Pharmacy Management System

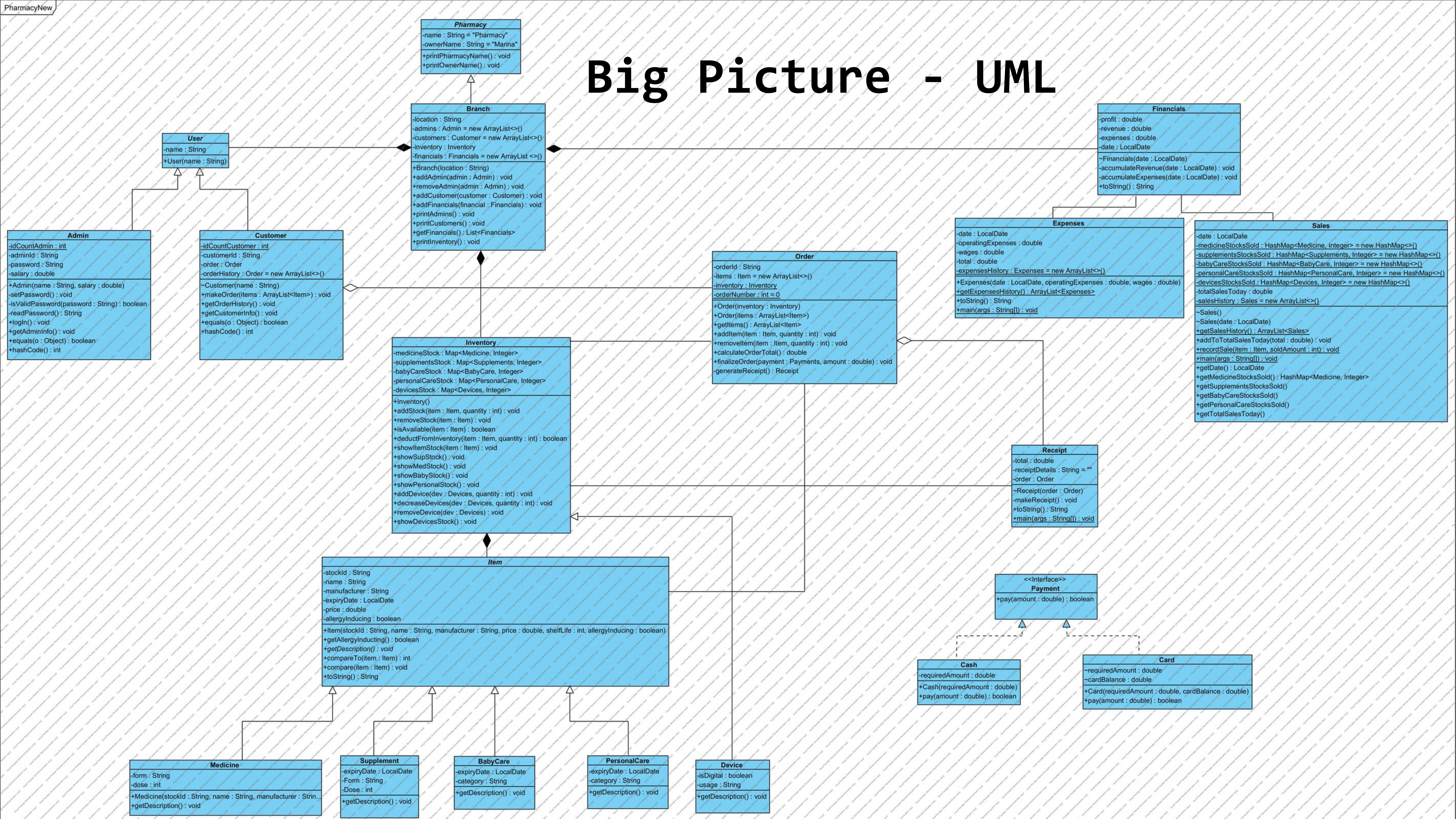
Team 19

Role of Each Member

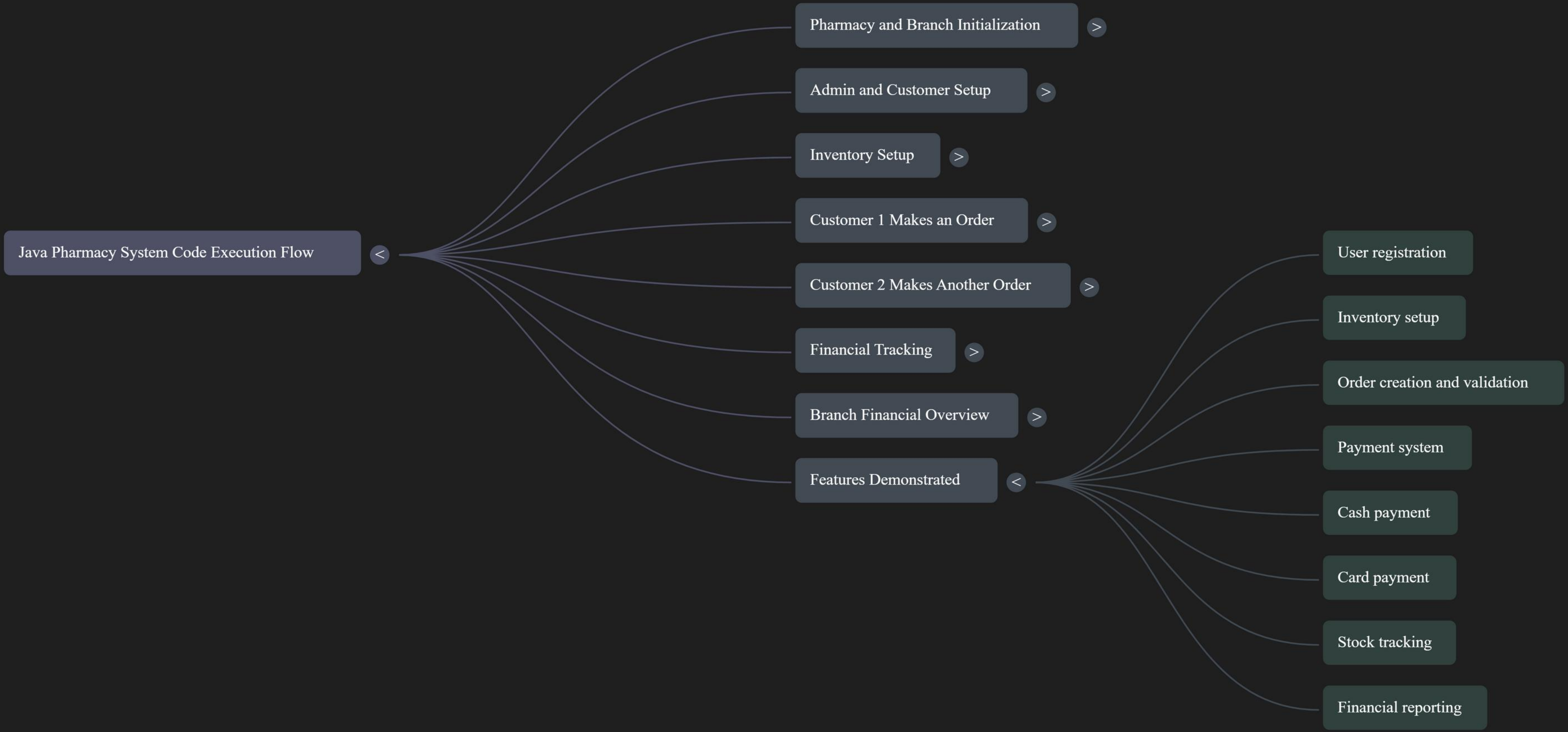
Member	Classes Handled	Extra Role
Marina Bebawy Nasr 2200826	(Abstract)Pharmacy ⟵ Branch (Abstract)User ⟵ Admin, Customer GUI - Receipt Generation	UML Planning and Layout
Karim Mohammed Elsayed 2200746	Inventory (Abstract)Item ⟵ Medicine, Supplements, personalCare, babyCare, Devices	GUI Event Handling JUnit Testing
Karim Khaled Gamaleldin 2201356	Financials, Sales, Expenses (Interface) Payments ⟵ Cash, Card	Main Class
Menna Ayman Hassan 2200236	Order, Receipt, GUI Login Page, GUI Shop Page	GUI Main Scenes

Table of Contents

Big Picture - UML


PharmacyNew
Pharmacy
-name : String = "Pharmacy"
-ownerName : String = "Marina"
+printPharmacyName() : void
+printOwnerName() : void
Branch
-location : String
-admins : Admin = new ArrayList<>()
-customers : Customer = new ArrayList<>()
-inventory : Inventory
-financials : Financials = new ArrayList <>()
+Branch(location : String)
+addAdmin(admin : Admin) : void
+removeAdmin(admin : Admin) : void
+addCustomer(customer : Customer) : void
+addFinancials(financial : Financials) : void
+printAdmins() : void
+printCustomers() : void
+getFinancials() : List<Financials>
+printInventory() : void
User
-name : String
+User(name : String)
Admin
-idCountAdmin : int
-adminId : String
-password : String
-salary : double
+Admin(name : String, salary : double)
-setPassword() : void
-isValidPassword(password : String) : boolean
-readPassword() : String
+logIn() : void
+getAdminInfo() : void
+equals(o : Object) : boolean
+hashCode() : int
Customer
-idCountCustomer : int
-customerId : String
-order : Order
-orderHistory : Order = new ArrayList<>()
~Customer(name : String)
+makeOrder(items : ArrayList<Item>) : void
+getOrderHistory() : void
+getCustomerInfo() : void
+equals(o : Object) : boolean
+hashCode() : int
Financials
-profit : double
-revenue : double
-expenses : double
-date : LocalDate
~Financials(date : LocalDate)
-accumulateRevenue(date : LocalDate) : void
-accumulateExpenses(date : LocalDate) : void
+toString() : String
Expenses
-date : LocalDate
-operatingExpenses : double
-wages : double
-total : double
-expensesHistory : Expenses = new ArrayList<>()
+Expenses(date : LocalDate, operatingExpenses : double, wages : double)
+getExpensesHistory() : ArrayList<Expenses>
+toString() : String
+main(args : String[]) : void
Sales
-date : LocalDate
-medicineStocksSold : HashMap<Medicine, Integer> = new HashMap<>()
-supplementsStocksSold : HashMap<Supplements, Integer> = new HashMap<>()
-babyCareStocksSold : HashMap<BabyCare, Integer> = new HashMap<>()
-personalCareStocksSold : HashMap<PersonalCare, Integer> = new HashMap<>()
-devicesStocksSold : HashMap<Devices, Integer> = new HashMap<>()
-totalSalesToday : double
-salesHistory : Sales = new ArrayList<>()
~Sales()
~Sales(date : LocalDate)
+getSalesHistory() : ArrayList<Sales>
+addToTotalSalesToday(total : double) : void
+recordSale(item : Item, soldAmount : int) : void
+main(args : String[]) : void
+getDate() : LocalDate
+getMedicineStocksSold() : HashMap<Medicine, Integer>
+getSupplementsStocksSold()
+getBabyCareStocksSold()
+getPersonalCareStocksSold()
+getTotalSalesToday()
Order
-orderId : String
-items : Item = new ArrayList<>()
-inventory : Inventory
-orderNumber : int = 0
+Order(inventory : Inventory)
+Order(items : ArrayList<Item>)
+getItems() : ArrayList<Item>
+addItem(item : Item, quantity : int) : void
+removeItem(item : Item, quantity : int) : void
+calculateOrderTotal() : double
+finalizeOrder(payment : Payments, amount : double) : void
-generateReceipt() : Receipt
Inventory
-medicineStock : Map<Medicine, Integer>
-supplementsStock : Map<Supplements, Integer>
-babyCareStock : Map<BabyCare, Integer>
-personalCareStock : Map<PersonalCare, Integer>
-devicesStock : Map<Devices, Integer>
+Inventory()
+addStock(item : Item, quantity : int) : void
+removeStock(item : Item) : void
+isAvailable(item : Item) : boolean
+deductFromInventory(item : Item, quantity : int) : boolean
+showItemStock(item : Item) : void
+showSupStock() : void
+showMedStock() : void
+showBabyStock() : void
+showPersonalStock() : void
+addDevice(dev : Devices, quantity : int) : void
+decreaseDevices(dev : Devices, quantity : int) : void
+removeDevice(dev : Devices) : void
+showDevicesStock() : void
Receipt
-total : double
-receiptDetails : String = ""
-order : Order
~Receipt(order : Order)
-makeReceipt() : void
+toString() : String
+main(args : String[]) : void
Item
-stockId : String
-name : String
-manufacturer : String
-expiryDate : LocalDate
-price : double
-allergyInducing : boolean
+Item(stockId : String, name : String, manufacturer : String, price : double, shelfLife : int, allergyInducing : boolean)
+getAllergyInducting() : boolean
+getDescription() : void
+compareTo(item : Item) : int
+compare(item : Item) : void
+toString() : String
<<Interface>>
Payment
+pay(amount : double) : boolean
Cash
-requiredAmount : double
+Cash(requiredAmount : double)
+pay(amount : double) : boolean
Card
~requiredAmount : double
~cardBalance : double
+Card(requiredAmount : double, cardBalance : double)
+pay(amount : double) : boolean
Medicine
-form : String
-dose : int
+Medicine(stockId : String, name : String, manufacturer : Strin...
+getDescription() : void
Supplement
-expiryDate : LocalDate
-Form : String
-Dose : int
+getDescription() : void
BabyCare
-expiryDate : LocalDate
-category : String
+getDescription() : void
PersonalCare
-expiryDate : LocalDate
-category : String
+getDescription() : void
Device
-isDigital : boolean
-usage : String
+getDescription() : void

Our Architecture

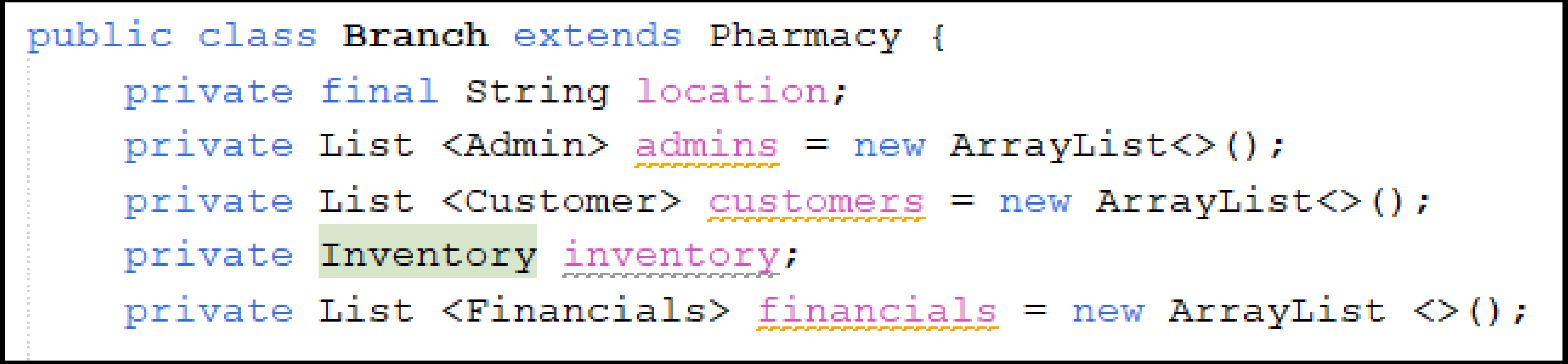

Java Pharmacy System Code Execution Flow
Pharmacy and Branch Initialization
Admin and Customer Setup
Inventory Setup
Customer 1 Makes an Order
Customer 2 Makes Another Order
Financial Tracking
Branch Financial Overview
Features Demonstrated
User registration
Inventory setup
Order creation and validation
Payment system
Cash payment
Card payment
Stock tracking
Financial reporting

Encapsulation

- Encapsulation is one of the core principles of object-oriented programming. It means keeping data (fields) private inside a class and providing controlled access through public methods (getters/setters), while also hiding internal implementation details via private helper methods.

Encapsulation-based Design of Our Pharmacy System:

- All class data fields are declared private.

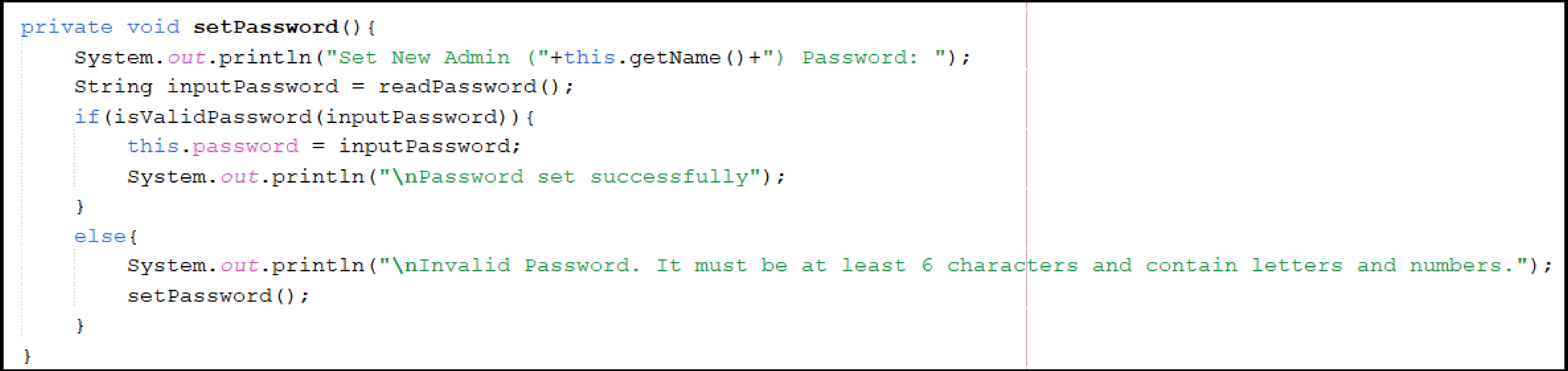


```java
public class Branch extends Pharmacy {
    private final String location;
    private List <Admin> admins = new ArrayList<>();
    private List <Customer> customers = new ArrayList<>();
    private Inventory inventory;
    private List <Financials> financials = new ArrayList <>();
```

Encapsulation

Encapsulation-based Design of Our Pharmacy System:

- Sensitive logic (e.g. password setup) is handled through private methods.

