <p>The UserSystem stores all users registered. It facilitates the
creation,
 * removal, and retrieval of users.</p>
 * @author Benjamin Carver
 */
public class UserSystem {
    private static UserSystem instance;
    private HashMap<String, User> users;
    private ArrayList<Customer> customers;
    private ArrayList<Seller> sellers;

    /**
     * <p>Private constructor for the UserSystem.</p>
     */
    private UserSystem() {
        users = new HashMap<>();
        customers = new ArrayList<>();
        sellers = new ArrayList<>();
    }

    /**
     * <p>Gets the current UserSystem instance or creates it if it doesn't
exist.</p>
     * @return The current UserSystem instance.
     */
    public static synchronized UserSystem getInstance() {
        if (instance == null) {
            instance = new UserSystem();
        }
        return instance;
    }

    /**
     * <p>Sets the UserSystem instance for use in serialization.</p>
     * @param userSystem The instance to be loaded.
     */
    public static void setInstance(UserSystem userSystem) {
        instance = userSystem;
    }

    /**
     * <p>Gets the {@code HashMap} containing all registered users.</p>
     * @return A {@code HashMap} with all registered users.
     */
    public HashMap<String, User> getUsers() {
        return users;
```

```java
    }

    /**
     * <p>Gets the {@code ArrayList} containing all registered customers.</p>
     * @return An {@code ArrayList} containing all registered customers.
     */
    public ArrayList<Customer> getCustomers() {
        return customers;
    }

    /**
     * <p>Gets the {@code ArrayList} containing all registered customers.</p>
     * @return An {@code ArrayList} containing all registered customers.
     */
    public ArrayList<Seller> getSellers() {
        return sellers;
    }

    /**
     * <p>Registers a new customer in the system.</p>
     * @param username The new user's username.
     * @param password The new user's password.
     * @throws IllegalArgumentException If the username is already in use.
     */
    public void registerCustomer(String username, String password) {
        if (users.containsKey(username)) {
            throw new IllegalArgumentException("Username is already in use");
        }
        Customer customer = new Customer(username, password);
        users.put(username, customer);
        customers.add(customer);
    }

    /**
     * <p>Registers a new seller in the system.</p>
     * @param username The new user's username.
     * @param password The new user's password.
     * @throws IllegalArgumentException If the username is already in use.
     */
    public void registerSeller(String username, String password) {
        if (users.containsKey(username)) {
            throw new IllegalArgumentException("Username is already in use");
        }
        Seller seller = new Seller(username, password);
        users.put(username, seller);
        sellers.add(seller);
    }

    /**
     * <p>Removes the user from the system.</p>
     * @param username The username of the user to be removed.
     */
    public void removeUser(String username) {
        User user = users.remove(username);
        if (user != null) {
            if (user instanceof Customer) {
                customers.remove((Customer) user);
            } else if (user instanceof Seller) {
                sellers.remove((Seller) user);
            }
        }
    }

    /**
     * <p>Handles the user validation process.</p>
     * @param username The username to check.
     * @param password The password for the user.
     * @return The user if verification was successful, null otherwise.
     */
    public User verifyUser(String username, String password) {
        User user = users.get(username);
        if (user != null && user.validatePassword(password)) {
            return user;
        }

        return null;
    }
}
```

cop4331.model.customer

Customer.java

```java
package cop4331.model.customer;

import cop4331.model.User;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * <p>Represents a customer in the system. Extends the {@code User} class.
 * Holds information specific to customers such as their cart and previous
orders.</p>
 * @author Jeremy Ladanowski
 * @author Benjamin Carver
 */
public class Customer extends User {
    private ShoppingCart cart;
    private ArrayList<Order> orders;

    /**
     * <p>Constructs a Customer object with the specified username and
password.</p>
     * @param username The new Customer's username.
     * @param password The new Customer's password.
     */
    public Customer(String username, String password) {
        super(username, password);
        this.cart = new ShoppingCart();
        this.orders = new ArrayList<>();
    }
    /**
     * <p>Gets the Customer's cart.</p>
     * @return The Customer's cart.
     */
    public ShoppingCart getCart() {
        return cart;
    }

    /**
     * <p>Returns the role of the Customer for identification in the
system.</p>
     * @return "Customer"
     */
    @Override
    public String getRole() {
        return "Customer";
    }

    /**
     * <p>Gets the Customer's previous orders.</p>
     * @return An {@code ArrayList} of the Customer's orders.
     */
    public ArrayList<Order> getOrders() {
        return orders;
    }

    /**
     * <p>Adds an order to the Customer's order history.</p>
     * @param order The order to add.
     */
    public void addOrder(Order order) {
        this.orders.add(order);
    }
}
```

Order.java

```
package cop4331.model.customer;

import cop4331.model.ProductComponent;

import java.io.Serializable;
import java.util.HashMap;

/**
 * <p>Represents a {@code Customer}'s order. Contains information like the
products purchased,
 * the total price, and an invoice string for printing.</p>
 * @author Benjamin Carver
 */
public class Order {
    private Customer customer;
    private HashMap<ProductComponent, Integer> items;
    private double total;
    private String invoice;
```

```java
    /**
     * <p>Constructs a new Order object.</p>
     * @param customer The {@code Customer} the order belongs to.
     * @param items A {@code HashMap} containing the products bought and
their quantities.
     * @param total The total price of the order.
     * @param invoice A string representing the invoice.
     */
    public Order(Customer customer, HashMap<ProductComponent, Integer> items,
double total, String invoice) {
        this.customer = customer;
        this.items = items;
        this.total = total;
        this.invoice = invoice;
    }

    /**
     * <p>Gets the {@code Customer} the order belongs to.</p>
     * @return The {@code Customer} the order belongs to.
     */
    public Customer getCustomer() {
        return customer;
    }

    /**
     * <p>Gets the {@code HashMap} of all items in the order.</p>
     * @return The {@code HashMap} of all items in the order.
     */
    public HashMap<ProductComponent, Integer> getItems() {
        return items;
    }

    /**
     * <p>Gets the total price of the order.</p>
     * @return The total price of the order.
     */
    public double getTotal() {
        return total;
    }

    /**
     * <p>Gets the invoice string from the order.</p>
     * @return The invoice string of the order.
     */
    public String getInvoice() {
        return invoice;
    }

    /**
     * <p>Provides the invoice string as the representation of the order.</p>
     * @return The invoice string.
     */
    @Override
    public String toString() {
        return invoice;
    }
}
```

ShoppingCart.java

```java
package cop4331.model.customer;

import cop4331.model.ProductComponent;

import java.io.Serializable;
import java.util.HashMap;

/**
 * <p>Represents a shopping cart that holds products.</p>
 * @author Jeremy Ladanowski
 * @author Benjamin Carver
 */
public class ShoppingCart implements Serializable {
    private HashMap<ProductComponent, Integer> cartItems;

    /**
     * <p>Constructs a new ShoppingCart object.</p>
     */
    public ShoppingCart() {
        this.cartItems = new HashMap<>();
    }

    /**
     * <p>Gets the cart items from the Shopping Cart.</p>
     * @return A {@code HashMap} of cart items stored inside the
ShoppingCart.
     */
    public HashMap<ProductComponent, Integer> getCartItems() {
        return cartItems;
    }

    /**
     * <p>Adds a product to the ShoppingCart. If the product exists in the
cart, add the
     * quantities together.</p>
     * @param product The product to add.
     * @param quantity The amount of the product to add.
     * @throws IllegalArgumentException If quantity is less than or equal to
0 or
     *                                  if product is null.
     */
    public void addProduct(ProductComponent product, int quantity) {
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null.");
        }
        if (quantity <= 0) {
            throw new IllegalArgumentException("Quantity must be greater than
0.");
        }
        // If product is already in the cart, add the desired quantity to the
existing item.
        if (cartItems.containsKey(product)) {
            cartItems.put(product, cartItems.get(product) + quantity);
        }
        cartItems.put(product, quantity);
    }

    /**
     * <p>Removes a product from the shopping cart.</p>
     * @param product The product to remove.
     * @throws IllegalArgumentException If the product is null or
     */
    public void removeProduct(ProductComponent product) {
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null.");
        }
        if (!cartItems.containsKey(product)) {
            throw new IllegalArgumentException("Product does not exist in
shopping cart.");
        }

        cartItems.remove(product);
    }

    /**
     * <p>Updates the quantity of a product in the shopping cart. If the
desired quantity is 0,
     * the product is removed from the shopping cart.</p>
     * @param product The product whose quantity is changing.
     * @param quantity The new quantity of the product in the cart.
     */
    public void updateProductQuantity(ProductComponent product, int quantity)
{
        if (product == null) {
            throw new IllegalArgumentException("Product cannot be null.");
        }
        if (!cartItems.containsKey(product)) {
            throw new IllegalArgumentException("Product does not exist in
shopping cart.");
        }
        if (quantity < 0) {
            throw new IllegalArgumentException("Quantity cannot be
negative.");
        } else if (quantity == 0) {
            // Setting the quantity to 0 removes the product from the cart.
            this.removeProduct(product);
        } else {
            cartItems.put(product, quantity);
        }
    }

    /**
     * <p>Calculates the total sale price of all items in the cart.</p>
     * @return The total sale price of all items in the cart.
     */
    public double calculateTotal() {
        double total = 0.0;
        for (ProductComponent product : cartItems.keySet()) {
            total += product.getSalePrice() * cartItems.get(product);
        }

        return total;
    }

    /**
     * <p>Converts the items in the shopping cart and their quantities into a
string.
     * Used exclusively for debugging purposes.</p>
     * @return The string of the contents of the shopping cart.
     */
    @Override
    public String toString() {
       StringBuilder output = new StringBuilder();
       output.append("Shopping Cart:\n");
       for (ProductComponent product : cartItems.keySet()) {
           output.append("- ").append(product.getName()).append("\t\t|
$").append(product.getSalePrice())
                   .append(" x").append(cartItems.get(product)).append("\n");
       }
       output.append("Total: $").append(this.calculateTotal());

       return output.toString();
    }

    /**
     * <p>Gets the total quantity of items in the cart.</p>
     * @return The total quantity of items in the cart.
     */
    public int getTotalQuantity() {
        int total = 0;
        for (ProductComponent product : cartItems.keySet()) {
            total += cartItems.get(product);
        }
        return total;
    }

    /**
     * <p>Clears the items currently stored in the cart.</p>
     */
    public void clear() {
        this.cartItems.clear();
    }
}
```

cop4331.model.seller

Seller.java

```
package cop4331.model.seller;

import cop4331.model.User;

import java.io.Serializable;

/**
 * <p>Represents a Seller in the system. Extends the {@code User} class.
 * Holds information relating to sellers such as their inventory and
financial data.</p>
```

```
 * @author Jeremy Ladanowski
 * @author Benjamin Carver
 */
public class Seller extends User implements Serializable {
    private double totalRevenue;
    private double totalCost;

    /**
     * <p>Constructs a Seller object.</p>
     * @param username The username of the new Seller.
     * @param password The password of the new Seller.
     */
    public Seller(String username, String password) {
        super(username, password);
    }

    /**
     * <p>Gets the total revenue from the seller.</p>
     * @return The seller's total revenue.
     */
    public double getTotalRevenue() {
        return totalRevenue;
    }

    /**
     * <p>Gets the total cost incurred by the seller.</p>
     * @return The seller's total cost.
     */
    public double getTotalCost() {
        return totalCost;
    }

    /**
     * <p>Calculates the profit of the seller.</p>
     * @return The profit of the seller (revenue - cost).
     */
    public double getProfit() {
        return totalRevenue - totalCost;
    }

    /**
     * <p>Returns the role of the Seller for identification within the
system.</p>
     * @return "Seller"
     */
    @Override
    public String getRole() {
        return "Seller";
    }

    /**
     * <p>Updates the financial data stored within the seller.</p>
     * @param revenue The revenue to add.
     * @param cost The cost to add.
     */
    public void updateFinancialData(double revenue, double cost) {
        totalRevenue += revenue;
        totalCost += cost;
    }
}
```

SellerAddProductCommand.java

```
package cop4331.model.seller;

import cop4331.model.Command;
import cop4331.view.seller.SellerDashboardView;

/**
 * <p>Concrete command class used for opening the add product menu.
Implements
 * the {@code Command} interface.</p>
 * @author Benjamin Carver
 */
public class SellerAddProductCommand implements Command {
    private SellerDashboardView dashboardView;

    /**
     * <p>Constructs a new SellerAddProductCommand object.</p>
     * @param dashBoardView The seller's dashboard view
     */
    public SellerAddProductCommand(SellerDashboardView dashBoardView) {
        this.dashboardView = dashBoardView;
    }

    /**
     * <p>Executes the addProduct command when called.</p>
     */
    @Override
    public void execute() {
        dashboardView.addProduct();
    }
}
```

SellerBundleItemsCommand.java

```
package cop4331.model.seller;

import cop4331.model.Command;
import cop4331.view.seller.SellerDashboardView;

/**
 * <p>Concrete command class used for opening the bundle items menu.
Implements
 * the {@code Command} interface.</p>
 * @author Benjamin Carver
 */
public class SellerBundleItemsCommand implements Command {
    private SellerDashboardView dashboardView;

    /**
     * <p>Constructs a new SellerBundleItemsCommand object.</p>
     * @param dashboardView The seller's dashboard view
     */
    public SellerBundleItemsCommand(SellerDashboardView dashboardView) {
        this.dashboardView = dashboardView;
    }

    /**
     * <p>Executes the bundleItems command when called.</p>
     */
    @Override
    public void execute() {
        dashboardView.bundleItems();
    }
}
```

SellerLogoutCommand.java

```
package cop4331.model.seller;

import cop4331.model.Command;
import cop4331.view.seller.SellerDashboardView;

/**
 * <p>Concrete command class used for logging the seller out. Implements
 * the {@code Command} interface.</p>
 * @author Benjamin Carver
 */
public class SellerLogoutCommand implements Command {
    private SellerDashboardView dashboardView;

    /**
     * <p>Constructs a new SellerLogoutCommand object.</p>
     * @param dashboardView The seller's dashboard view
     */
    public SellerLogoutCommand(SellerDashboardView dashboardView) {
        this.dashboardView = dashboardView;
    }

    /**
     * <p>Executes the logout function when called.</p>
     */
    @Override
    public void execute() {
        dashboardView.logout();
    }
}
```

cop4331.view.customer

BuyItemView.java

```
package cop4331.view.customer;

import cop4331.controller.ShoppingCartController;
import cop4331.model.ProductComponent;

import javax.swing.*;
import javax.swing.text.NumberFormatter;
import java.awt.*;

/**
 * <p>GUI class for the view presented to customers when they wish to add
items to the cart.
 * Allows the customer to specify the quantity of the item they want or
cancel the action.</p>
 * @author Benjamin Carver
 */
public class BuyItemView extends JFrame {
    private CustomerDashboardView dashboardView;
    private ProductComponent productComponent;
    private JFormattedTextField quantityField;

    /**
     * <p>Constructs a BuyItemView object with the provided information.</p>
     * @param dashboardView The customer dashboard.
     * @param productComponent The product being added to the cart.
     */
    public BuyItemView(CustomerDashboardView dashboardView, ProductComponent
productComponent) {
        this.dashboardView = dashboardView;
        this.productComponent = productComponent;
        setTitle("Quantity to Purchase");
        setSize(300, 200);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new GridLayout(2, 2));

        panel.add(new JLabel("Quantity:", JLabel.CENTER));

        // Create a NumberFormatter for integers
        NumberFormatter intFormatter = new NumberFormatter();
        intFormatter.setValueClass(Integer.class);
        intFormatter.setMinimum(0);
        intFormatter.setAllowsInvalid(false);

        quantityField = new JFormattedTextField(intFormatter);
        panel.add(quantityField);
        quantityField.setValue(1);

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());
        panel.add(cancelButton);
```

```java
        JButton confirmButton = new JButton("Confirm");
        confirmButton.addActionListener(e -> confirmQuantity());
        panel.add(confirmButton);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Adds the product to the cart.</p>
     */
    private void confirmQuantity() {
        ShoppingCartController shoppingCartController =
dashboardView.getShoppingCartController();
        shoppingCartController.addProductToCart(productComponent,
                Integer.parseInt(quantityField.getValue().toString()));
        dashboardView.refreshView();
        dispose();
    }
}
```

CheckoutView.java

```java
package cop4331.view.customer;

import cop4331.controller.CheckoutController;
import cop4331.model.customer.Customer;
import cop4331.model.Session;
import cop4331.model.customer.ShoppingCart;

import javax.swing.*;
import java.awt.*;

/**
 * <p>Represents the checkout gui that shows when the customer is ready to
check out.
 * Allows the user the ability to review their order before proceeding with
payment.</p>
 * @author Benjamin Carver
 */
public class CheckoutView extends JFrame {
    private Customer customer;
    private ShoppingCart shoppingCart;
    private CheckoutController checkoutController;
    private JTextArea invoiceTextArea;

    /**
     * <p>Constructs the CheckoutView object with the given parameters.</p>
     * @param checkoutController The {@code CheckoutController} used in the
process.
     */
    public CheckoutView(CheckoutController checkoutController) {
        this.customer = (Customer) Session.getInstance().getCurrentUser();
        this.shoppingCart = customer.getCart();
        this.checkoutController = checkoutController;

        setTitle("Checkout");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create invoice display
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());

        JLabel titleLabel = new JLabel("Invoice", SwingConstants.CENTER);
        panel.add(titleLabel, BorderLayout.NORTH);

        invoiceTextArea = new JTextArea(10, 30);
        invoiceTextArea.setEditable(false);
        JScrollPane invoiceScrollPane = new JScrollPane(invoiceTextArea);
        panel.add(invoiceScrollPane, BorderLayout.CENTER);

        // Generate and display the invoice
        String invoice = checkoutController.generateInvoice(shoppingCart);
        invoiceTextArea.setText(invoice);