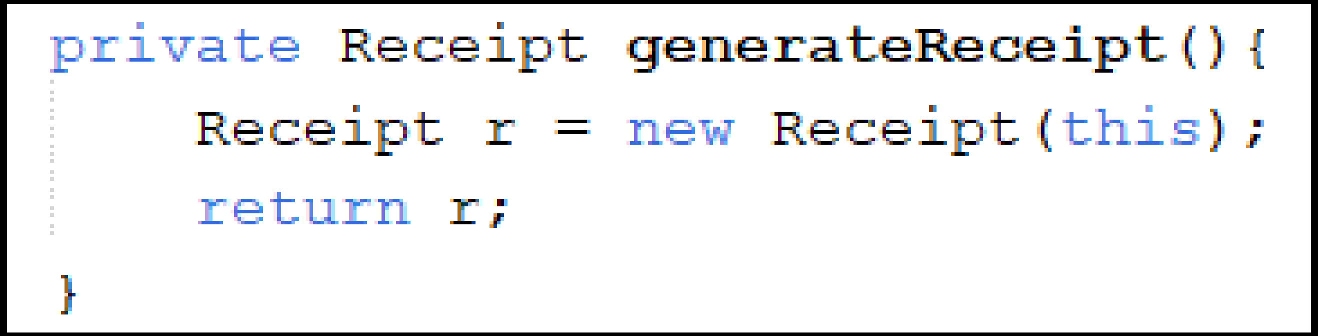


```java
private void setPassword(){
    System.out.println("Set New Admin ("+this.getName()+") Password: ");
    String inputPassword = readPassword();
    if(isValidPassword(inputPassword)){
        this.password = inputPassword;
        System.out.println("\nPassword set successfully");
    }
    else{
        System.out.println("\nInvalid Password. It must be at least 6 characters and contain letters and numbers.");
        setPassword();
    }
}
```

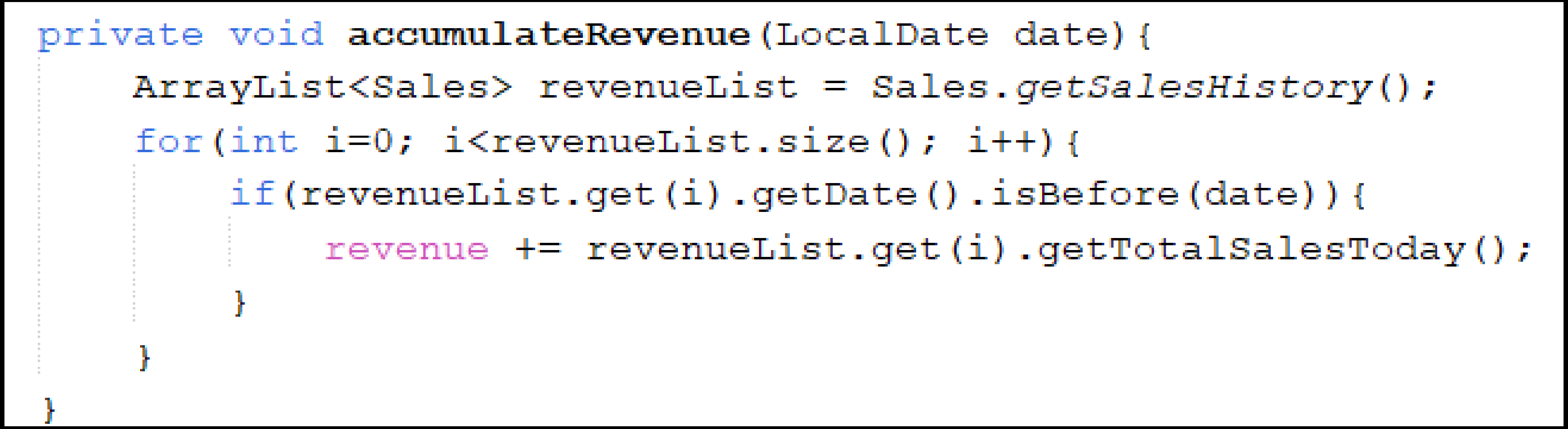
Encapsulation

Encapsulation-based Design of Our Pharmacy System:

- Methods that are only used within their own class.
- setPassword(), isValidPassword(), readPassword() in Admin
- generateReceipt() in Order
- accumulateRevenue(), accumulateExpenses() in Financials



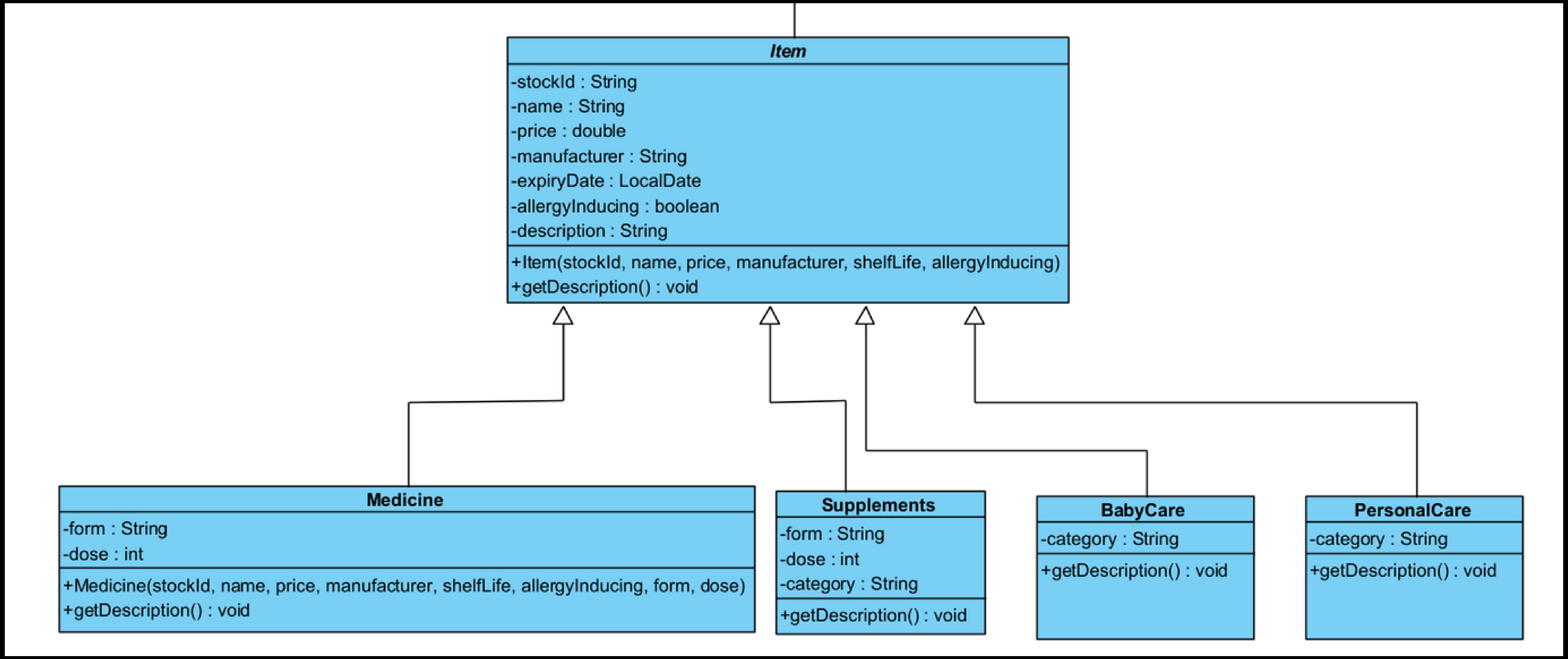
```
private Receipt generateReceipt(){
    Receipt r = new Receipt(this);
    return r;
}
```



```
private void accumulateRevenue(LocalDate date){
    ArrayList<Sales> revenueList = Sales.getSalesHistory();
    for(int i=0; i<revenueList.size(); i++){
        if(revenueList.get(i).getDate().isBefore(date)){
            revenue += revenueList.get(i).getTotalSalesToday();
        }
    }
}
```

Inheritance and Polymorphism

Abstract Class: Item



Concrete classes extending abstract class Item

Medicine

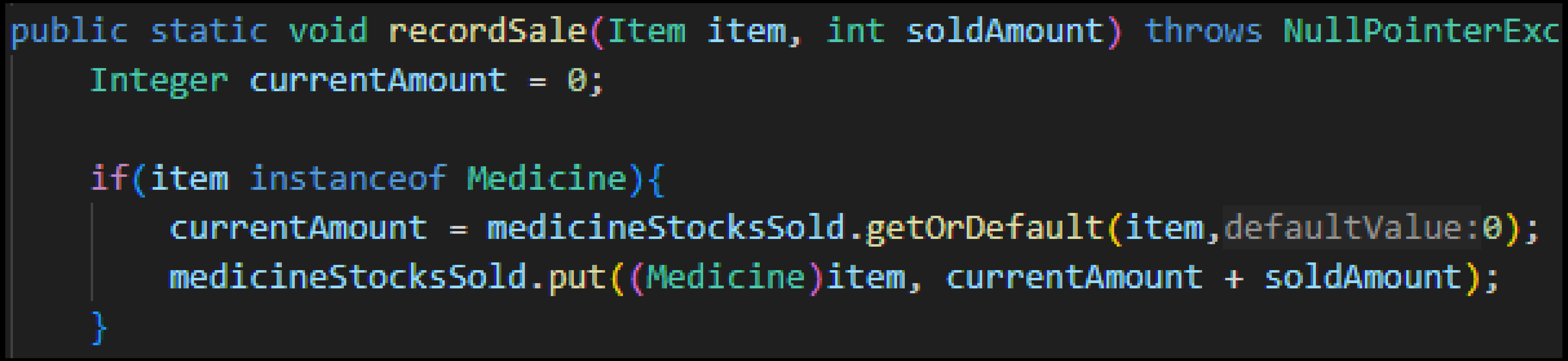
Supplement

BabyCare

PersonalCare

Inheritance and Polymorphism

Polymorphism Code Example:



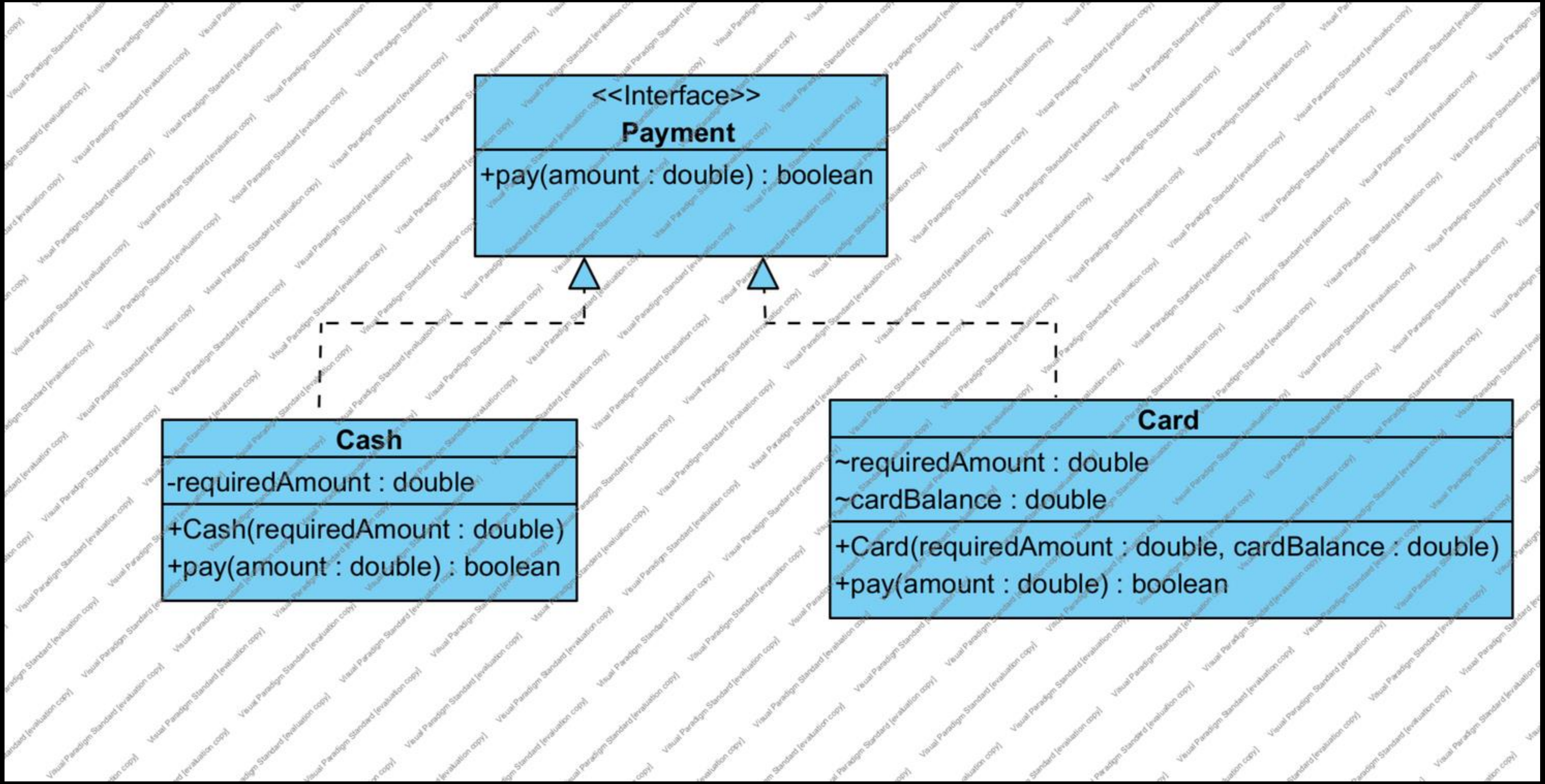
```
public static void recordSale(Item item, int soldAmount) throws NullPointerExc
    Integer currentAmount = 0;

    if(item instanceof Medicine){
        currentAmount = medicineStocksSold.getOrDefault(item,defaultValue:0);
        medicineStocksSold.put((Medicine)item, currentAmount + soldAmount);
    }
```

Explanation:

- Argument passed to function has to be of Item type
- MedicineStocksSold is a HashMap<Medicine, Integer>
- Thus, we have to *downcast* Item type to its *subclass* Medicine.

Inheritance and Polymorphism

Interface: Payments



<<Interface>>
Payment
+pay(amount : double) : boolean
Cash
-requiredAmount : double
+Cash(requiredAmount : double)
+pay(amount : double) : boolean
Card
~requiredAmount : double
~cardBalance : double
+Card(requiredAmount : double, cardBalance : double)
+pay(amount : double) : boolean

Concrete Classes Implementing Payment:

Cash
Card

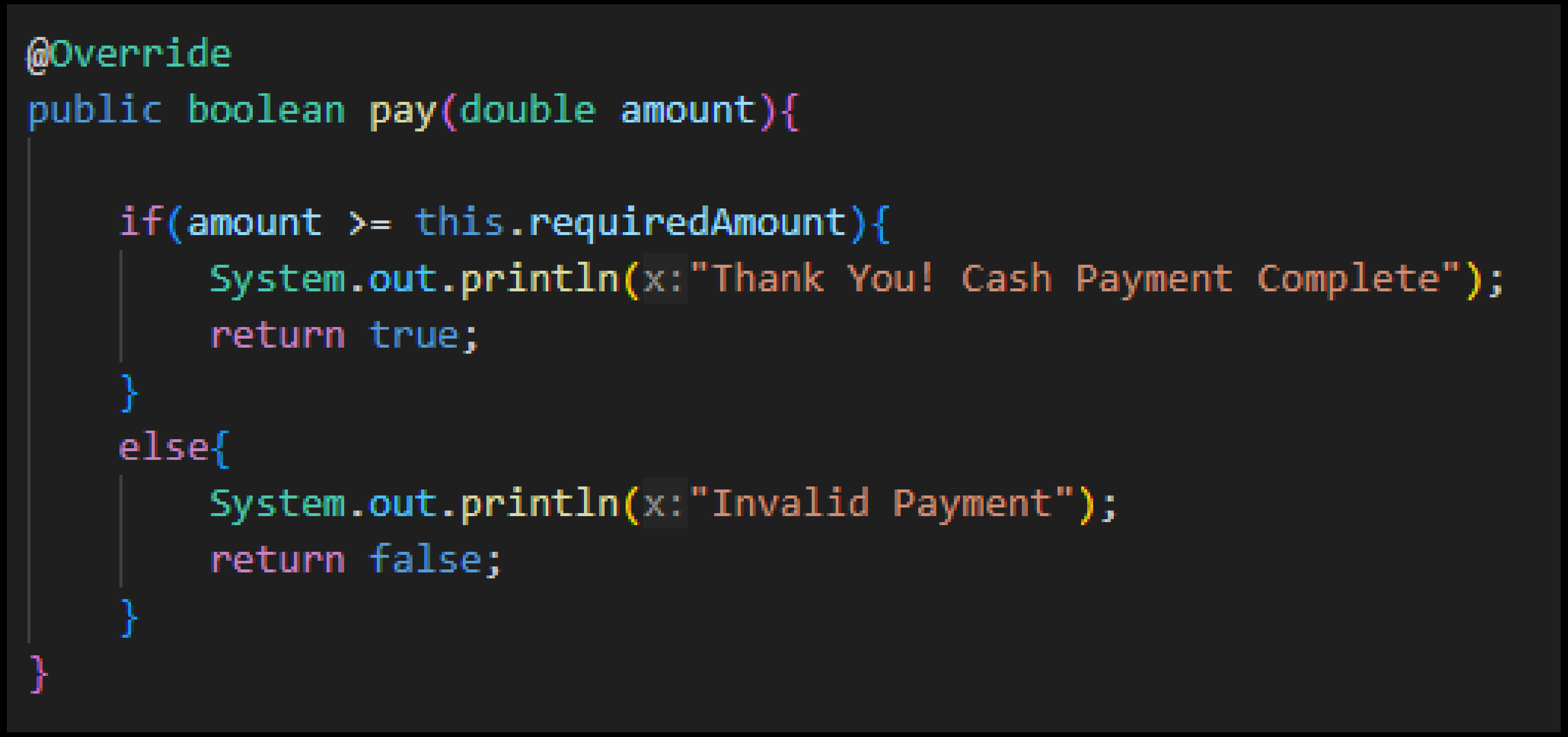
Description:

Payment interface describes the common behavior of a “payment transaction”.