        // Create buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout());

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> cancelCheckout());
        buttonPanel.add(cancelButton);

        JButton proceedButton = new JButton("Proceed to Payment");
        proceedButton.addActionListener(e -> showPaymentWindow());
        buttonPanel.add(proceedButton);

        panel.add(buttonPanel, BorderLayout.SOUTH);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Cancels the checkout process.</p>
     */
    private void cancelCheckout() {
        dispose();
    }

    /**
     * <p>Shows the payment window.</p>
     */
    private void showPaymentWindow() {
        new PaymentWindow(this, customer, shoppingCart, checkoutController);
    }
}

/**
```

```
 * <p>Represents the GUI shown to the customer when payment details must be
provided.
 * Facilitates the collection of payment details and verifies them through
the {@code CheckoutController}.</p>
 * @author Benjamin Carver
 */
class PaymentWindow extends JFrame {
    private CheckoutView checkoutView;
    private Customer customer;
    private ShoppingCart shoppingCart;
    private CheckoutController checkoutController;
    private JTextField cardNumberField;
    private JTextField expiryDateField;
    private JTextField cvvField;
    private JLabel errorLabel;

    /**
     * <p>Creates the PaymentWindow object with the specified arguments.</p>
     * @param checkoutView The checkout view window.
     * @param customer The customer making the purchase.
     * @param shoppingCart The customer's shopping cart.
     * @param checkoutController The {@code CheckoutController}
     */
    public PaymentWindow(CheckoutView checkoutView, Customer customer,
ShoppingCart shoppingCart,
                         CheckoutController checkoutController) {
        this.checkoutView = checkoutView;
        this.customer = customer;
        this.shoppingCart = shoppingCart;
        this.checkoutController = checkoutController;

        setTitle("Enter Payment Details");
        setSize(300, 250);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new BorderLayout());

        JPanel checkoutPanel = new JPanel();
        checkoutPanel.setLayout(new GridLayout(4, 2, 10, 10));

        checkoutPanel.add(new JLabel("Card Number:"));
        cardNumberField = new JTextField();
        checkoutPanel.add(cardNumberField);

        checkoutPanel.add(new JLabel("Expiry Date (MM/YY):"));
        expiryDateField = new JTextField();
        checkoutPanel.add(expiryDateField);

        checkoutPanel.add(new JLabel("CVV:"));
        cvvField = new JTextField();
        checkoutPanel.add(cvvField);

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());
        checkoutPanel.add(cancelButton);
```

```java
        JButton submitButton = new JButton("Submit Payment");
        submitButton.addActionListener(e -> submitPayment());
        checkoutPanel.add(submitButton);

        JPanel errorPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        errorLabel = new JLabel("");
        errorLabel.setForeground(Color.RED);
        errorPanel.add(errorLabel);

        panel.add(checkoutPanel, BorderLayout.CENTER);
        panel.add(errorPanel, BorderLayout.SOUTH);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Submits the payment information to the {@code CheckoutController}
for finalization of the order.</p>
     */
    private void submitPayment() {
        String cardNumber = cardNumberField.getText();
        String expiryDate = expiryDateField.getText();
        String cvv = cvvField.getText();

        if (cardNumber.isEmpty() || expiryDate.isEmpty() || cvv.isEmpty()) {
            errorLabel.setText("Please fill out all fields.");
            return;
        }

        checkoutController.checkout(customer, shoppingCart, cardNumber,
expiryDate, cvv);
        JOptionPane.showMessageDialog(null, "Payment Successful");
        dispose();
        checkoutView.dispose();
    }
}
```

CustomerDashboardView.java

```java
package cop4331.view.customer;

import cop4331.controller.CheckoutController;
import cop4331.controller.InventoryController;
import cop4331.controller.LoginController;
import cop4331.controller.ShoppingCartController;
import cop4331.model.Inventory;
import cop4331.model.Observer;
import cop4331.model.ProductComponent;
import cop4331.model.customer.*;
import cop4331.view.login.LoginView;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * <p>Represents the main GUI on the customer-side of the application. Holds
a table
 * containing all products the customer may buy as well as access to order
history, the shopping cart,
 * and the checkout windows.</p>
 * @author Benjamin Carver
 */
public class CustomerDashboardView extends JFrame implements Observer {
    private Customer customer;
    private JTable inventoryTable;
    private Inventory inventory;
    private ShoppingCart shoppingCart;
    private ShoppingCartController shoppingCartController;
    private JLabel cartStatus;
    private CheckoutController checkoutController = new
CheckoutController(this);
    private InventoryController inventoryController = new
InventoryController(this);
    private ArrayList<ProductComponent> currentInventory = new ArrayList<>();

    /**
     * <p>Constructs a new CustomerDashboardView object.</p>
     * @param customer The current customer.
     */
    public CustomerDashboardView(Customer customer) {
        this.customer = customer;
        this.inventory = Inventory.getInstance();
        inventory.registerObserver(this);
        this.shoppingCart = customer.getCart();
        this.shoppingCartController = new ShoppingCartController(this,
shoppingCart);

        setTitle(customer.getUsername() + "'s Dashboard (Customer)");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create top bar
        JPanel topBar = getTopBar(customer);

        // Create table model
        updateCurrentInventory();

        String[] columnNames = {"Name", "Type", "Stock Quantity", "Price"};
        Object[][] data = new
Object[currentInventory.size()][columnNames.length];
        for (ProductComponent product : currentInventory) {
            data[currentInventory.indexOf(product)][0] = product.getName();
            data[currentInventory.indexOf(product)][1] = product.getType();
            if (product.getStockQuantity() > 0) {
                data[currentInventory.indexOf(product)][2] = "In Stock";
            } else {
                data[currentInventory.indexOf(product)][2] = "Out of Stock";
            }
            data[currentInventory.indexOf(product)][3] = "$" +
String.format("%.2f", product.getSalePrice());
        }

        inventoryTable = new JTable(new DefaultTableModel(data, columnNames)
{
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        });
        JScrollPane scrollPane = new JScrollPane(inventoryTable);

        inventoryTable.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                if (SwingUtilities.isRightMouseButton(e)) {
                    int row = inventoryTable.rowAtPoint(e.getPoint());
                    if (row >= 0) {
                        inventoryTable.setRowSelectionInterval(row, row);
                        showActionPopupMenu(e.getComponent(), e.getX(),
e.getY(), row);
                    }
                }
            }
        });

        cartStatus = new JLabel("Cart: 0 items", JLabel.CENTER);
        JButton viewCartButton = new JButton("View Cart");
        viewCartButton.addActionListener(e -> new ShoppingCartView(this));

        JPanel cartPanel = new JPanel(new GridLayout(1, 2));
        cartPanel.add(cartStatus);
        cartPanel.add(viewCartButton);


        add(topBar, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);
        add(cartPanel, BorderLayout.SOUTH);

        setVisible(true);
    }

    /**
     * <p>Listener for inventory updates. Triggers a refresh of the view.</p>
     */
    @Override
    public void update() {
        refreshView();
    }

    /**
     * <p>Updates the local inventory held by the dashboard.</p>
     */
    private void updateCurrentInventory() {
        currentInventory.clear();

        Iterator<ProductComponent> it = inventory.iterator();
        while (it.hasNext()) {
            currentInventory.add(it.next());
        }
    }

    /**
     * <p>Constructs the "topBar" section of the GUI.</p>
     * @param customer The current customer.
     * @return the topBar {@code JPanel}.
     */
    private JPanel getTopBar(Customer customer) {
        JPanel topBar = new JPanel();
        topBar.setLayout(new FlowLayout());

        JLabel welcomeLabel = new JLabel("Welcome, " +
customer.getUsername());
        topBar.add(welcomeLabel);

        JButton orderHistoryButton = new JButton("View Order History");
        orderHistoryButton.addActionListener(e -> showOrderHistory());
        topBar.add(orderHistoryButton);

        JButton logoutButton = new JButton("Logout");
        logoutButton.addActionListener(e -> logout());
        topBar.add(logoutButton);
        return topBar;
    }

    /**
     * <p>Gets the dashboard's checkoutController.</p>
     * @return the dashboard's checkoutController.
     */
    public CheckoutController getCheckoutController() {
        return checkoutController;
    }

    /**
     * <p>Gets the dashboard's inventoryController.</p>
     * @return the dashboard's inventoryController.
     */
    public InventoryController getInventoryController() {
        return inventoryController;
    }

    /**
     * <p>Gets the dashboard's shoppingCartController.</p>
     * @return the dashboard's shoppingCartController.
     */
    public ShoppingCartController getShoppingCartController() {
        return shoppingCartController;
    }

    /**
     * <p>Shows the order history window.</p>
     */
    private void showOrderHistory() {
        new OrderHistoryView(customer);
    }

    /**
     * <p>Logs the customer out and returns to the login page.</p>
     */
    public void logout() {
        LoginController loginController = new LoginController();
        loginController.logout();
        new LoginView();
        JOptionPane.showMessageDialog(this, "Logout successful");
        dispose();
    }

    /**
     * <p>Creates the right click menu and handles its functionality.</p>
     * @param component
     * @param x
     * @param y
     * @param row
     */
    private void showActionPopupMenu(Component component, int x, int y, int
row) {
        JPopupMenu popupMenu = new JPopupMenu();

        JMenuItem viewDetails = new JMenuItem("View Details");
        viewDetails.addActionListener(e -> showViewDetails(row));
        popupMenu.add(viewDetails);

        JMenuItem addToCart = new JMenuItem("Add to Cart");
        addToCart.addActionListener(e -> addItemToCart(row));
        popupMenu.add(addToCart);

        popupMenu.show(component, x, y);
    }

    /**
     * <p>Displays product details for the product.</p>
     * @param row The row the product exists on.
     */
    private void showViewDetails(int row) {
        ProductComponent product = inventoryController.findProductByName(
                (String) inventoryTable.getValueAt(row, 0));
        StringBuilder output = new StringBuilder();
        output.append("Product Details:\n");
        output.append("Name: ").append(product.getName()).append("\nType:
").append(product.getType())
                .append("\nDescription: ").append(product.getDescription());
        if (product.getStockQuantity() > 0) {
            output.append("\nStock Quantity:
").append(product.getStockQuantity());
        } else {
            output.append("\nStock Quantity: Out of Stock");
        }
        output.append("\nSale Price: $").append(String.format("%.2f",
product.getSalePrice()));

        JOptionPane.showMessageDialog(this, output.toString());
    }

    /**
     * <p>Adds the product in the row to the cart.</p>
     * @param row The row the product is in.
     */
    private void addItemToCart(int row) {
        ProductComponent product = inventoryController.findProductByName(
                (String) inventoryTable.getValueAt(row, 0));

        new BuyItemView(this, product);
        refreshView();
    }

    /**
     * <p>Refreshes the table to update any changes that occurred.</p>
     */
    public void refreshView() {
        updateCurrentInventory();

        String[] columnNames = {"Name", "Type", "Stock Quantity", "Price"};
        Object[][] data = new
Object[currentInventory.size()][columnNames.length];
        for (ProductComponent product : currentInventory) {
            data[currentInventory.indexOf(product)][0] = product.getName();
            data[currentInventory.indexOf(product)][1] = product.getType();
            if (product.getStockQuantity() > 0) {
                data[currentInventory.indexOf(product)][2] =
product.getStockQuantity();
            } else {
                data[currentInventory.indexOf(product)][2] = "Out of Stock";
            }
            data[currentInventory.indexOf(product)][3] = "$" +
String.format("%.2f", product.getSalePrice());
        }

        inventoryTable = new JTable(new DefaultTableModel(data, columnNames)
{
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        });

        cartStatus.setText("Cart: " + shoppingCart.getTotalQuantity() + "
items");

        revalidate();
        repaint();
    }
}
```

OrderHistoryView.java

```java
package cop4331.view.customer;

import cop4331.model.customer.Customer;
import cop4331.model.customer.Order;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

/**
 * <p>Represents the GUI popup for the order history. Displays all previous
orders the customer has made.</p>
 * @author Benjamin Carver
 */
public class OrderHistoryView extends JFrame {
    /**
     * <p>Creates a new OrderHistoryView object.</p>
     * @param customer The customer to view.
     */
    public OrderHistoryView(Customer customer) {
        setTitle(customer.getUsername() + "'s Order History");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        ArrayList<Order> orders = customer.getOrders();

        if (orders.isEmpty()) {
            JOptionPane.showMessageDialog(null, "No orders found");
            return;
        }

        JPanel mainPanel = new JPanel(new BorderLayout());

        JTextArea orderTextArea = new JTextArea(10, 30);
        orderTextArea.setEditable(false);

        for (Order order : orders) {
            orderTextArea.append(order.getInvoice() + "\n\n");
        }

        JScrollPane orderScrollPane = new JScrollPane(orderTextArea);
        mainPanel.add(orderScrollPane, BorderLayout.CENTER);

        add(mainPanel);
        setVisible(true);
    }
}
```

ShoppingCartView.java

```java
package cop4331.view.customer;

import cop4331.controller.InventoryController;
import cop4331.controller.ShoppingCartController;
import cop4331.model.customer.Customer;
import cop4331.model.ProductComponent;
import cop4331.model.Session;
import cop4331.model.customer.ShoppingCart;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Map;

/**
 * <p>View class for displaying the shopping cart and its contents.</p>
 * @author Benjamin Carver
 */
public class ShoppingCartView extends JFrame {
    private CustomerDashboardView dashboardView;
    private ShoppingCart shoppingCart;
    private JTable cartTable;
    private JLabel totalPriceLabel;

    /**
     * <p>Constructs the ShoppingCartView object.</p>
     * @param dashboardView The customer dashboard.
     */
    public ShoppingCartView(CustomerDashboardView dashboardView) {
        this.dashboardView = dashboardView;
        this.shoppingCart = ((Customer)
Session.getInstance().getCurrentUser()).getCart();
        setTitle("Shopping Cart");
        setSize(500,300);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create table model
        String[] columnNames = {"Name", "Quantity", "Price", "Total"};
        cartTable = new JTable(new DefaultTableModel(columnNames, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        });

        cartTable.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                if (SwingUtilities.isRightMouseButton(e)) {
                    int row = cartTable.rowAtPoint(e.getPoint());
                    if (row >= 0) {
                        cartTable.setRowSelectionInterval(row, row);
                        showActionPopupMenu(e.getComponent(), e.getX(),
e.getY(), row);
                    }
                }
            }
        });

        JScrollPane scrollPane = new JScrollPane(cartTable);

        JPanel mainPanel = new JPanel(new BorderLayout());
        mainPanel.add(scrollPane, BorderLayout.CENTER);

        JPanel bottomPanel = new JPanel(new FlowLayout());

        totalPriceLabel = new JLabel("Total Price: $0.00");
        bottomPanel.add(totalPriceLabel);

        JButton clearCartButton = new JButton("Clear Cart");
        clearCartButton.addActionListener(e -> clearCart());
        bottomPanel.add(clearCartButton);

        JButton checkoutButton = new JButton("Checkout");
        checkoutButton.addActionListener(e -> startCheckout());
        bottomPanel.add(checkoutButton);

        mainPanel.add(bottomPanel, BorderLayout.SOUTH);

        add(mainPanel);
        setVisible(true);

        updateCartDisplay();
    }

    /**
     * <p>Creates the right click menu.</p>
     * @param component
     * @param x
     * @param y
     * @param row
     */
    private void showActionPopupMenu(Component component, int x, int y, int
row) {
        JPopupMenu popupMenu = new JPopupMenu();

        JMenuItem removeFromCart = new JMenuItem("Remove from Cart");
        removeFromCart.addActionListener(e -> removeItemFromCart(row));
        popupMenu.add(removeFromCart);

        popupMenu.show(component, x, y);
    }

    /**
     * <p>Removes the item in the row from the cart.</p>
     * @param row The row the item is in.
     */
    private void removeItemFromCart(int row) {
        InventoryController inventoryController = new InventoryController();
        ProductComponent product = inventoryController.findProductByName(
                (String) cartTable.getValueAt(row, 0));

        ShoppingCartController shoppingCartController = new
ShoppingCartController(dashboardView, shoppingCart);
```

```
        shoppingCartController.removeProductFromCart(product);

        updateCartDisplay();
        dashboardView.refreshView();
    }

    /**
     * <p>Clears the cart's contents.</p>
     */
    private void clearCart() {
        ShoppingCartController shoppingCartController = new
ShoppingCartController(dashboardView, shoppingCart);
        shoppingCartController.clearCart();
        JOptionPane.showMessageDialog(null, "Cart has been cleared.");
        updateCartDisplay();
    }

    /**
     * <p>Updates the cart display.</p>
     */
    public void updateCartDisplay() {
        DefaultTableModel tableModel = (DefaultTableModel)
cartTable.getModel();
        tableModel.setRowCount(0);

        double totalPrice = 0.0;

        for (Map.Entry<ProductComponent, Integer> entry :
shoppingCart.getCartItems().entrySet()) {
            ProductComponent product = entry.getKey();
            int quantity = entry.getValue();
            double itemPrice = product.getSalePrice() * quantity;
            totalPrice += itemPrice;

            Object[] rowData = {
                    product.getName(),
                    quantity,
                    String.format("%.2f", product.getSalePrice()),
                    String.format("%.2f", itemPrice)
            };
            tableModel.addRow(rowData);
        }

        totalPriceLabel.setText("Total Price: $" + String.format("%.2f",
totalPrice));
    }

    /**
     * <p>Starts the checkout process.</p>
     */
    private void startCheckout() {
        new CheckoutView(dashboardView.getCheckoutController());
        updateCartDisplay();
        dispose();
    }
}
```

cop4331.view.login

CreateAccountView.java

```java
package cop4331.view.login;

import cop4331.controller.LoginController;

import javax.swing.*;
import java.awt.*;