Inheritance and Polymorphism

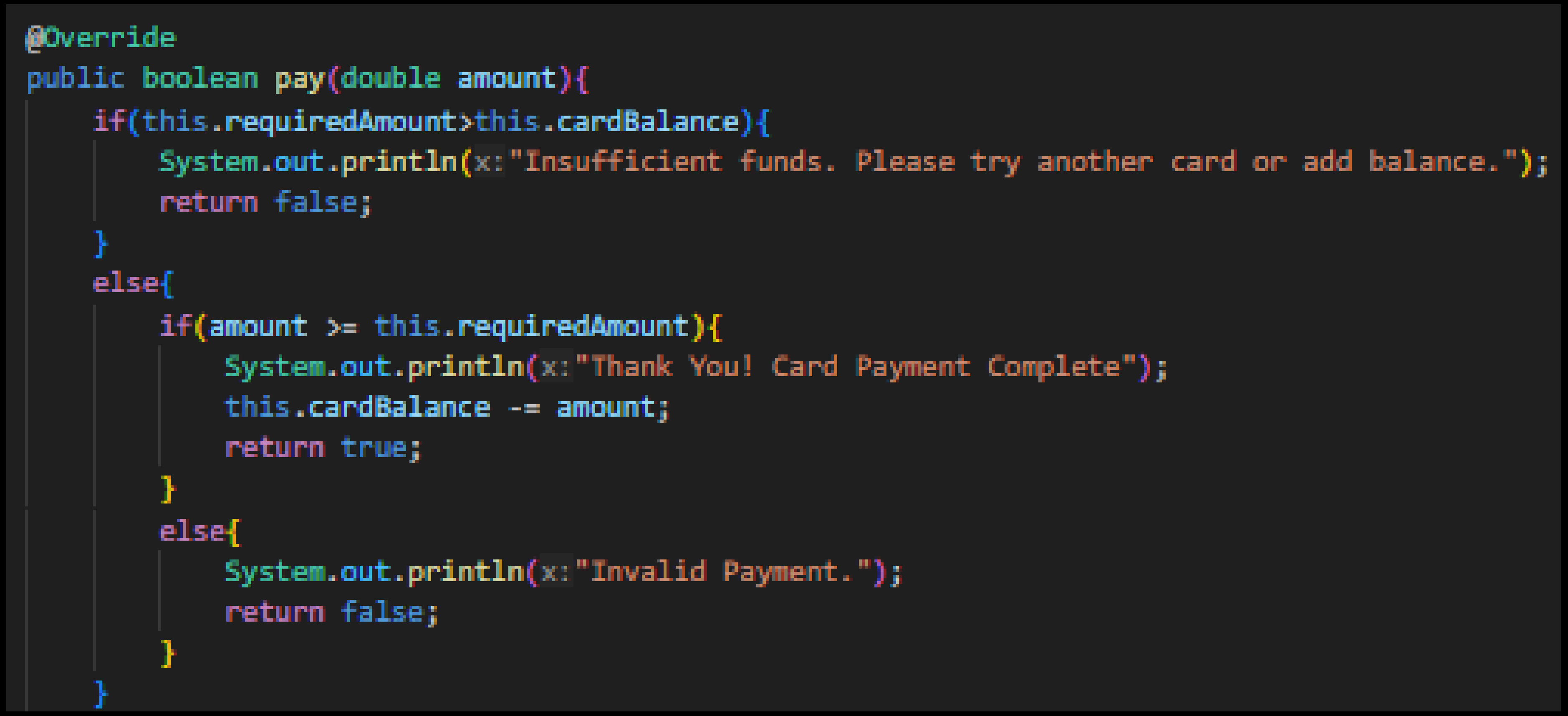
Interface: Payments

Cash Implementation



```java
@Override
public boolean pay(double amount){

    if(amount >= this.requiredAmount){
        System.out.println(x:"Thank You! Cash Payment Complete");
        return true;
    }
    else{
        System.out.println(x:"Invalid Payment");
        return false;
    }
}
```

Card Implementation

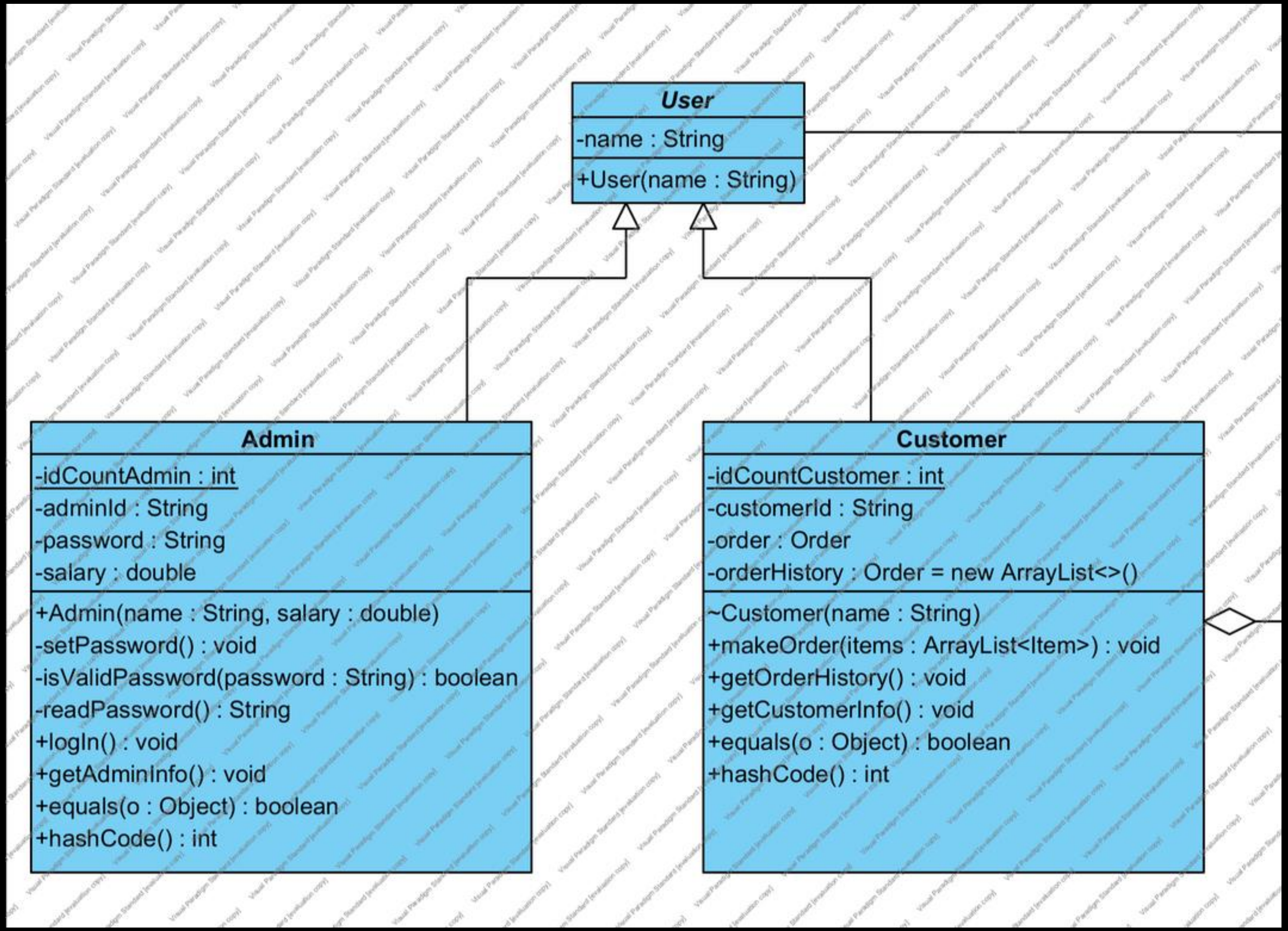


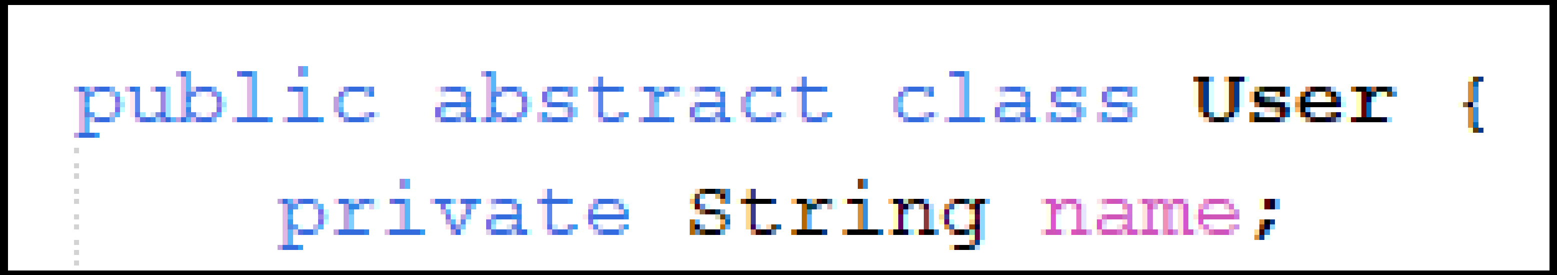
```java
@Override
public boolean pay(double amount){
    if(this.requiredAmount>this.cardBalance){
        System.out.println(x:"Insufficient funds. Please try another card or add balance.");
        return false;
    }
    else{
        if(amount >= this.requiredAmount){
            System.out.println(x:"Thank You! Card Payment Complete");
            this.cardBalance -= amount;
            return true;
        }
        else{
            System.out.println(x:"Invalid Payment.");
            return false;
        }
    }
```

Inheritance and Polymorphism

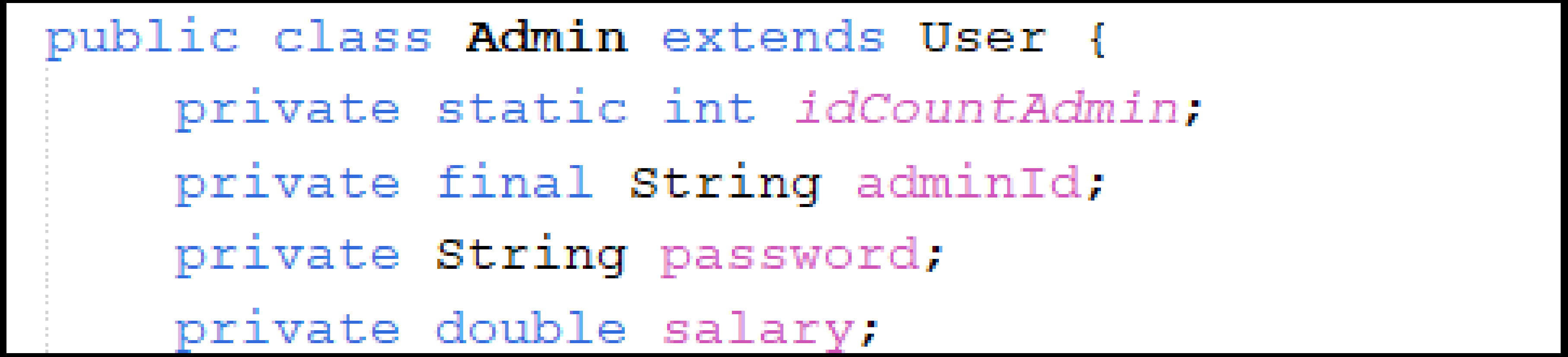
Abstract Class: User

User is an abstract entity, part of the Branch class. Concretely extended by two classes: Admin and Customer

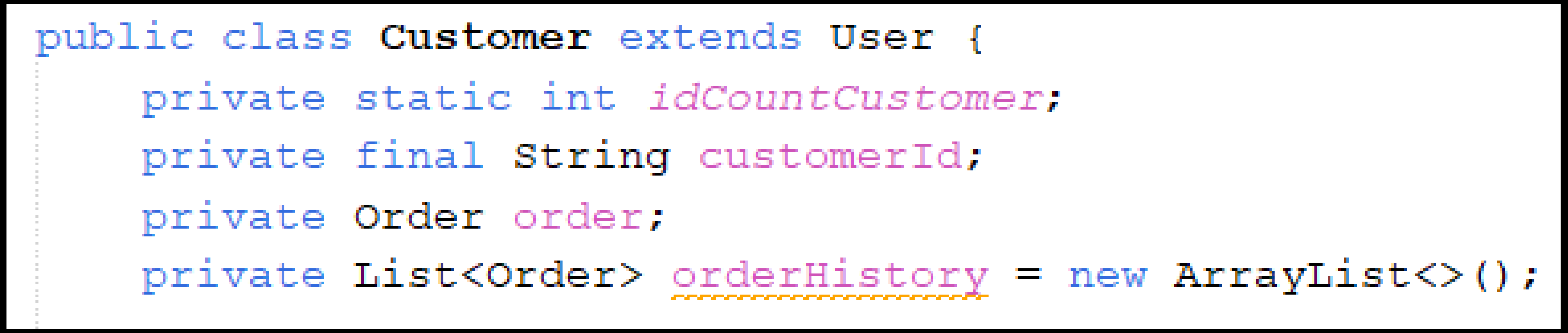




```
public abstract class User {
    private String name;
```



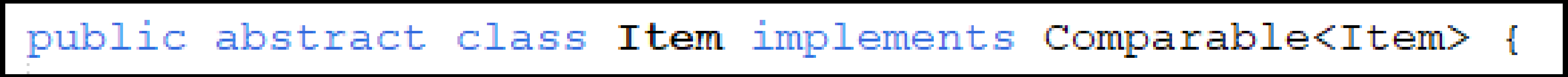
```
public class Admin extends User {
    private static int idCountAdmin;
    private final String adminId;
    private String password;
    private double salary;
```



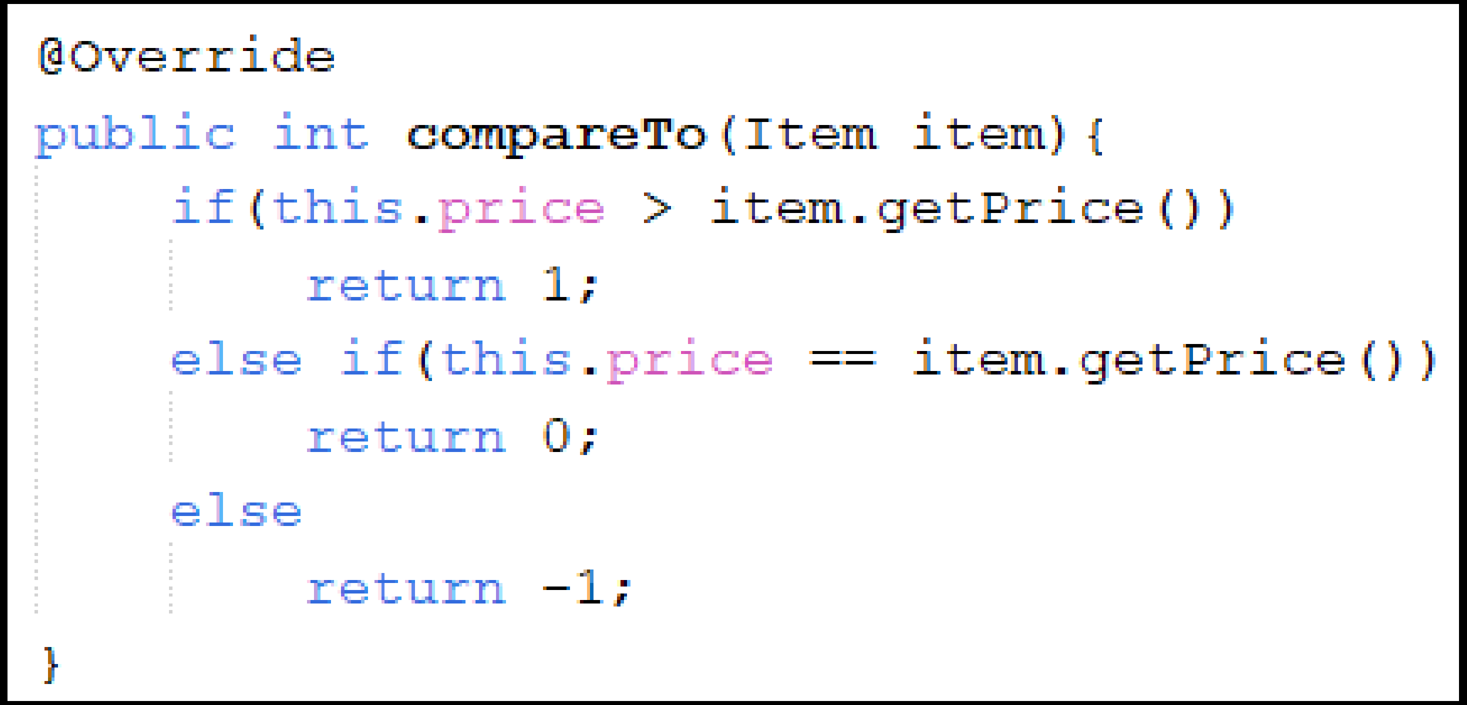
```
public class Customer extends User {
    private static int idCountCustomer;
    private final String customerId;
    private Order order;
    private List<Order> orderHistory = new ArrayList<>();
```

Sorting

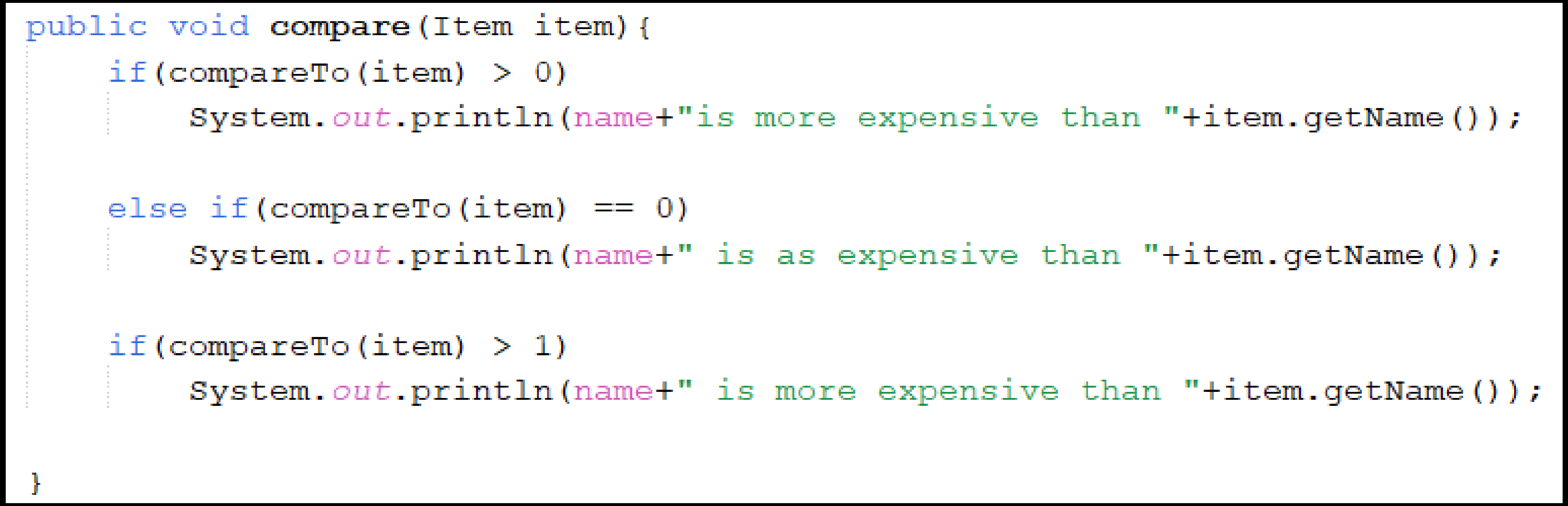
- To improve user experience we sorted items by price, by implementing the Comparable interface which allowed easy and efficient sorting of products.