/**
 * <p>Represents the window taking in information for account creation.</p>
 * @author Benjamin Carver
 */
public class CreateAccountView extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JPasswordField confirmPasswordField;
    private JComboBox<String> userTypeComboBox;
    private JButton createAccountButton;
    private JButton cancelButton;
    private JLabel errorLabel;

    private LoginController loginController;

    /**
     * <p>Creates the CreateAccountView object.</p>
     * @param loginController The {@code LoginController}.
     */
    public CreateAccountView(LoginController loginController) {
        this.loginController = loginController;
        setTitle("Create Account");
        setSize(350, 250);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(6, 2, 10, 10));

        panel.add(new JLabel("Username:", JLabel.CENTER));
        usernameField = new JTextField(16);
        panel.add(usernameField);

        panel.add(new JLabel("Password:", JLabel.CENTER));
        passwordField = new JPasswordField(16);
        panel.add(passwordField);
        panel.add(new JLabel("Confirm Password:", JLabel.CENTER));
        confirmPasswordField = new JPasswordField(16);
        panel.add(confirmPasswordField);

        panel.add(new JLabel("User Type:", JLabel.CENTER));
        userTypeComboBox = new JComboBox<>(new String[]{"Customer",
"Seller"});
        panel.add(userTypeComboBox);

        createAccountButton = new JButton("Create Account");
        createAccountButton.addActionListener(e -> createAccount());
        panel.add(createAccountButton);

        cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());
        panel.add(cancelButton);

        errorLabel = new JLabel("", JLabel.CENTER);
        errorLabel.setForeground(Color.RED);
        panel.add(errorLabel);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Takes data from the fields and creates a new account.</p>
     */
    private void createAccount() {
        String username = usernameField.getText();
        String password = String.valueOf(passwordField.getPassword());
        String confirmPassword =
String.valueOf(confirmPasswordField.getPassword());
        String userType = userTypeComboBox.getSelectedItem().toString();

        if (username.isEmpty() || password.isEmpty() ||
confirmPassword.isEmpty() || userType.isEmpty()) {
            errorLabel.setText("Please fill all the fields");
            return;
        }

        if (!password.equals(confirmPassword)) {
            errorLabel.setText("Passwords do not match");
            return;
        }

        try {
            if (userType.equals("Customer")) {
                loginController.createCustomerAccount(username, password);
            } else {
                loginController.createSellerAccount(username, password);
            }

            JOptionPane.showMessageDialog(this, "Account created
successfully");
            dispose();
        } catch (IllegalArgumentException e) {
            errorLabel.setText(e.getMessage());
        }
    }
}
```

LoginView.java

```
package cop4331.view.login;

import cop4331.controller.LoginController;

import javax.swing.*;
import java.awt.*;

/**
 * <p>Represents the loginView screen where users log into the system.</p>
 * @author Benjamin Carver
 */
public class LoginView extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private JButton createAccountButton;
    private JLabel errorLabel;

    private LoginController loginController;

    /**
     * <p>Creates a new LoginView object.</p>
     */
    public LoginView() {
        this.loginController = new LoginController();
        setTitle("Login");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(4, 2, 10, 10));

        panel.add(new JLabel("Username:", JLabel.CENTER));
        usernameField = new JTextField(16);
        panel.add(usernameField);

        panel.add(new JLabel("Password:", JLabel.CENTER));
        passwordField = new JPasswordField(16);
        panel.add(passwordField);

        loginButton = new JButton("Login");
        loginButton.addActionListener(e -> login());
        panel.add(loginButton);

        createAccountButton = new JButton("Create Account");
        createAccountButton.addActionListener(e ->
showCreateAccountWindow());
        panel.add(createAccountButton);

        errorLabel = new JLabel("", JLabel.CENTER);
        errorLabel.setForeground(Color.RED);
        panel.add(errorLabel);

        add(panel);
```

```java
        setVisible(true);
    }

    /**
     * <p>Starts the login process using the provided information.</p>
     */
    private void login() {
        try {
            String username = usernameField.getText();
            String password = String.valueOf(passwordField.getPassword());
            loginController.login(username, password);
            JOptionPane.showMessageDialog(this, "Login Successful",
                    "Login Successful", JOptionPane.INFORMATION_MESSAGE);
        } catch (IllegalArgumentException e) {
            errorLabel.setText(e.getMessage());
            return;
        }

        dispose();
    }

    /**
     * <p>Shows the createAccountWindow</p>
     */
    private void showCreateAccountWindow() {
        new CreateAccountView(loginController);
    }
}
```

cop4331.view.seller

AddItemView.java

```java
package cop4331.view.seller;

import cop4331.controller.InventoryController;
import cop4331.model.Product;

import javax.swing.*;
import javax.swing.text.NumberFormatter;
import java.awt.*;
import java.text.DecimalFormat;

/**
 * <p>Represents the seller GUI for adding new products</p>
 * @author Benjamin Carver
 */
public class AddItemView extends JFrame {
    private InventoryController inventoryController;
    private JTextField productNameField;
    private JTextField productTypeField;
    private JTextArea productDescriptionField;
    private JFormattedTextField productSalePriceField;
    private JFormattedTextField productInvoicePriceField;
    private JFormattedTextField productStockQuantityField;
    private JButton cancelButton;
    private JButton createProductButton;
    private JLabel errorLabel;

    /**
     * <p>Creates the AddItemView object.</p>
     * @param dashboardView The seller dashboard.
     */
    public AddItemView(SellerDashboardView dashboardView) {
        this.inventoryController = dashboardView.getInventoryController();

        setTitle("Add New Product");
        setSize(350, 400);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(8, 2, 10, 10));

        panel.add(new JLabel("Product Name: ", JLabel.RIGHT));
        productNameField = new JTextField(10);
        panel.add(productNameField);

        panel.add(new JLabel("Product Type: ", JLabel.RIGHT));
        productTypeField = new JTextField(10);
        panel.add(productTypeField);

        panel.add(new JLabel("Product Description: ", JLabel.RIGHT));
        productDescriptionField = new JTextArea(5, 10);
        panel.add(productDescriptionField);

        // Specify the price format
        DecimalFormat priceFormat = new DecimalFormat("#.##");
        priceFormat.setMinimumFractionDigits(2);
        priceFormat.setMaximumFractionDigits(2);

        // Create a NumberFormatter for doubles
        NumberFormatter priceFormatter = new NumberFormatter(priceFormat);
        priceFormatter.setValueClass(Double.class);
        priceFormatter.setAllowsInvalid(false);

        panel.add(new JLabel("Product Sale Price: $", JLabel.RIGHT));
        productSalePriceField = new JFormattedTextField(priceFormatter);
        productSalePriceField.setColumns(10);
        panel.add(productSalePriceField);

        panel.add(new JLabel("Product Invoice Price: $", JLabel.RIGHT));
        productInvoicePriceField = new JFormattedTextField(priceFormatter);
        productInvoicePriceField.setColumns(10);
        panel.add(productInvoicePriceField);

        // Create a NumberFormatter for integers
        NumberFormatter intFormatter = new NumberFormatter();
        intFormatter.setValueClass(Integer.class);
        intFormatter.setAllowsInvalid(false);

        panel.add(new JLabel("Product Stock Quantity: ", JLabel.RIGHT));
        productStockQuantityField = new JFormattedTextField(intFormatter);
        productStockQuantityField.setColumns(10);
        panel.add(productStockQuantityField);

        cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());
        panel.add(cancelButton);

        createProductButton = new JButton("Create Product");
        createProductButton.addActionListener(e -> createProduct());
        panel.add(createProductButton);

        errorLabel = new JLabel("");
        errorLabel.setForeground(Color.RED);
        panel.add(errorLabel);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Creates the product with the given information from the form.</p>
     */
    private void createProduct() {
        String productName = productNameField.getText();
        String productType = productTypeField.getText();
        String productDescription = productDescriptionField.getText();

        if (productName.isEmpty() || productType.isEmpty() ||
productDescription.isEmpty() ||
                productSalePriceField.getText().isEmpty() ||
productInvoicePriceField.getText().isEmpty() ||
                productStockQuantityField.getText().isEmpty()) {
            errorLabel.setText("Please fill all the fields.");
            return;
        }

        double productSalePrice =
Double.parseDouble(productSalePriceField.getText());
        double productInvoicePrice =
Double.parseDouble(productInvoicePriceField.getText());
        int productStockQuantity =
Integer.parseInt(productStockQuantityField.getText());

        if (productSalePrice < 0 || productInvoicePrice < 0 ||
productStockQuantity < 0) {
            errorLabel.setText("Prices and stock quantity cannot be
negative.");
        }

        try {
            Product newProduct = new Product(productName, productType,
productDescription,
                    productSalePrice, productInvoicePrice,
productStockQuantity);
            if (inventoryController.addProduct(newProduct)) {
                JOptionPane.showMessageDialog(this, "Product added
successfully.");
```

```
                dispose();
            } else {
                errorLabel.setText("Product already exists.");
            }
        } catch (IllegalArgumentException e) {
            errorLabel.setText(e.getMessage());
        }
    }
}
```

BundleItemsView.java

```
package cop4331.view.seller;

import cop4331.model.Bundle;
import cop4331.model.ProductComponent;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.text.NumberFormatter;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;

/**
 * <p>Represents the seller GUI for bundling items.
 * Allows sellers to choose from a list the products to bundle.</p>
 * @author Benjamin Carver
 */
public class BundleItemsView extends JFrame {
    private SellerDashboardView dashboardView;
    private JTextField bundleNameField;
    private JFormattedTextField percentDiscountField;
    private JTable productTable;
    private JLabel errorLabel;

    /**
     * <p>Creates a new BundleItemsView object.</p>
     * @param dashboardView The seller dashboard.
     */
    public BundleItemsView(SellerDashboardView dashboardView) {
        this.dashboardView = dashboardView;
        setTitle("Bundle Items");
        setSize(500, 200);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new BorderLayout());

        JPanel dataEntryPanel = getDataEntryPanel();
        panel.add(dataEntryPanel, BorderLayout.NORTH);

        List<ProductComponent> products = dashboardView.getSellerInventory();
        String[] columnNames = {"Name", "Type", "Sale Price"};
        Object[][] data = new Object[products.size()][columnNames.length];
        for (int i = 0; i < products.size(); i++) {
```

```java
            ProductComponent product = products.get(i);
            data[i][0] = product.getName();
            data[i][1] = product.getType();
            data[i][2] = product.getSalePrice();
        }

        productTable = new JTable(new DefaultTableModel(data, columnNames) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        });
        JScrollPane scrollPane = new JScrollPane(productTable);

        panel.add(scrollPane, BorderLayout.CENTER);

        JPanel buttonPanel = getButtonPanel();
        panel.add(buttonPanel, BorderLayout.SOUTH);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Creates the button panel.</p>
     * @return The buttonPanel {@code JPanel}
     */
    private JPanel getButtonPanel() {
        JPanel buttonPanel = new JPanel(new FlowLayout());
        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());

        JButton bundleButton = new JButton("Bundle Selected Items");
        bundleButton.addActionListener(e -> bundleSelectedItems());

        errorLabel = new JLabel("");
        errorLabel.setForeground(Color.RED);

        buttonPanel.add(errorLabel);
        buttonPanel.add(cancelButton);
        buttonPanel.add(bundleButton);
        return buttonPanel;
    }

    /**
     * <p>Creates the dataEntryPanel.</p>
     * @return The dataEntryPanel {@code JPanel}
     */
    private JPanel getDataEntryPanel() {
        JPanel dataEntryPanel = new JPanel(new GridLayout(2, 2, 10, 10));
        dataEntryPanel.add(new JLabel("Bundle Name: "));
        bundleNameField = new JTextField(10);
        dataEntryPanel.add(bundleNameField);
        dataEntryPanel.add(new JLabel("Discount Percentage: %"));

        // Create an integer input field
        NumberFormatter intFormatter = new NumberFormatter();
        intFormatter.setValueClass(Integer.class);
        intFormatter.setMinimum(0);
        intFormatter.setMaximum(100);
        intFormatter.setAllowsInvalid(false);

        percentDiscountField = new JFormattedTextField(intFormatter);
        dataEntryPanel.add(percentDiscountField);
        return dataEntryPanel;
    }

    /**
     * <p>Bundles the items selected by the seller.</p>
     */
    private void bundleSelectedItems() {
        try {
            int[] selectedRows = productTable.getSelectedRows();
            String bundleName = bundleNameField.getText();
            int percentDiscount =
Integer.parseInt(percentDiscountField.getText());
            double convertedDiscount = percentDiscount / 100.0;

            if (bundleName.isEmpty()) {
                errorLabel.setText("Bundle name cannot be empty");
                return;
            }

            List<ProductComponent> selectedItems = new ArrayList<>();
            for (int row : selectedRows) {
                String productName = productTable.getValueAt(row,
0).toString();
                ProductComponent product =
dashboardView.getInventoryController().findProductByName(productName);
                if (product == null) {
                    errorLabel.setText("Product " + productName + " not
found");
                    return;
                }
                selectedItems.add(product);
            }
            if (selectedItems.isEmpty()) {
                errorLabel.setText("No items selected");
                return;
            }

            Bundle bundle = new Bundle(bundleName, convertedDiscount,
selectedItems);
            dashboardView.getInventoryController().addProduct(bundle);
            dashboardView.refreshView();
            JOptionPane.showMessageDialog(null, "Bundle created
successfully.");
            dispose();
        } catch (NumberFormatException e) {
            errorLabel.setText("Invalid discount percentage");
        }
    }
}
```

DiscountView.java

```
package cop4331.view.seller;

import cop4331.model.FlatDiscount;
import cop4331.model.PercentageDiscount;
import cop4331.model.ProductComponent;

import javax.swing.*;
import java.awt.*;
import java.text.NumberFormat;

/**
 * <p>Represents the Discount product GUI for sellers.</p>
 * @author Benjamin Carver
 */
public class DiscountView extends JFrame {
    private SellerDashboardView dashboardView;
    private ProductComponent product;
    private JComboBox<String> discountTypeComboBox;
    private JFormattedTextField discountAmountField;
    private JLabel errorLabel;

    /**
     * <p>Creates the DiscountView object.</p>
     * @param dashboardView the seller dashboard.
     * @param product the product being discounted.
     */
    public DiscountView(SellerDashboardView dashboardView, ProductComponent 
product) {
        this.dashboardView = dashboardView;
        this.product = product;
        setTitle("Apply Discount");
        setSize(300, 250);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new BorderLayout());

        JPanel dataInputPanel = new JPanel(new GridLayout(4, 2, 10, 10));
        dataInputPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 
20));

        dataInputPanel.add(new JLabel("Product:"));
        dataInputPanel.add(new JLabel(product.getName()));

        dataInputPanel.add(new JLabel("Discount Type:"));
        discountTypeComboBox = new JComboBox<>(new String[] {"Percentage", 
"Flat"});
        dataInputPanel.add(discountTypeComboBox);

        NumberFormat numberFormat = NumberFormat.getInstance();
        numberFormat.setParseIntegerOnly(false);
        numberFormat.setMaximumFractionDigits(2);

        dataInputPanel.add(new JLabel("Discount Amount:"));
        discountAmountField = new JFormattedTextField(numberFormat);
        discountAmountField.setValue(0);
        dataInputPanel.add(discountAmountField);

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());
        dataInputPanel.add(cancelButton);

        JButton applyButton = new JButton("Apply Discount");
        applyButton.addActionListener(e -> applyDiscount());
        dataInputPanel.add(applyButton);

        panel.add(dataInputPanel, BorderLayout.CENTER);

        errorLabel = new JLabel("");
        errorLabel.setForeground(Color.RED);
        panel.add(errorLabel, BorderLayout.SOUTH);

        add(panel);
        setVisible(true);
    }

    /**
     * <p>Applies the discount to the specifications provided in the
fields.</p>
     */
    private void applyDiscount() {
        try {
            String discountType =
discountTypeComboBox.getSelectedItem().toString();
            double discountAmount =
Double.parseDouble(discountAmountField.getText());

            if (discountType.equals("Percentage")) {
                if (discountAmount <= 0 || discountAmount >= 100) {
                    errorLabel.setText("Percentage must be between 0 and
100.");
                    return;
                }
                product = new PercentageDiscount(product, discountAmount /
100);
            } else {
                if (discountAmount <= 0 || discountAmount >=
product.getSalePrice()) {
                    errorLabel.setText("Discount amount must be greater than
0 and " +
                            "less than the product's sale price.");
                    return;
                }
                product = new FlatDiscount(product, discountAmount);
            }

            dashboardView.getInventoryController().updateProduct(product);
            dashboardView.refreshView();
            JOptionPane.showMessageDialog(null, "Discount applied
successfully.");
            dispose();
        } catch (NumberFormatException e) {
```

```java
            errorLabel.setText("Invalid discount amount.");
        }
    }
}
```

EditItemView.java

```java
package cop4331.view.seller;

import cop4331.model.Bundle;
import cop4331.model.Product;
import cop4331.model.ProductComponent;

import javax.swing.*;
import javax.swing.text.NumberFormatter;
import java.awt.*;
import java.text.DecimalFormat;

/**
 * <p>Represents the seller GUI for editing items in their inventory.</p>
 * @author Benjamin Carver
 */
public class EditItemView extends JFrame {
    private SellerDashboardView dashboardView;
    private ProductComponent productComponent;
    private JTextField nameField;
    private JTextArea descriptionField;
    private JTextField typeField;
    private JFormattedTextField invoicePriceField;
    private JFormattedTextField salePriceField;
    private JFormattedTextField stockQuantityField;
    private JLabel errorLabel;

    /**
     * <p>Creates a new EditItemView object.</p>
     * @param dashboardView the seller dashboard.
     * @param productComponent the product being edited.
     */
    public EditItemView(SellerDashboardView dashboardView, ProductComponent
productComponent) {
        this.dashboardView = dashboardView;
        this.productComponent = productComponent;
        setTitle("Edit Product");
        setSize(400, 350);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        if (productComponent instanceof Bundle) {
            JOptionPane.showMessageDialog(null,"Editing bundles is not yet
supported.");
            return;
        }