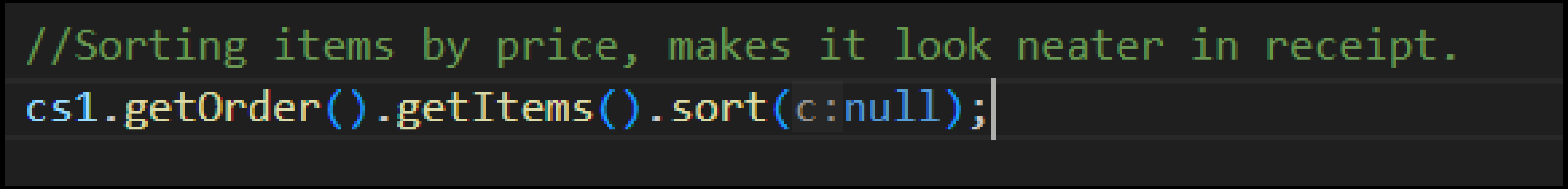
```java
public abstract class Item implements Comparable<Item> {
```



```java
@Override
public int compareTo(Item item){
    if(this.price > item.getPrice())
        return 1;
    else if(this.price == item.getPrice())
        return 0;
    else
        return -1;
}
```

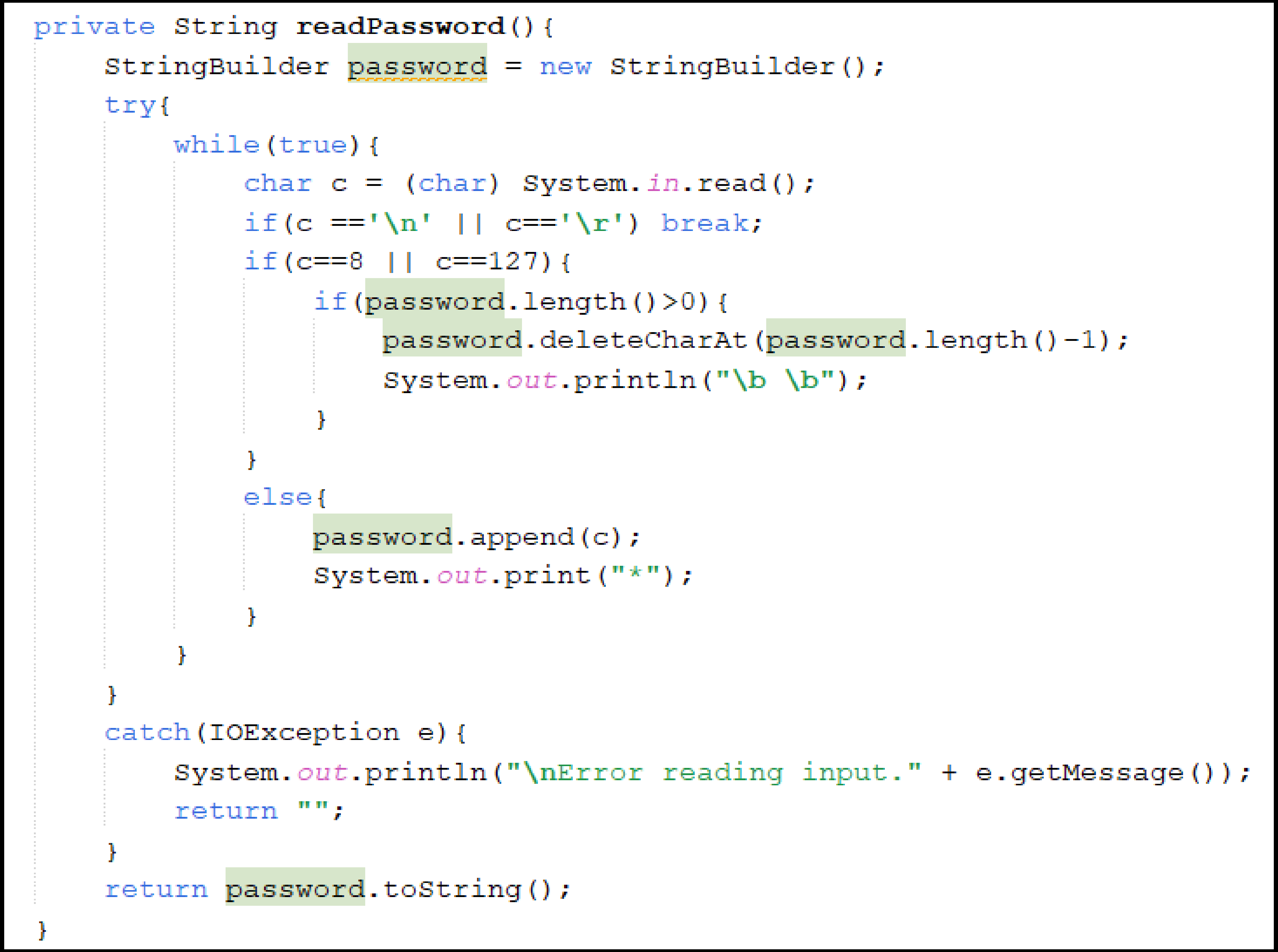


```java
public void compare(Item item){
    if(compareTo(item) > 0)
        System.out.println(name+"is more expensive than "+item.getName());

    else if(compareTo(item) == 0)
        System.out.println(name+" is as expensive than "+item.getName());

    if(compareTo(item) > 1)
        System.out.println(name+" is more expensive than "+item.getName());

}
```



```java
//Sorting items by price, makes it look neater in receipt.
cs1.getOrder().getItems().sort(c:null);
```

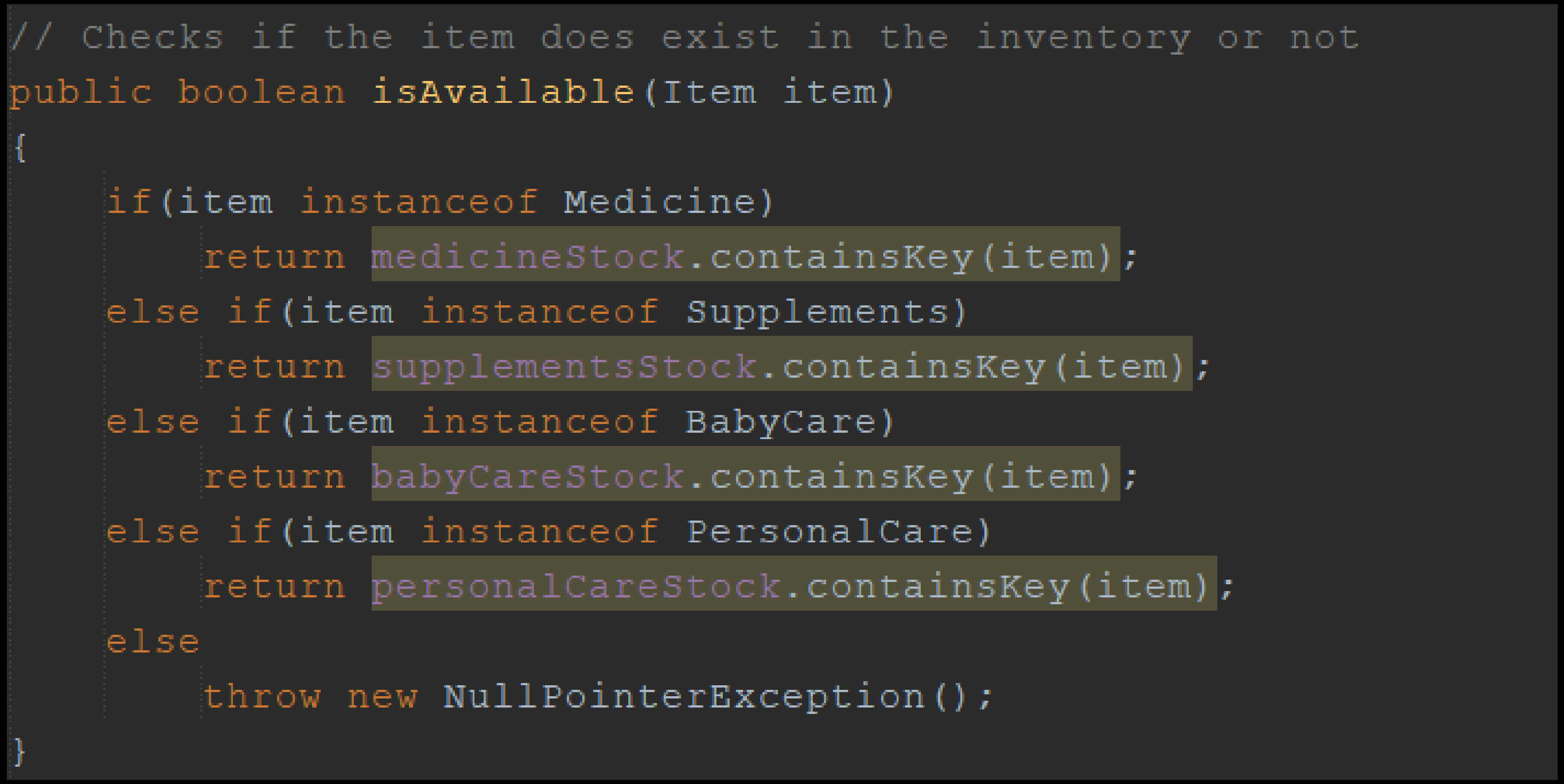
Exception Handling



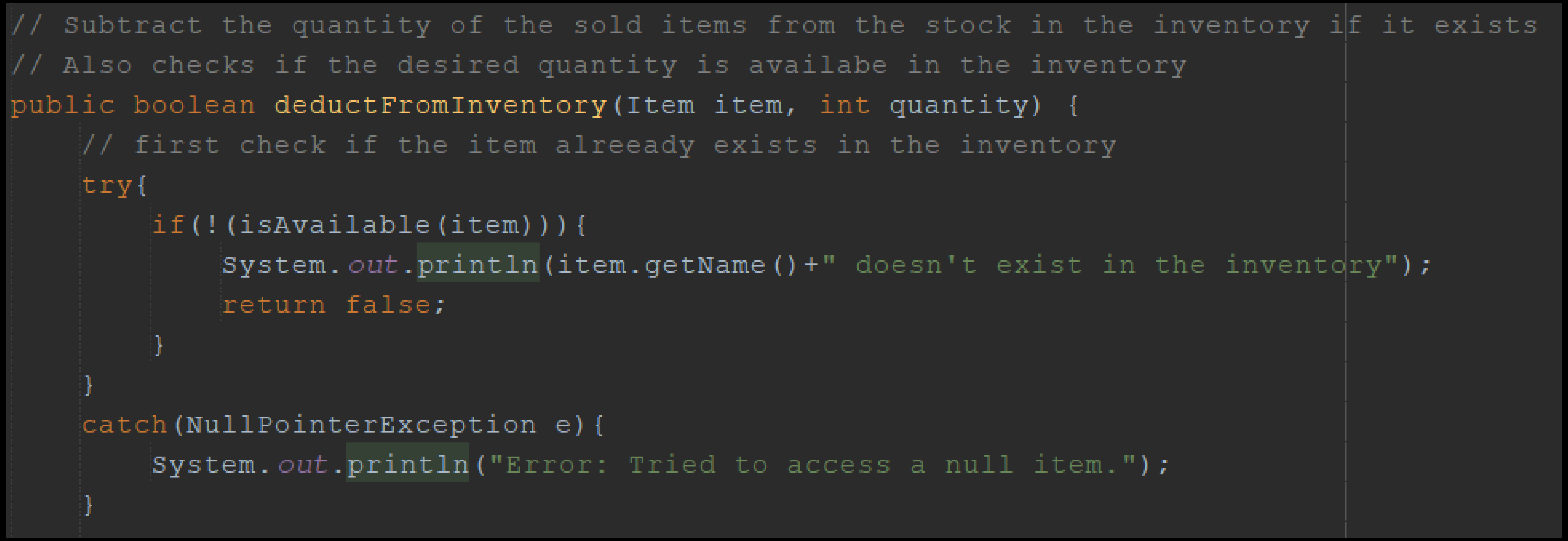
```java
private String readPassword(){
    StringBuilder password = new StringBuilder();
    try{
        while(true){
            char c = (char) System.in.read();
            if(c =='\n' || c=='\r') break;
            if(c==8 || c==127){
                if(password.length()>0){
                    password.deleteCharAt(password.length()-1);
                    System.out.println("\b \b");
                }
            }
            else{
                password.append(c);
                System.out.print("*");
            }
        }
    }
    catch(IOException e){
        System.out.println("\nError reading input." + e.getMessage());
        return "";
    }
    return password.toString();
}
```

- IO Exception Handling is a checked exception which needs to be implemented in readPassword().
- It catches input reading errors in to notify the user and allows retrying.

Exception Handling



```java
// Checks if the item does exist in the inventory or not
public boolean isAvailable(Item item)
{
    if(item instanceof Medicine)
        return medicineStock.containsKey(item);
    else if(item instanceof Supplements)
        return supplementsStock.containsKey(item);
    else if(item instanceof BabyCare)
        return babyCareStock.containsKey(item);
    else if(item instanceof PersonalCare)
        return personalCareStock.containsKey(item);
    else
        throw new NullPointerException();
}
```



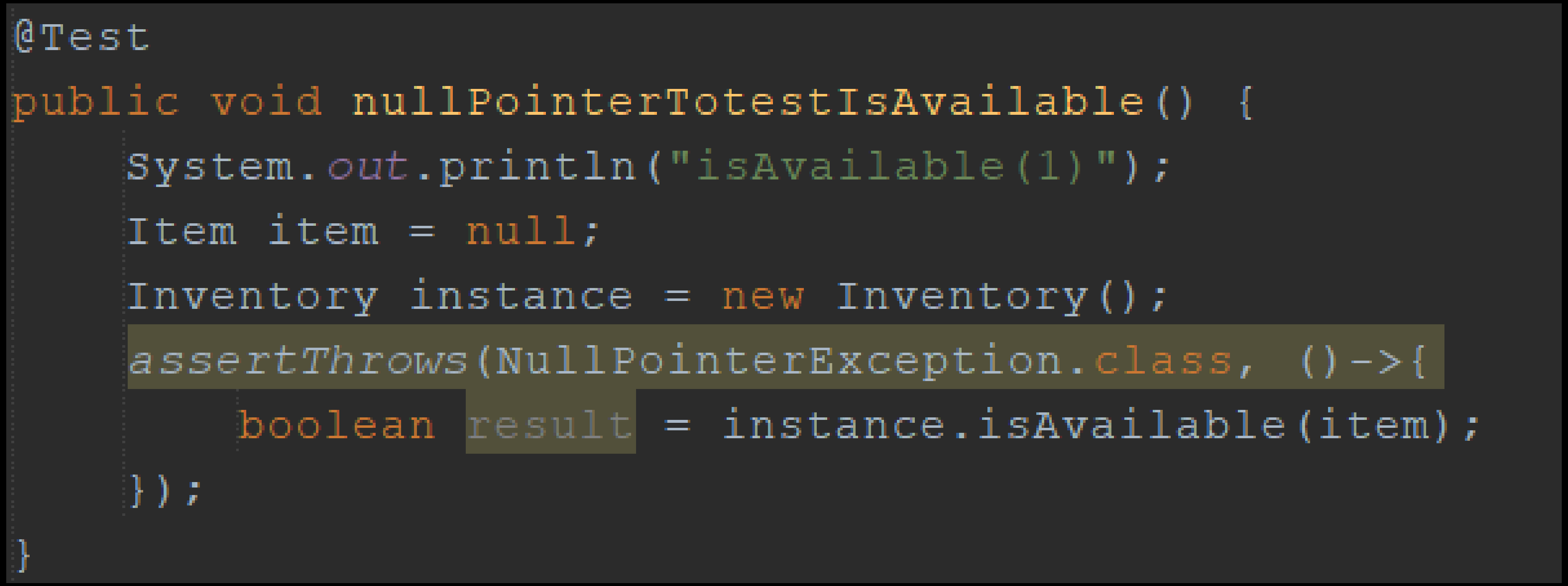
```java
// Subtract the quantity of the sold items from the stock in the inventory if it exists
// Also checks if the desired quantity is availabe in the inventory
public boolean deductFromInventory(Item item, int quantity) {
    // first check if the item alreeady exists in the inventory
    try{
        if(!(isAvailable(item))){
            System.out.println(item.getName()+" doesn't exist in the inventory");
            return false;
        }
    }
    catch(NullPointerException e){
        System.out.println("Error: Tried to access a null item.");
    }
```

- The method isAvailable(item) may throw a NullPointerException if the item is not initialized (i.e., it is null).