        JPanel panel = new JPanel(new BorderLayout());

        // Construct the Data Entry Panel
        JPanel dataEntryPanel = new JPanel(new GridLayout(7, 2, 10, 10));
        dataEntryPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20,
20));

        dataEntryPanel.add(new JLabel("Name:"));
        nameField = new JTextField();
        nameField.setText(productComponent.getName());
        dataEntryPanel.add(nameField);

        dataEntryPanel.add(new JLabel("Description:"));
        descriptionField = new JTextArea(5, 10);
        descriptionField.setText(productComponent.getDescription());
        dataEntryPanel.add(descriptionField);

        dataEntryPanel.add(new JLabel("Type:"));
        typeField = new JTextField();
        typeField.setText(productComponent.getType());
        dataEntryPanel.add(typeField);

        // Specify the price format
        DecimalFormat priceFormat = new DecimalFormat("#.##");
        priceFormat.setMinimumFractionDigits(2);
        priceFormat.setMaximumFractionDigits(2);

        // Create a NumberFormatter for doubles
        NumberFormatter priceFormatter = new NumberFormatter(priceFormat);
        priceFormatter.setValueClass(Double.class);
        priceFormatter.setMinimum(0);
        priceFormatter.setAllowsInvalid(false);

        dataEntryPanel.add(new JLabel("Invoice Price: $"));
        invoicePriceField = new JFormattedTextField(priceFormatter);
        invoicePriceField.setValue(productComponent.getInvoicePrice());
        dataEntryPanel.add(invoicePriceField);

        dataEntryPanel.add(new JLabel("Sale Price: $"));
        salePriceField = new JFormattedTextField(priceFormatter);
        salePriceField.setValue(productComponent.getSalePrice());
        dataEntryPanel.add(salePriceField);

        // Create a NumberFormatter for integers
        NumberFormatter intFormatter = new NumberFormatter();
        intFormatter.setValueClass(Integer.class);
        intFormatter.setMinimum(0);
        intFormatter.setAllowsInvalid(false);

        dataEntryPanel.add(new JLabel("Stock Quantity:"));
        stockQuantityField = new JFormattedTextField(intFormatter);
        stockQuantityField.setValue(productComponent.getStockQuantity());
        dataEntryPanel.add(stockQuantityField);

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(e -> dispose());
        dataEntryPanel.add(cancelButton);

        JButton saveButton = new JButton("Save Changes");
        saveButton.addActionListener(e -> saveChanges());
        dataEntryPanel.add(saveButton);

        // Construct the error container
        JPanel errorPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        errorLabel = new JLabel("");
        errorLabel.setForeground(Color.RED);
        errorPanel.add(errorLabel);

        panel.add(dataEntryPanel, BorderLayout.CENTER);
        panel.add(errorPanel, BorderLayout.SOUTH);

        add(panel, BorderLayout.CENTER);
        setVisible(true);
    }

    /**
     * <p>Applies the changes made on the form.</p>
     */
    private void saveChanges() {
        try {
            Product newProduct = getNewProduct();


dashboardView.getInventoryController().removeProduct(productComponent);
            dashboardView.getInventoryController().addProduct(newProduct);
            dashboardView.refreshView();
            JOptionPane.showMessageDialog(this, "Product has been saved");
            dispose();
        } catch (NumberFormatException e) {
            errorLabel.setText("Please enter a valid number");
        } catch (IllegalArgumentException e) {
            errorLabel.setText("Invalid argument, please verify product name
is unique.");
        } catch (Exception e) {
            errorLabel.setText("Something went wrong");
        }
    }

    /**
     * <p>Handles the generation of the updated product.</p>
     * @return the updated product.
     */
    private Product getNewProduct() {
        String productName = nameField.getText();
        String productDescription = descriptionField.getText();
        String productType = typeField.getText();
        double productInvoicePrice =
Double.parseDouble(invoicePriceField.getText());
        double productSalePrice =
Double.parseDouble(salePriceField.getText());
        int productStockQuantity =
Integer.parseInt(stockQuantityField.getText());

        return new Product(productName, productType, productDescription,
                productSalePrice, productInvoicePrice, productStockQuantity);
    }
}
```

SellerDashboardView.java

```
package cop4331.view.seller;

import cop4331.controller.InventoryController;
import cop4331.controller.LoginController;
import cop4331.model.DiscountDecorator;
import cop4331.model.Inventory;
import cop4331.model.Observer;
import cop4331.model.ProductComponent;
import cop4331.model.seller.Seller;
import cop4331.model.seller.SellerAddProductCommand;
import cop4331.model.seller.SellerBundleItemsCommand;
import cop4331.model.seller.SellerLogoutCommand;
import cop4331.view.login.LoginView;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * <p>Represents the main GUI for the Seller side of the application.
Contains a Table with the
 * seller's inventory and functionality to access the financials and add/edit
products.</p>
 * @author Benjamin Carver
 */
public class SellerDashboardView extends JFrame implements Observer {
    private Seller seller;
    ArrayList<ProductComponent> sellerInventory = new ArrayList<>();
    private JTable inventoryTable;
    private InventoryController inventoryController = new
InventoryController(this);

    /**
     * <p>Creates a new SellerDashboardView object.</p>
     * @param seller The current seller.
     */
    public SellerDashboardView(Seller seller) {
        this.seller = seller;
        Inventory inventory = Inventory.getInstance();
        inventory.registerObserver(this);

        setTitle(seller.getUsername() + "'s Dashboard (Seller)");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        JToolBar toolbar = new JToolBar();

        JButton addButton = new JButton("Add Product");
        addButton.addActionListener(e -> new
SellerAddProductCommand(this).execute());
```

```
        toolbar.add(addButton);

        JButton bundleButton = new JButton("Bundle");
        bundleButton.addActionListener(e -> new
SellerBundleItemsCommand(this).execute());
        toolbar.add(bundleButton);

        JButton logoutButton = new JButton("Logout");
        logoutButton.addActionListener(e -> new
SellerLogoutCommand(this).execute());
        toolbar.add(logoutButton);

        JPanel topBar = getTopBar(seller);

        // Create table model
        updateSellerInventory();

        String[] columnNames = {"Name", "Type", "Quantity", "Invoice Price",
"Sale Price"};
        Object[][] data = new
Object[sellerInventory.size()][columnNames.length];
        for (ProductComponent product : sellerInventory) {
            data[sellerInventory.indexOf(product)][0] = product.getName();
            data[sellerInventory.indexOf(product)][1] = product.getType();
            if (product.getStockQuantity() > 0) {
                data[sellerInventory.indexOf(product)][2] =
product.getStockQuantity();
            } else {
                data[sellerInventory.indexOf(product)][2] = "Out of Stock";
            }
            data[sellerInventory.indexOf(product)][3] = "$" +
String.format("%.2f", product.getInvoicePrice());
            data[sellerInventory.indexOf(product)][4] = "$" +
String.format("%.2f", product.getSalePrice());
        }

        inventoryTable = new JTable(new DefaultTableModel(data, columnNames)
{
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        });

        JScrollPane scrollPane = new JScrollPane(inventoryTable);

        inventoryTable.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                if (SwingUtilities.isRightMouseButton(e)) {
                    int row = inventoryTable.rowAtPoint(e.getPoint());
                    if (row >= 0) {
                        inventoryTable.setRowSelectionInterval(row, row);
                        showActionPopupMenu(e.getComponent(), e.getX(),
e.getY(), row);
                    }
                }
            }
        });

        JPanel bottomBar = getBottomBar();

        JPanel northPanel = new JPanel(new BorderLayout());
        northPanel.add(toolbar, BorderLayout.NORTH);
        northPanel.add(topBar, BorderLayout.CENTER);

        add(northPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);
        add(bottomBar, BorderLayout.SOUTH);
        setVisible(true);
    }

    /**
     * <p>Listener for updates to the {@code Inventory} and triggers the
refreshView() method.</p>
     */
    @Override
    public void update() {
        refreshView();
    }

    /**
     * <p>Creates the topBar component of the dashboard.</p>
     * @param seller The current seller.
     * @return the topBar {@code JPanel}
     */
    private JPanel getTopBar(Seller seller) {
        JPanel topBar = new JPanel();
        topBar.setLayout(new GridLayout(1, 3));

        JLabel welcomeLabel = new JLabel("Welcome, " + seller.getUsername() +
"!", JLabel.CENTER);
        topBar.add(welcomeLabel);

        JButton financialsButton = new JButton("Financials");
        financialsButton.addActionListener(e -> showFinancialDetails());
        topBar.add(financialsButton);

        JButton logoutButton = new JButton("Logout");
        logoutButton.addActionListener(e -> logout());
        topBar.add(logoutButton);
        return topBar;
    }

    /**
     * <p>Creates the bottomBar component of the dashboard.</p>
     * @return the bottomBar {@code JPanel}
     */
    private JPanel getBottomBar() {
        JPanel bottomBar = new JPanel();
        bottomBar.setLayout(new FlowLayout());

        JButton addButton = new JButton("Add Product");
        addButton.addActionListener(e -> addProduct());
```

```java
        bottomBar.add(addButton);

        JButton bundleButton = new JButton("Bundle Items");
        bundleButton.addActionListener(e -> bundleItems());
        bottomBar.add(bundleButton);

        JButton removeButton = new JButton("Remove Selected Items");
        removeButton.addActionListener(e -> removeItems());
        bottomBar.add(removeButton);
        return bottomBar;
    }

    /**
     * <p>Displays the seller financials window.</p>
     */
    public void showFinancialDetails() {
        new SellerFinancialsView();
    }

    /**
     * <p>Logs the seller out and returns to the login screen.</p>
     */
    public void logout() {
        LoginController loginController = new LoginController();
        loginController.logout();
        new LoginView();
        JOptionPane.showMessageDialog(this, "Logout successful");
        dispose();
    }

    /**
     * <p>Creates the right click menu.</p>
     * @param component
     * @param x
     * @param y
     * @param row
     */
    private void showActionPopupMenu(Component component, int x, int y, int
row) {
        JPopupMenu popupMenu = new JPopupMenu();

        JMenuItem viewItem = new JMenuItem("View Details");
        viewItem.addActionListener(e -> showViewDetails(row));
        popupMenu.add(viewItem);

        JMenuItem editItem = new JMenuItem("Edit Details");
        editItem.addActionListener(e -> showEditDetails(row));
        popupMenu.add(editItem);

        JMenuItem discountItem = new JMenuItem("Discount Item");
        discountItem.addActionListener(e -> showDiscountOptions(row));
        popupMenu.add(discountItem);

        JMenuItem removeDiscount = new JMenuItem("Remove Discount");
        removeDiscount.addActionListener(e -> removeDiscount(row));
        popupMenu.add(removeDiscount);
        JMenuItem removeItem = new JMenuItem("Remove Item");
        removeItem.addActionListener(e -> removeItem(row));
        popupMenu.add(removeItem);

        popupMenu.show(component, x, y);
    }

    /**
     * <p>Updates the local seller inventory by scanning for products
assigned to the seller.</p>
     */
    public void updateSellerInventory() {
        sellerInventory.clear();

        Iterator<ProductComponent> it = Inventory.getInstance().iterator();
        while (it.hasNext()) {
            ProductComponent product = it.next();
            if (product.getSeller() != null &&
                    product.getSeller().equals(seller.getUsername())) {
                sellerInventory.add(product);
            }
        }
    }

    /**
     * <p>Refreshes the seller dashboard to change with any updates.</p>
     */
    public void refreshView() {
        updateSellerInventory();

        String[] columnNames = {"Name", "Type", "Quantity", "Invoice Price",
"Sale Price"};
        Object[][] data = new
Object[sellerInventory.size()][columnNames.length];
        for (ProductComponent product : sellerInventory) {
            data[sellerInventory.indexOf(product)][0] = product.getName();
            data[sellerInventory.indexOf(product)][1] = product.getType();
            if (product.getStockQuantity() > 0) {
                data[sellerInventory.indexOf(product)][2] =
product.getStockQuantity();
            } else {
                data[sellerInventory.indexOf(product)][2] = "Out of Stock";
            }
            data[sellerInventory.indexOf(product)][3] = "$" +
String.format("%.2f", product.getInvoicePrice());
            data[sellerInventory.indexOf(product)][4] = "$" +
String.format("%.2f", product.getSalePrice());
        }

        inventoryTable.setModel(new DefaultTableModel(data, columnNames) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        });

        revalidate();
        repaint();
    }

    /**
     * <p>Displays the view details window for a product.</p>
     * @param row The row the product is in.
     */
    private void showViewDetails(int row) {
        ProductComponent product = sellerInventory.get(row);
        StringBuilder output = new StringBuilder();
        output.append("Product Details:\n");
        output.append("Name: ").append(product.getName()).append("\nType:
").append(product.getType())
                .append("\nDescription: ").append(product.getDescription());
        if (product.getStockQuantity() > 0) {
            output.append("\nStock Quantity:
").append(product.getStockQuantity());
        } else {
            output.append("\nStock Quantity: Out of Stock");
        }
        output.append("\nSale Price: $").append(String.format("%.2f",
product.getSalePrice()));
        output.append("\nInvoice Price: $").append(String.format("%.2f",
product.getInvoicePrice()));

        JOptionPane.showMessageDialog(this, output.toString());
    }

    /**
     * <p>Gets the {@code InventoryController} for the dashboard.</p>
     * @return the {@code InventoryController} for the dashboard.
     */
    public InventoryController getInventoryController() {
        return inventoryController;
    }

    /**
     * <p>Gets the sellerInventory {@code ArrayList}.</p>
     * @return the sellerInventory {@code ArrayList}.
     */
    public ArrayList<ProductComponent> getSellerInventory() {
        return sellerInventory;
    }

    /**
     * <p>Creates the editDetails window for the selected product.</p>
     * @param row The row the product is in.
     */
    private void showEditDetails(int row) {
        ProductComponent product = sellerInventory.get(row);
        new EditItemView(this, product);
    }

    /**
     * <p>Creates the discountOptions window for the selected product.</p>
     * @param row The row the product is in.
     */
```

```java
    private void showDiscountOptions(int row) {
        ProductComponent product = sellerInventory.get(row);
        new DiscountView(this, product);
    }

    /**
     * <p>Removes the discount from the item.</p>
     * @param row the row the item is in.
     */
    private void removeDiscount(int row) {
        ProductComponent product = sellerInventory.get(row);
        if (product instanceof DiscountDecorator) {
            ProductComponent newProduct = ((DiscountDecorator)
product).getProduct();
            inventoryController.updateProduct(newProduct);
            refreshView();
            JOptionPane.showMessageDialog(this, "Discount removed.");
        } else {
            JOptionPane.showMessageDialog(this, "Product does not have a
discount");
        }
    }

    /**
     * <p>Shows the addProduct window.</p>
     */
    public void addProduct() {
        new AddItemView(this);
    }

    /**
     * <p>Removes the item from the selected row.</p>
     * @param row The row to remove the item from.
     */
    private void removeItem(int row) {
        ProductComponent product = sellerInventory.get(row);
        int result = JOptionPane.showConfirmDialog(this,
                "Are you sure you want to delete this item?", "Delete Item",
                JOptionPane.YES_NO_OPTION);

        if (result == JOptionPane.YES_OPTION) {
            try {
                inventoryController.removeProduct(product);
                refreshView();
            } catch (IllegalArgumentException e) {
                System.err.println(e.getMessage());
            }
        }
    }

    /**
     * <p>Shows the bundleItems window.</p>
     */
    public void bundleItems() {
        new BundleItemsView(this);
    }
    /**
     * <p>Shows the removeItems window.</p>
     */
    private void removeItems() {
        JOptionPane.showMessageDialog(this, "Remove item placeholder.");
    }
}
```

SellerFinancialsView.java

```
package cop4331.view.seller;

import cop4331.model.seller.Seller;
import cop4331.model.Session;

import javax.swing.*;
import java.awt.*;

public class SellerFinancialsView extends JFrame {
    private Seller seller;
    private JLabel revenueLabel;
    private JLabel costLabel;
    private JLabel profitLabel;

    SellerFinancialsView() {
        this.seller = (Seller) Session.getInstance().getCurrentUser();

        setTitle("Financial Overview");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);

        JPanel panel = new JPanel(new BorderLayout());

        JPanel financialsPanel = new JPanel(new GridLayout(3, 2));

        financialsPanel.add(new JLabel("Total Revenue: ", JLabel.RIGHT));
        revenueLabel = new JLabel("$" + String.format("%.2f",
seller.getTotalRevenue()), JLabel.LEFT);
        revenueLabel.setForeground(Color.GREEN);
        financialsPanel.add(revenueLabel);

        financialsPanel.add(new JLabel("Total Costs: ", JLabel.RIGHT));
        costLabel = new JLabel("-$" + String.format("%.2f",
seller.getTotalCost()), JLabel.LEFT);
        costLabel.setForeground(Color.RED);
        financialsPanel.add(costLabel);

        financialsPanel.add(new JLabel("Profit: ", JLabel.RIGHT));
        if (seller.getProfit() < 0) {
            // Need to multiply the profit by -1 to remove a redundant '-'
from appearing
            // on the wrong side of the '$' when formatted.
            profitLabel = new JLabel("-$" + String.format("%.2f", -1 *
seller.getProfit()), JLabel.LEFT);
            profitLabel.setForeground(Color.RED);
        } else {
            profitLabel = new JLabel("$" + String.format("%.2f",
seller.getProfit()), JLabel.LEFT);
            profitLabel.setForeground(Color.GREEN);
        }
        financialsPanel.add(profitLabel);

        JButton closeButton = new JButton("Close");
        closeButton.addActionListener(e -> dispose());

        panel.add(financialsPanel, BorderLayout.CENTER);
        panel.add(closeButton, BorderLayout.SOUTH);

        add(panel, BorderLayout.CENTER);
        setVisible(true);
    }
}
```