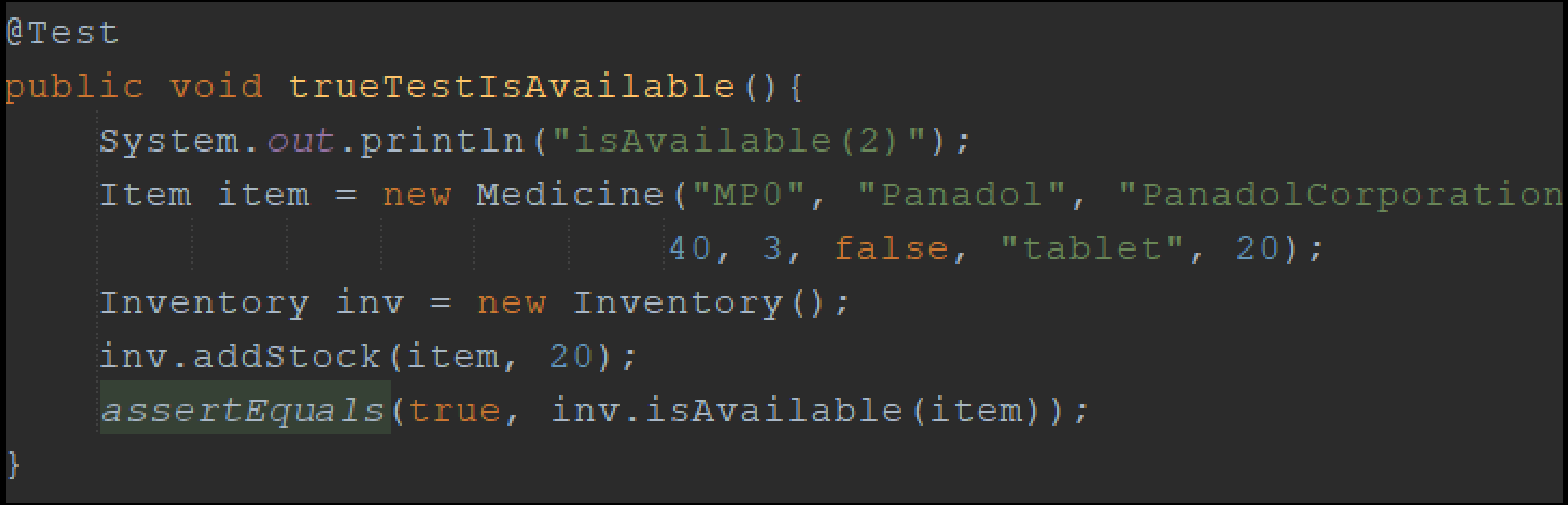
- Any method that calls isAvailable() ensures proper handling of this exception to maintain program stability and avoid unexpected termination such as the method deductFromInventory(item, quantity).

JUnit Testing

Here we tested the function isAvailable by forcing three possible scenarios to insure the functionality of this function

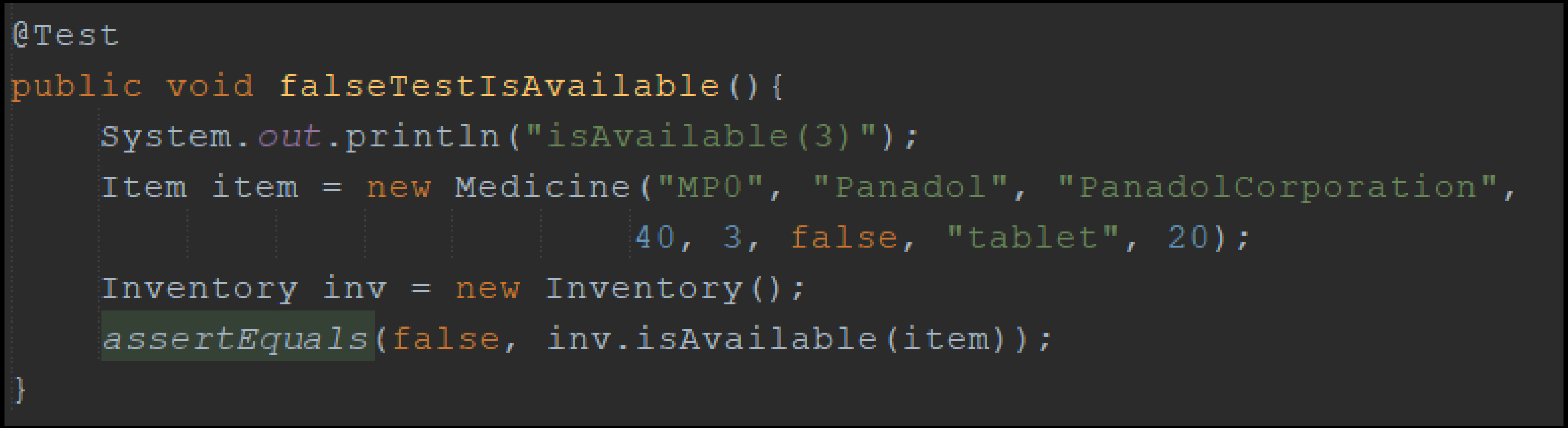


```java
@Test
public void nullPointerTotestIsAvailable() {
    System.out.println("isAvailable(1)");
    Item item = null;
    Inventory instance = new Inventory();
    assertThrows(NullPointerException.class, ()->{
        boolean result = instance.isAvailable(item);
    });
}
```

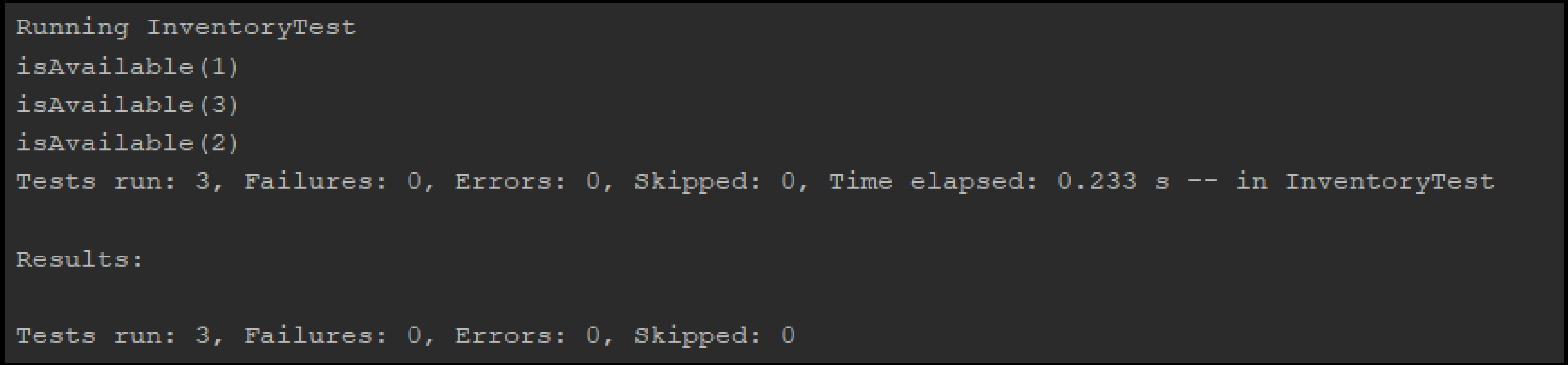


```java
@Test
public void trueTestIsAvailable(){
    System.out.println("isAvailable(2)");
    Item item = new Medicine("MP0", "Panadol", "PanadolCorporation
                            40, 3, false, "tablet", 20);
    Inventory inv = new Inventory();
    inv.addStock(item, 20);
    assertEquals(true, inv.isAvailable(item));
}
```

JUnit Testing



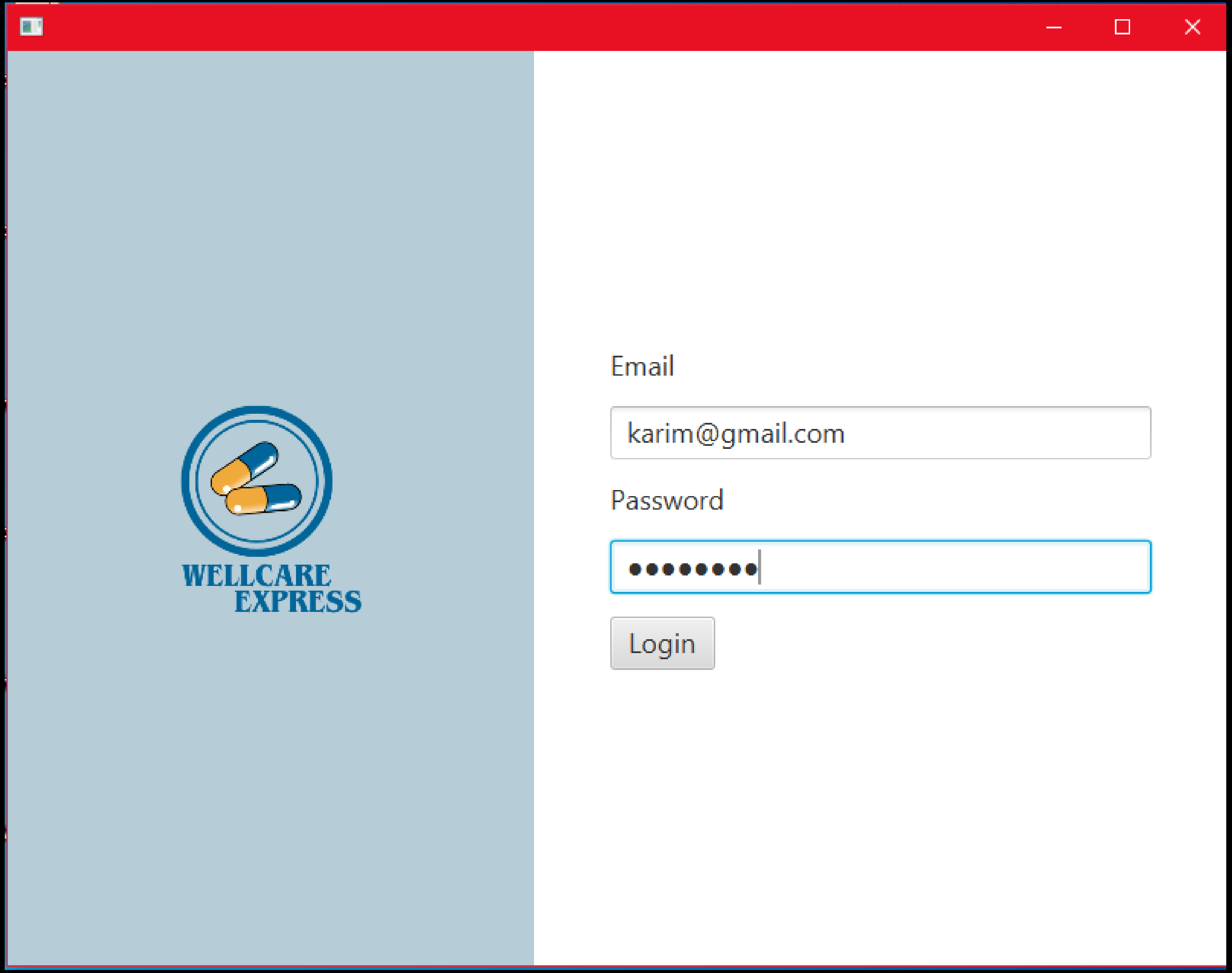
```java
@Test
public void falseTestIsAvailable(){
    System.out.println("isAvailable(3)");
    Item item = new Medicine("MP0", "Panadol", "PanadolCorporation",
                            40, 3, false, "tablet", 20);
    Inventory inv = new Inventory();
    assertEquals(false, inv.isAvailable(item));
}
```



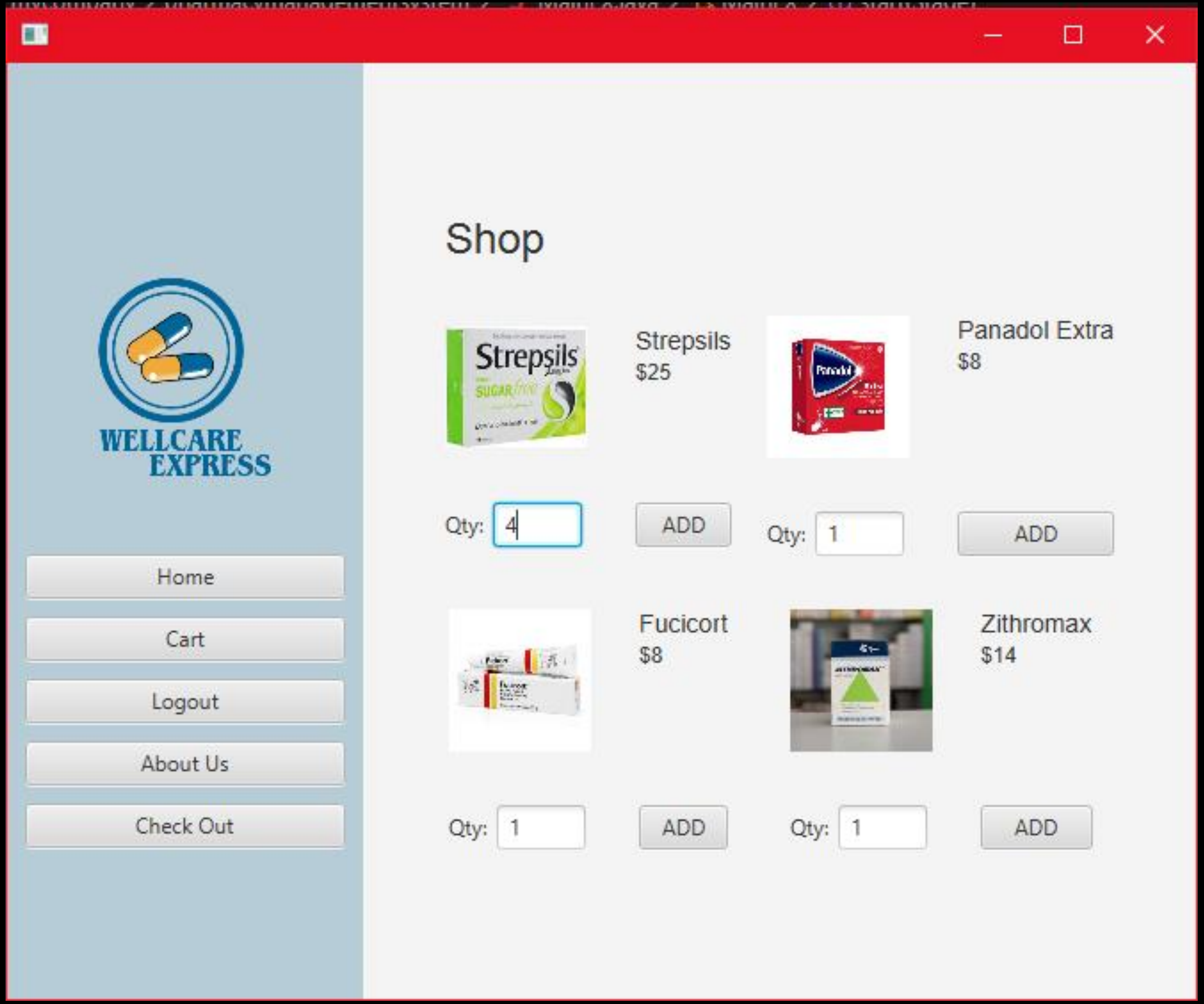
```
Running InventoryTest
isAvailable(1)
isAvailable(3)
isAvailable(2)
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.233 s -- in InventoryTest

Results:

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
```

GUI - Main Flow

Scene 1: On Launch

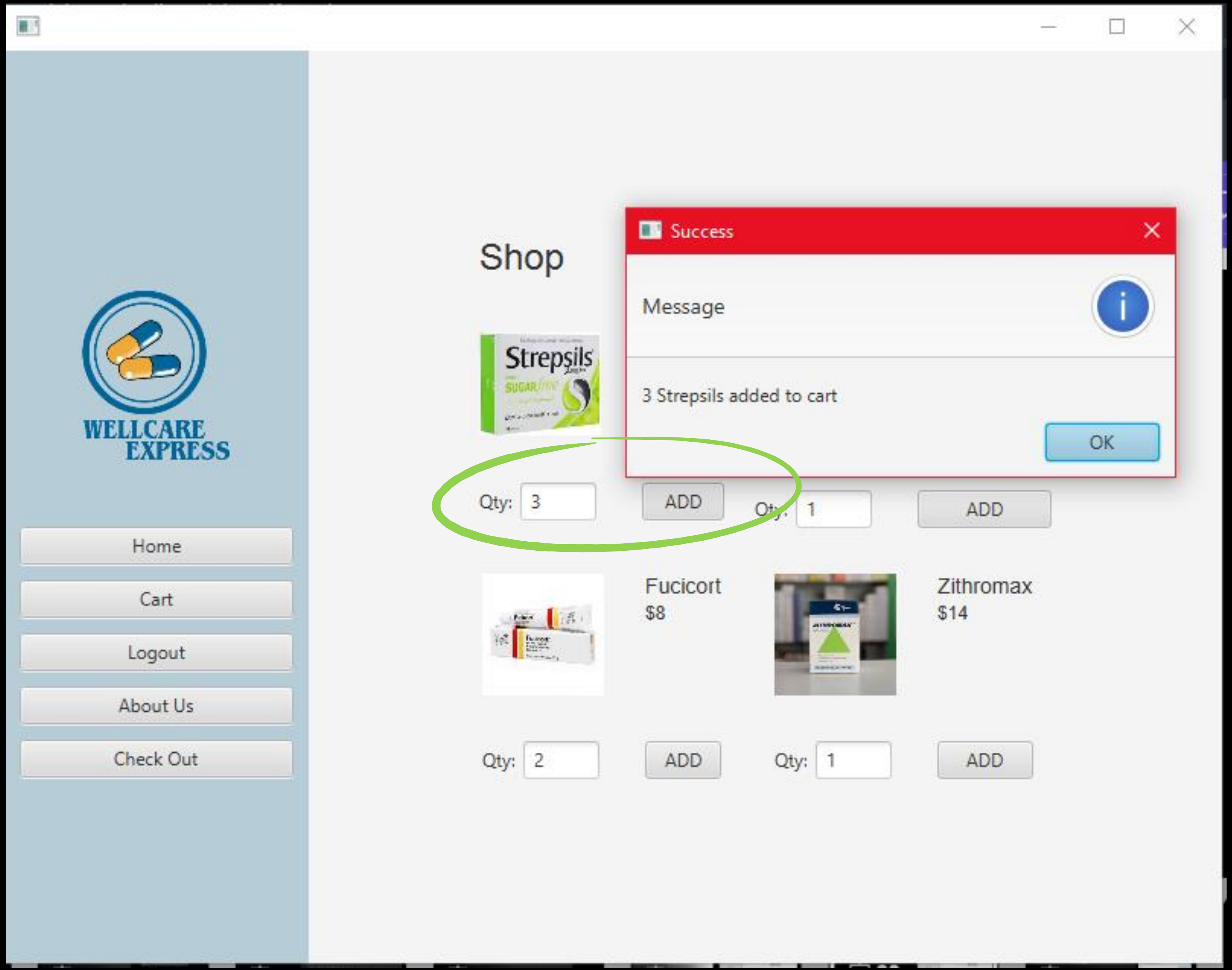


Scene 2: On Clicking Login



GUI - Main Flow

Scene 3: On Clicking Add



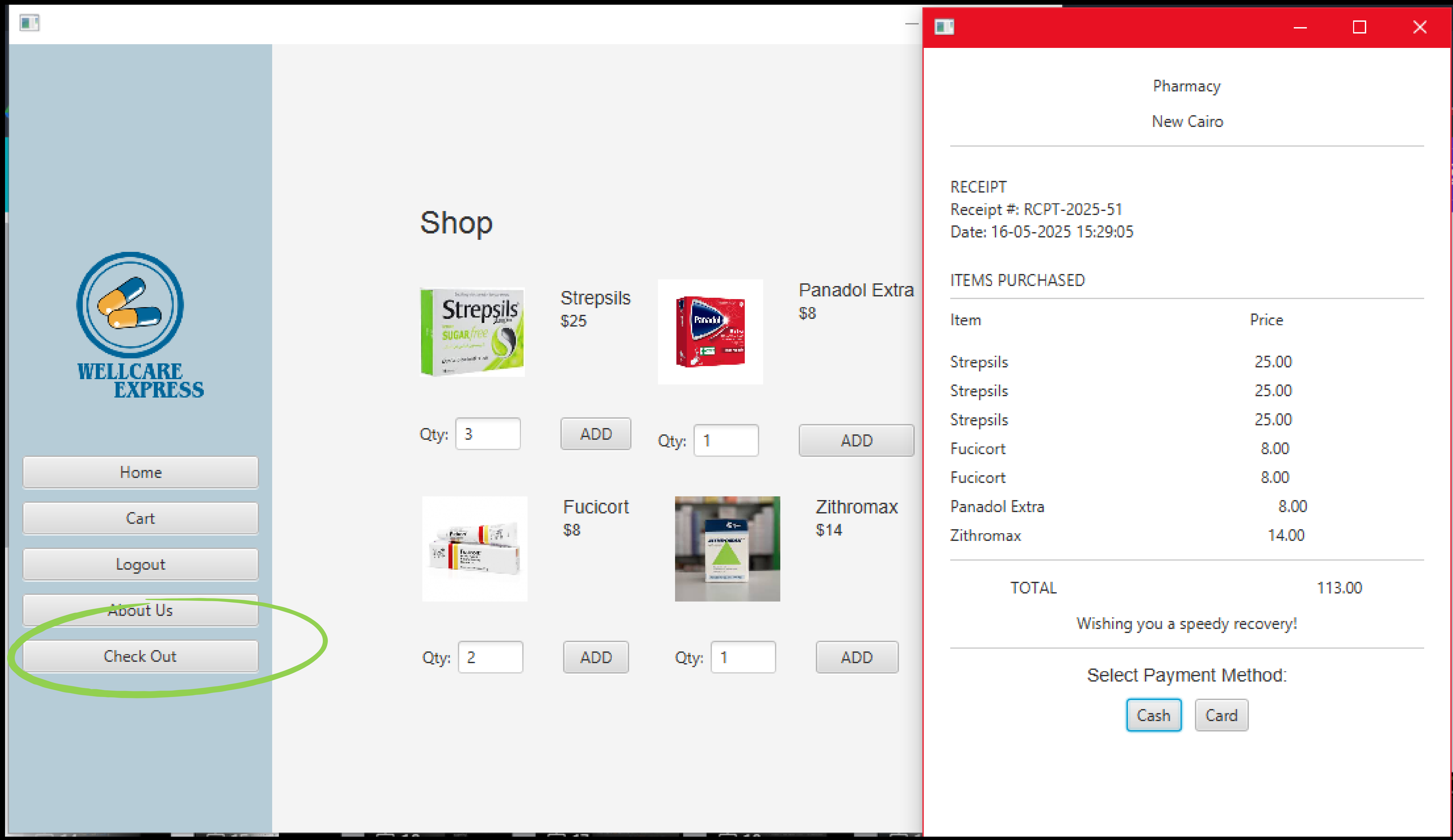
Success

Message

3 Strepsils added to cart

OK

Scene 4: On Clicking Checkout



Pharmacy

New Cairo

RECEIPT
Receipt #: RCPT-2025-51
Date: 16-05-2025 15:29:05

ITEMS PURCHASED

Item	Price
Strepsils	25.00
Strepsils	25.00
Strepsils	25.00
Fucicort	8.00
Fucicort	8.00
Panadol Extra	8.00
Zithromax	14.00

TOTAL 113.00

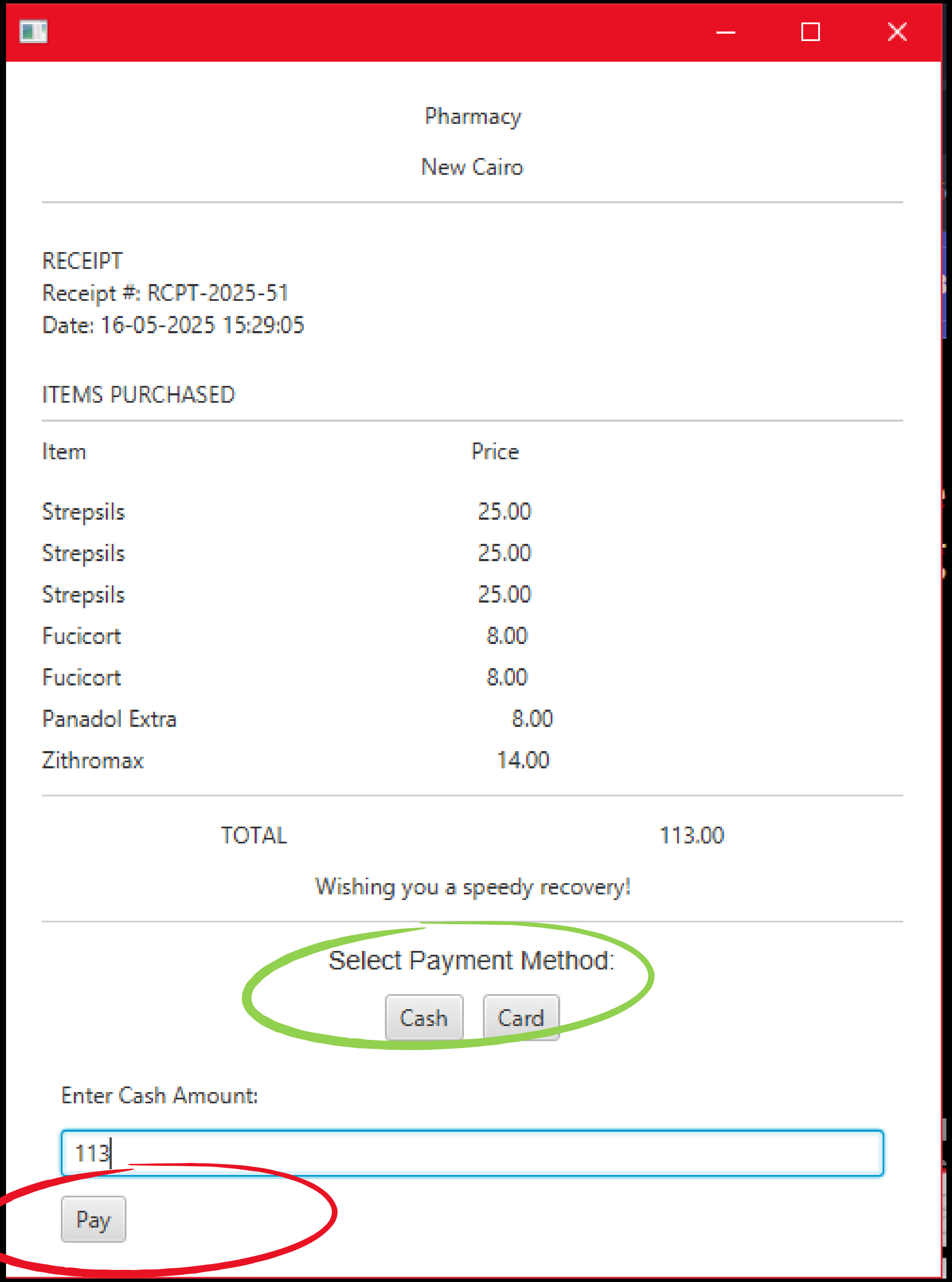
Wishing you a speedy recovery!

Select Payment Method:

Cash Card

GUI - Main Flow

Scene 5: On Clicking Cash



Pharmacy

New Cairo

RECEIPT
Receipt #: RCPT-2025-51
Date: 16-05-2025 15:29:05

ITEMS PURCHASED

Item	Price
Strepsils	25.00
Strepsils	25.00
Strepsils	25.00
Fucicort	8.00
Fucicort	8.00
Panadol Extra	8.00
Zithromax	14.00

TOTAL 113.00

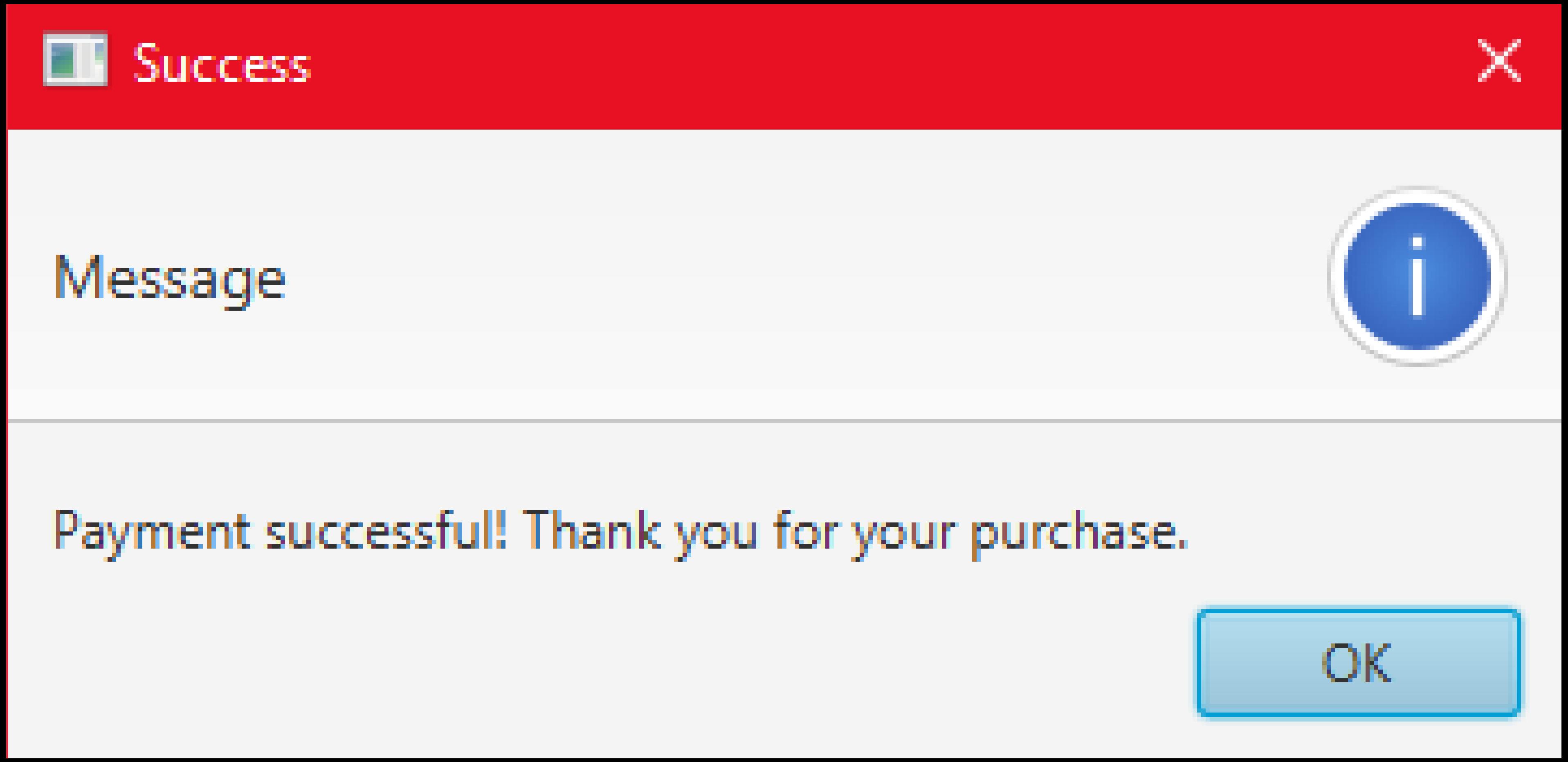
Wishing you a speedy recovery!

Select Payment Method:

Enter Cash Amount:

113

Scene 6: On Clicking Pay



Success

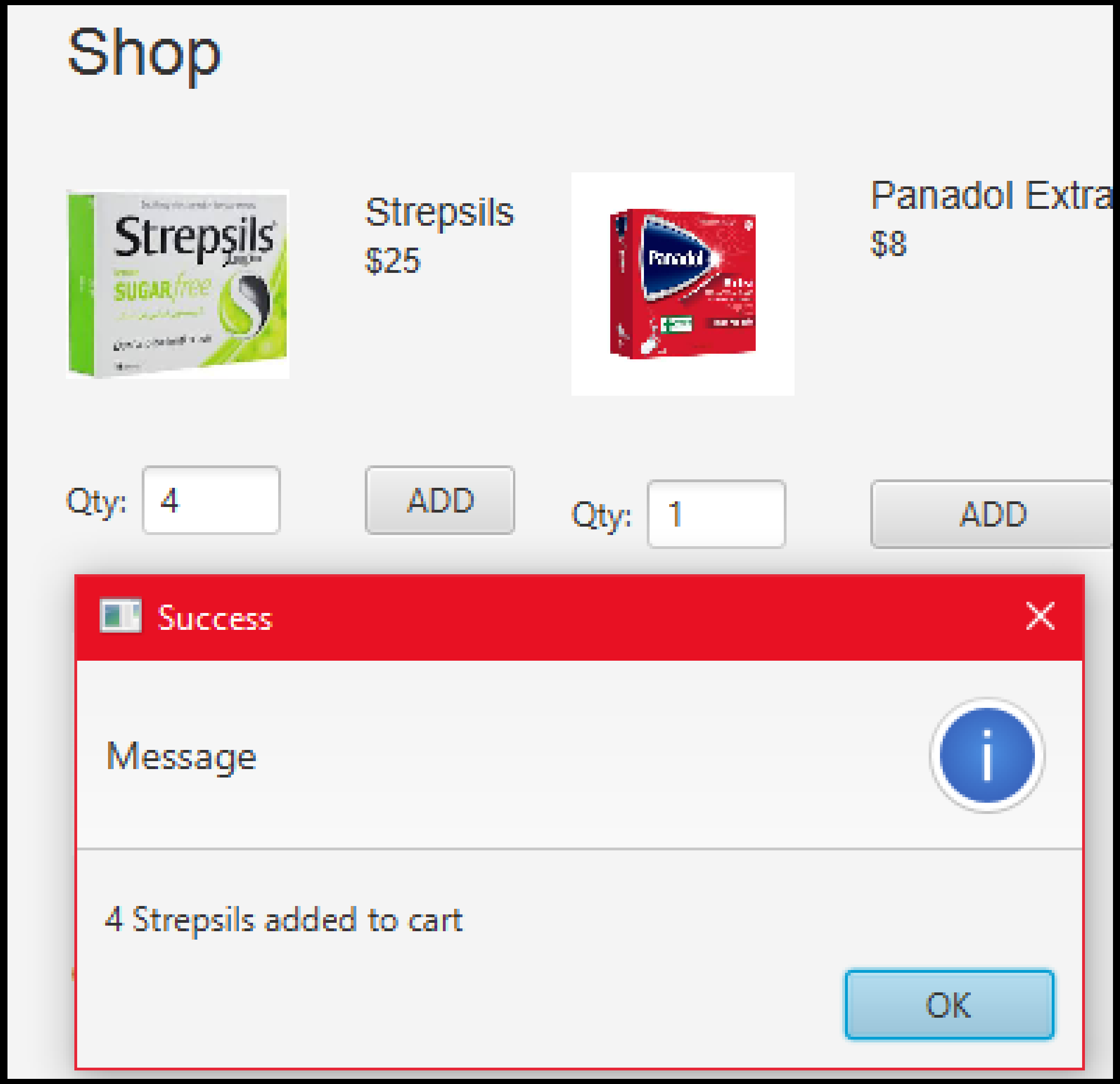
Message

Payment successful! Thank you for your purchase.

GUI - Event Handling

Add Button: calls addItem() from Order object

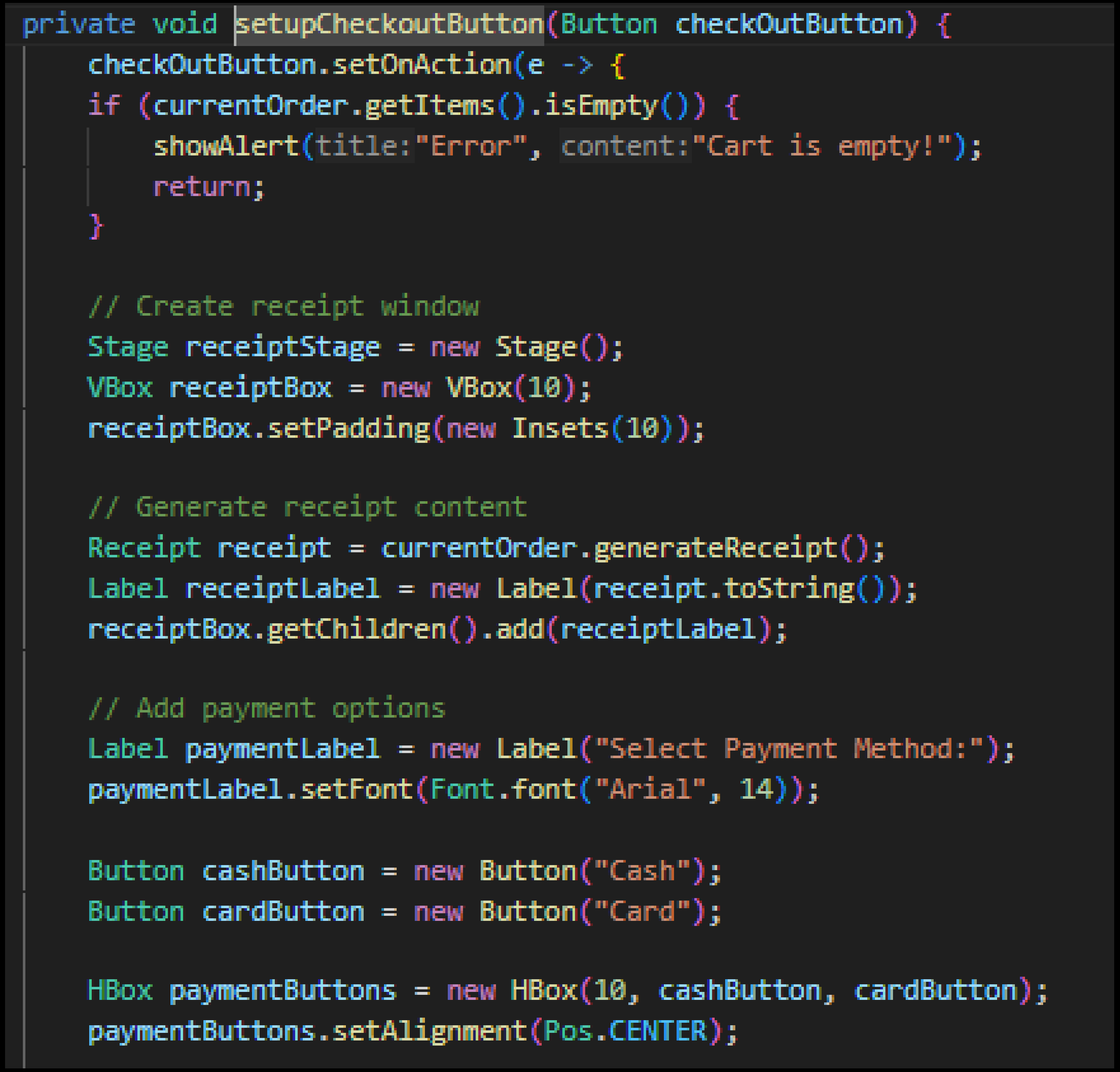
```
addButton.setOnAction(e -> {
try {
    int quantity = Integer.parseInt(quantityField.getText());
    Medicine medicine = medicineMap.get(name);
    if (medicine != null) {
        currentOrder.addItem(medicine, quantity);
        showAlert(title:"Success", quantity + " " + name + " added to cart");
    }
} catch (NumberFormatException ex) {
    showAlert(title:"Error", content:"Please enter a valid quantity");
} catch (Exception ex) {
    showAlert(title:"Error", ex.getMessage());
}
```



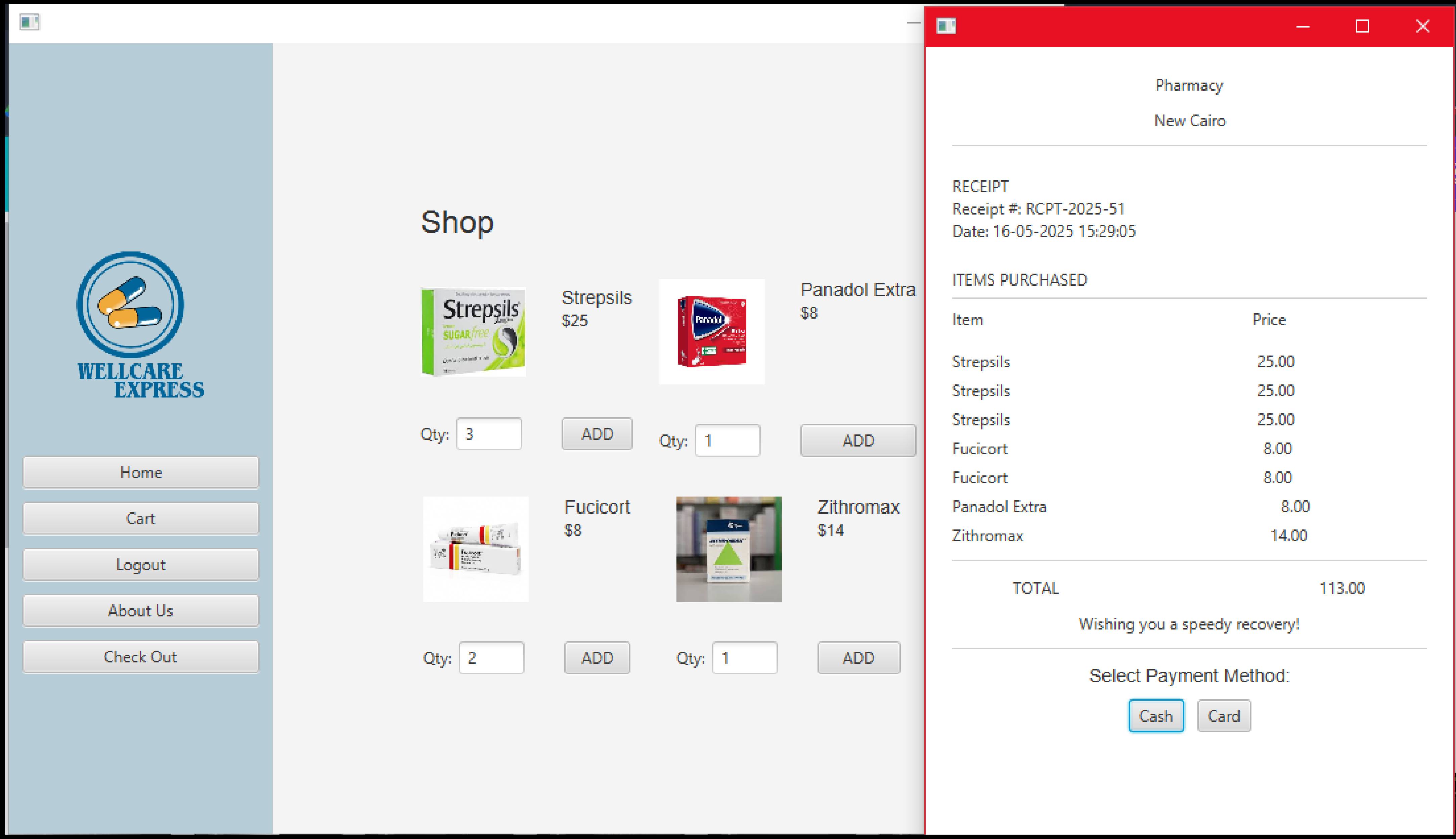
4 Strepsils added to cart

GUI - Event Handling

Checkout Button: calls generateReceipt on current order

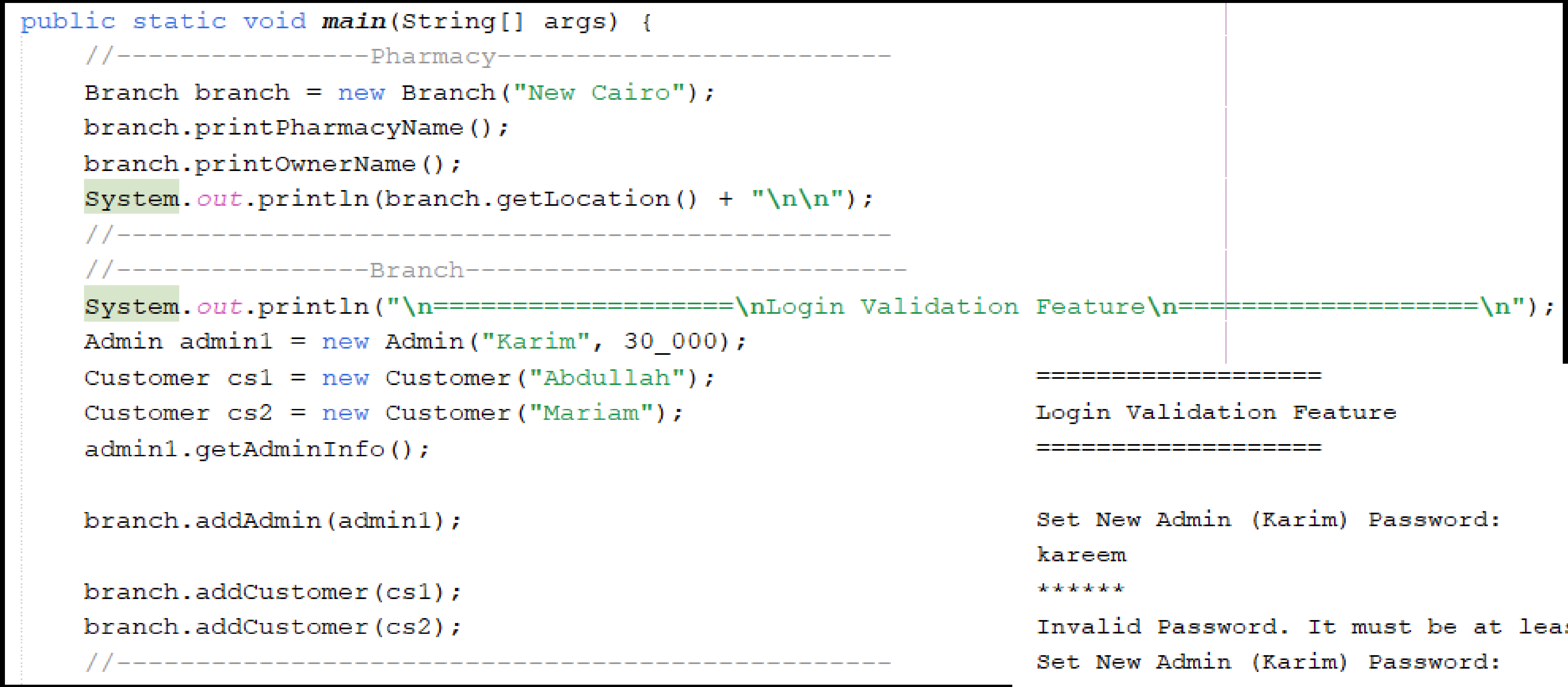


```java
private void setupCheckoutButton(Button checkOutButton) {
    checkOutButton.setOnAction(e -> {
    if (currentOrder.getItems().isEmpty()) {
        showAlert(title:"Error", content:"Cart is empty!");
        return;
    }

    // Create receipt window
    Stage receiptStage = new Stage();
    VBox receiptBox = new VBox(10);
    receiptBox.setPadding(new Insets(10));

    // Generate receipt content
    Receipt receipt = currentOrder.generateReceipt();
    Label receiptLabel = new Label(receipt.toString());
    receiptBox.getChildren().add(receiptLabel);

    // Add payment options
    Label paymentLabel = new Label("Select Payment Method:");
    paymentLabel.setFont(Font.font("Arial", 14));

    Button cashButton = new Button("Cash");
    Button cardButton = new Button("Card");

    HBox paymentButtons = new HBox(10, cashButton, cardButton);
    paymentButtons.setAlignment(Pos.CENTER);
```

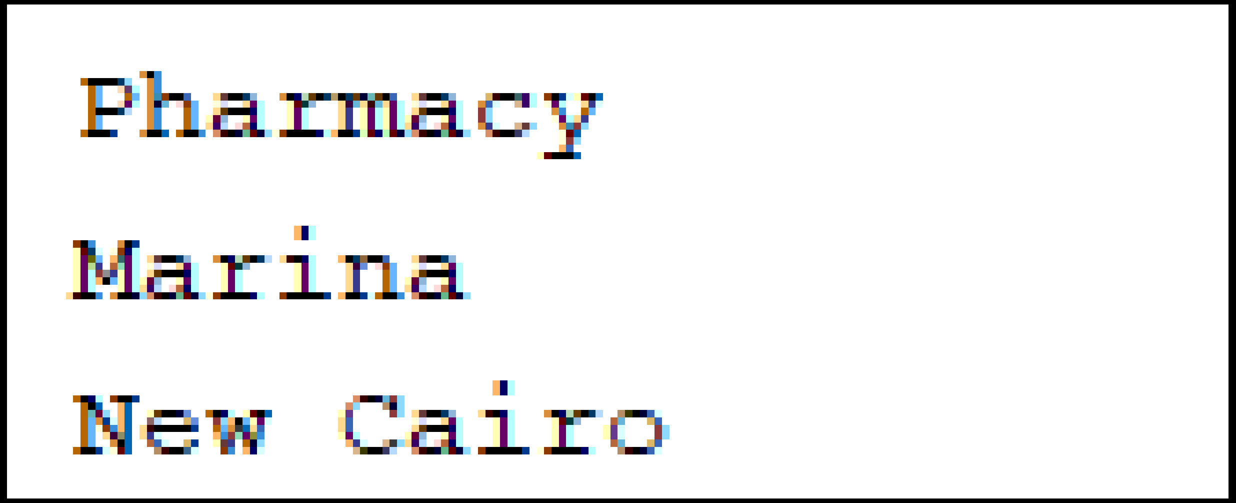


```
Pharmacy
New Cairo

RECEIPT
Receipt #: RCPT-2025-51
Date: 16-05-2025 15:29:05

ITEMS PURCHASED
Item                Price
Strepsils           25.00
Strepsils           25.00
Strepsils           25.00
Fucicort             8.00
Fucicort             8.00
Panadol Extra        8.00
Zithromax           14.00

TOTAL              113.00

Wishing you a speedy recovery!

Select Payment Method:
Cash   Card
```

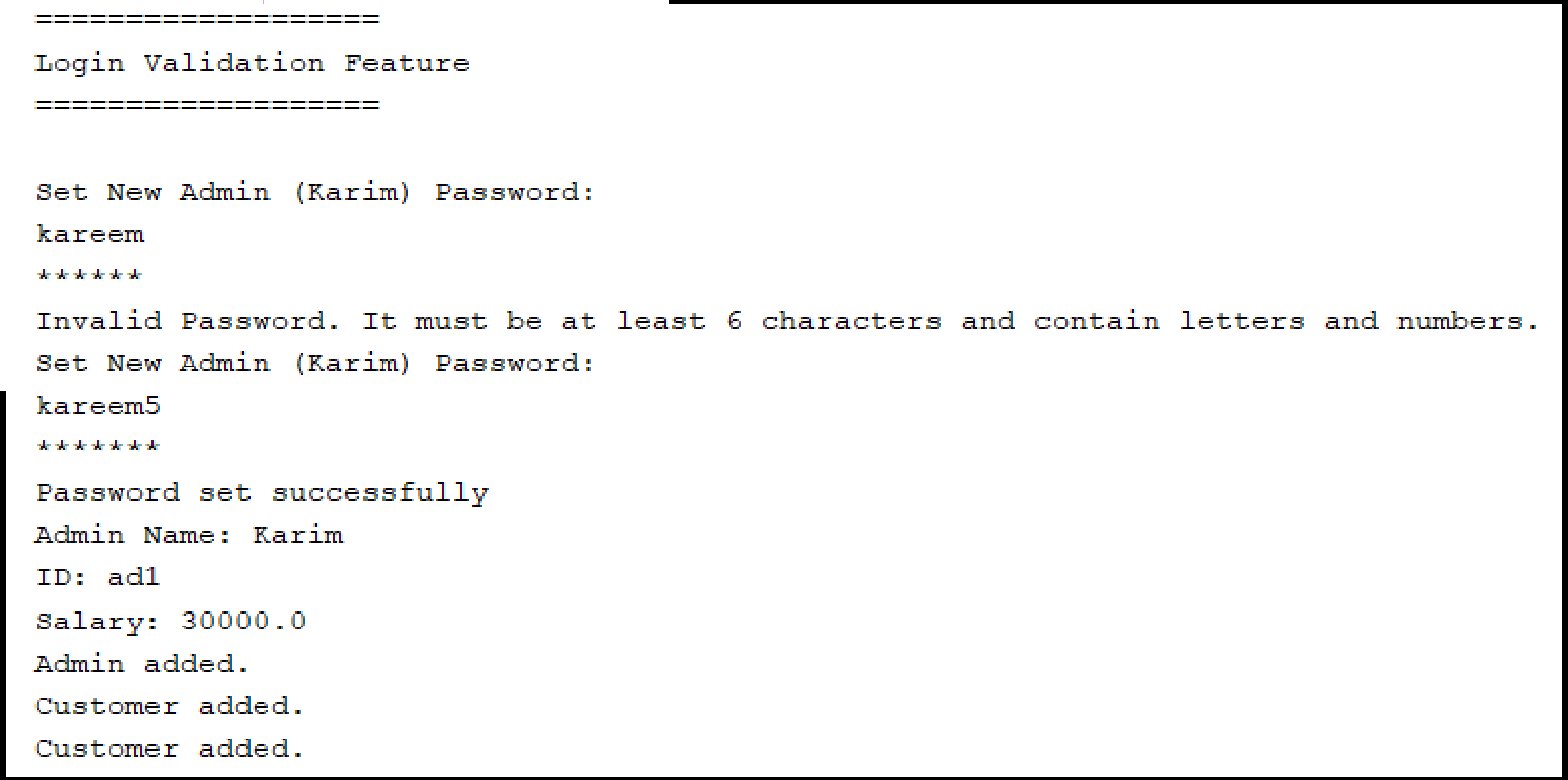
Full Demo: Login Validation



```
public static void main(String[] args) {
    //----------------Pharmacy-------------------------
    Branch branch = new Branch("New Cairo");
    branch.printPharmacyName();
    branch.printOwnerName();
    System.out.println(branch.getLocation() + "\n\n");
    //-------------------------------------------------
    //----------------Branch---------------------------
    System.out.println("\n===================\nLogin Validation Feature\n===================\n");
    Admin admin1 = new Admin("Karim", 30_000);
    Customer cs1 = new Customer("Abdullah");
    Customer cs2 = new Customer("Mariam");
    admin1.getAdminInfo();

    branch.addAdmin(admin1);

    branch.addCustomer(cs1);
    branch.addCustomer(cs2);
    //-------------------------------------------------
```

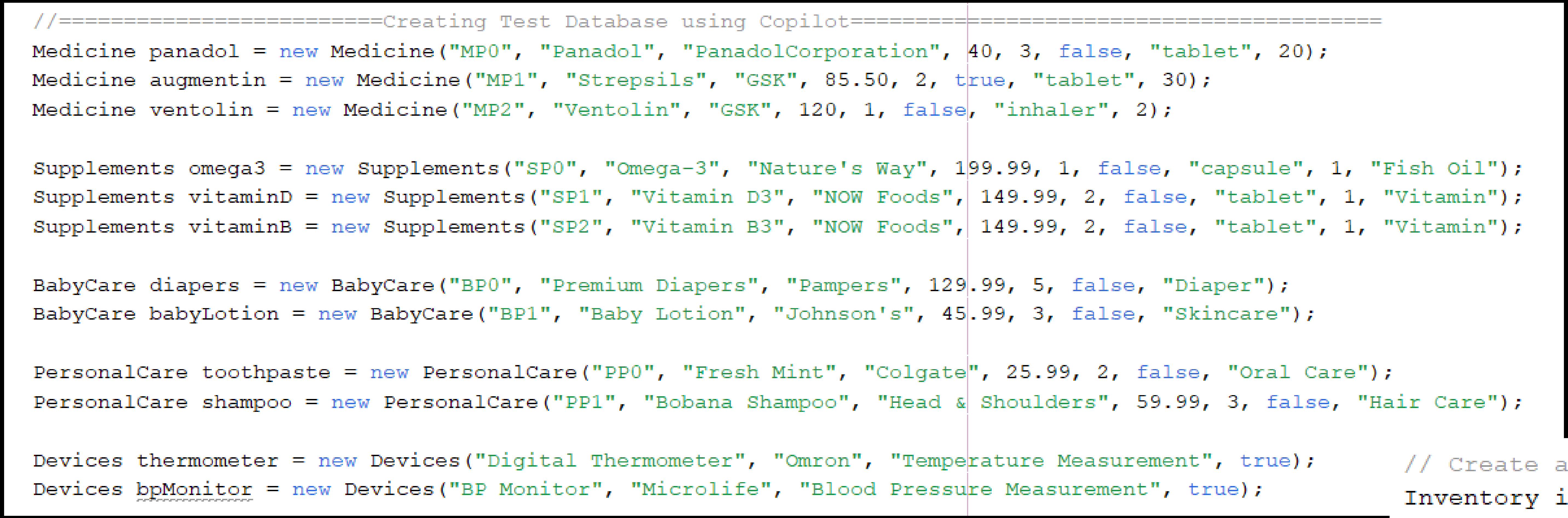


```
Pharmacy
Marina
New Cairo
```

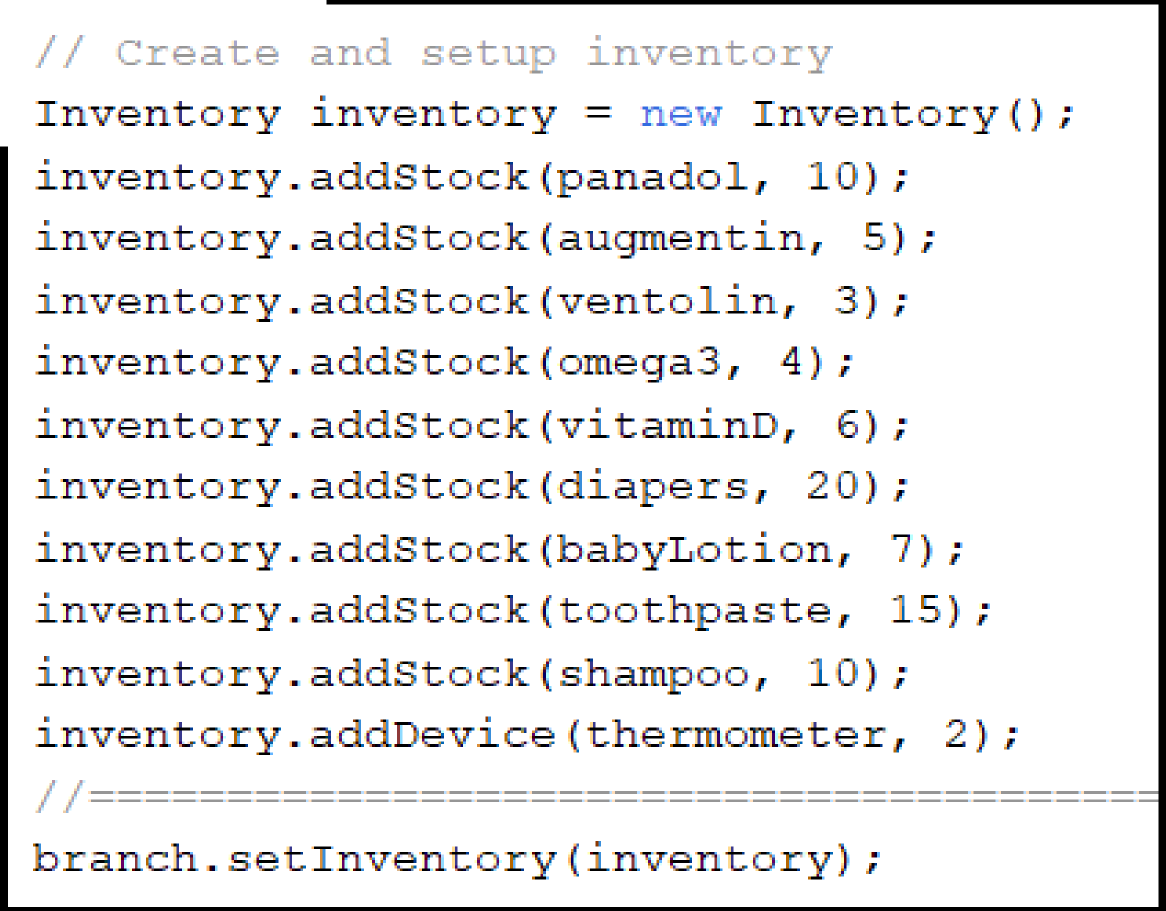


```
===================
Login Validation Feature
===================

Set New Admin (Karim) Password:
kareem
******
Invalid Password. It must be at least 6 characters and contain letters and numbers.
Set New Admin (Karim) Password:
kareem5
*******
Password set successfully
Admin Name: Karim
ID: ad1
Salary: 30000.0
Admin added.
Customer added.
Customer added.
```

Full Demo: Inventory Setup

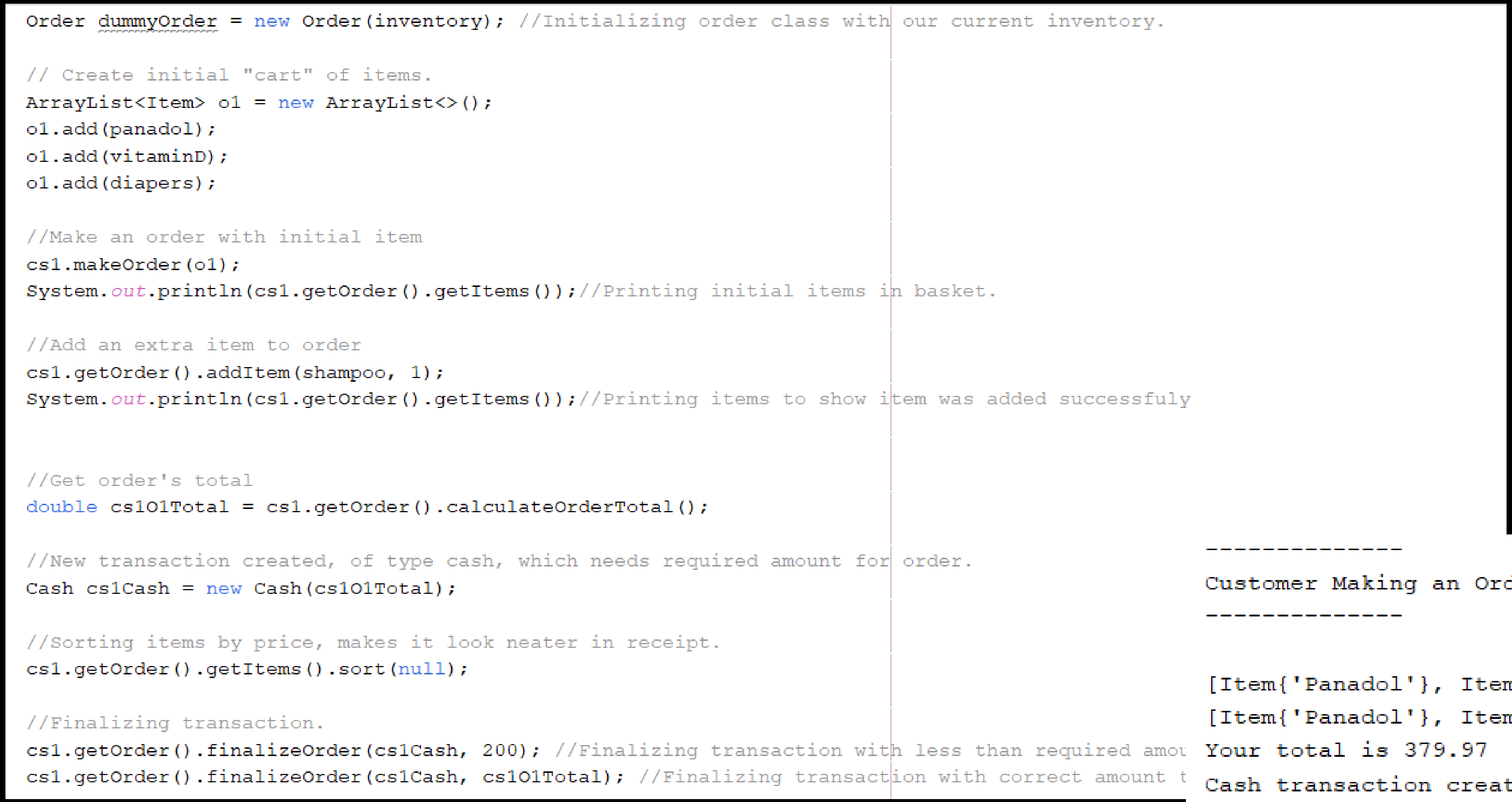


```java
//========================Creating Test Database using Copilot======================================
Medicine panadol = new Medicine("MP0", "Panadol", "PanadolCorporation", 40, 3, false, "tablet", 20);
Medicine augmentin = new Medicine("MP1", "Strepsils", "GSK", 85.50, 2, true, "tablet", 30);
Medicine ventolin = new Medicine("MP2", "Ventolin", "GSK", 120, 1, false, "inhaler", 2);

Supplements omega3 = new Supplements("SP0", "Omega-3", "Nature's Way", 199.99, 1, false, "capsule", 1, "Fish Oil");
Supplements vitaminD = new Supplements("SP1", "Vitamin D3", "NOW Foods", 149.99, 2, false, "tablet", 1, "Vitamin");
Supplements vitaminB = new Supplements("SP2", "Vitamin B3", "NOW Foods", 149.99, 2, false, "tablet", 1, "Vitamin");

BabyCare diapers = new BabyCare("BP0", "Premium Diapers", "Pampers", 129.99, 5, false, "Diaper");
BabyCare babyLotion = new BabyCare("BP1", "Baby Lotion", "Johnson's", 45.99, 3, false, "Skincare");

PersonalCare toothpaste = new PersonalCare("PP0", "Fresh Mint", "Colgate", 25.99, 2, false, "Oral Care");
PersonalCare shampoo = new PersonalCare("PP1", "Bobana Shampoo", "Head & Shoulders", 59.99, 3, false, "Hair Care");

Devices thermometer = new Devices("Digital Thermometer", "Omron", "Temperature Measurement", true);
Devices bpMonitor = new Devices("BP Monitor", "Microlife", "Blood Pressure Measurement", true);
```

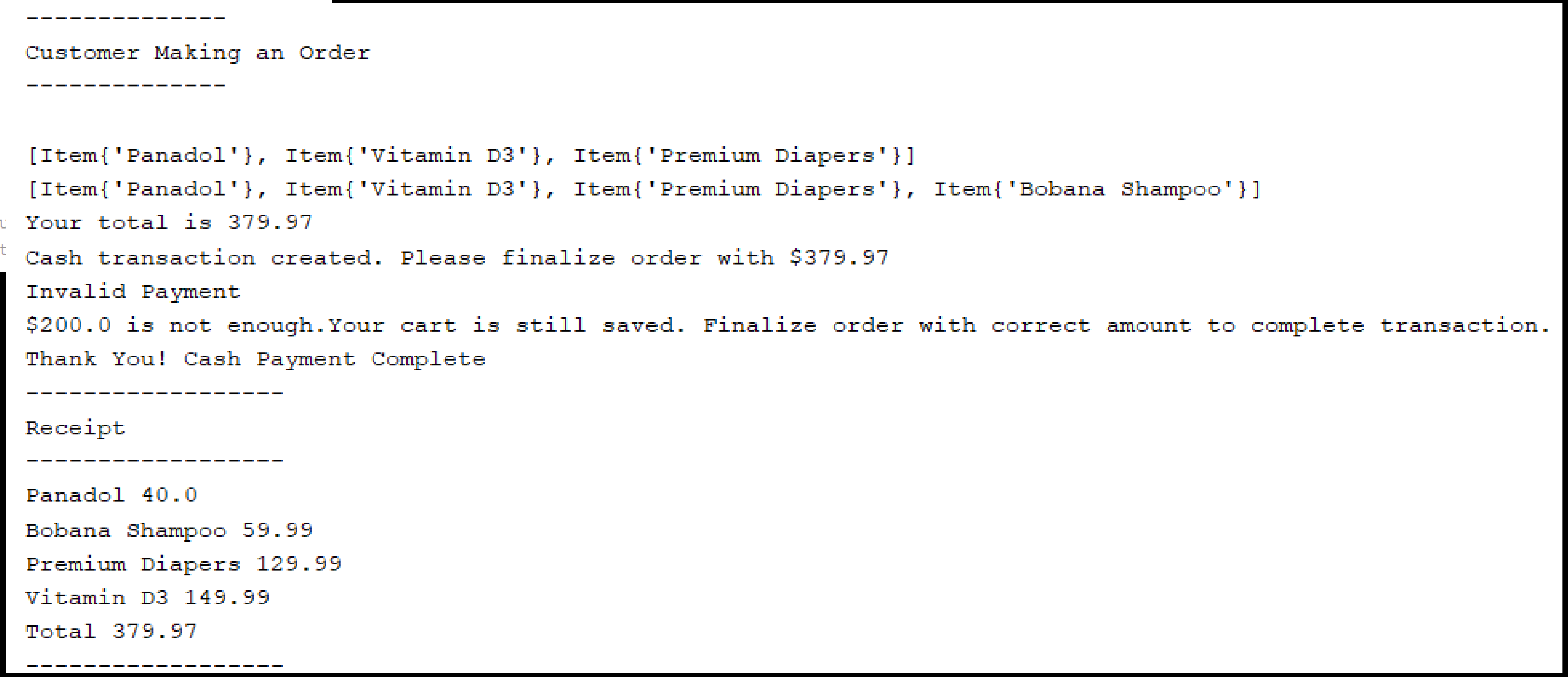


```java
// Create and setup inventory
Inventory inventory = new Inventory();
inventory.addStock(panadol, 10);
inventory.addStock(augmentin, 5);
inventory.addStock(ventolin, 3);
inventory.addStock(omega3, 4);
inventory.addStock(vitaminD, 6);
inventory.addStock(diapers, 20);
inventory.addStock(babyLotion, 7);
inventory.addStock(toothpaste, 15);
inventory.addStock(shampoo, 10);
inventory.addDevice(thermometer, 2);
//=======================================
branch.setInventory(inventory);
```

Full Demo: Making Order

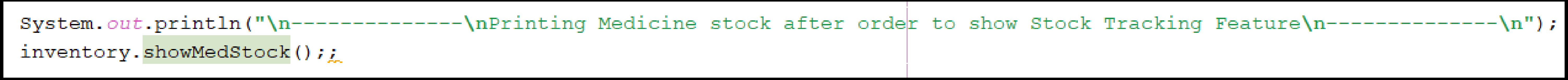


```java
Order dummyOrder = new Order(inventory); //Initializing order class with our current inventory.

// Create initial "cart" of items.
ArrayList<Item> o1 = new ArrayList<>();
o1.add(panadol);
o1.add(vitaminD);
o1.add(diapers);

//Make an order with initial item
cs1.makeOrder(o1);
System.out.println(cs1.getOrder().getItems());//Printing initial items in basket.

//Add an extra item to order
cs1.getOrder().addItem(shampoo, 1);
System.out.println(cs1.getOrder().getItems());//Printing items to show item was added successfuly


//Get order's total
double cs1O1Total = cs1.getOrder().calculateOrderTotal();

//New transaction created, of type cash, which needs required amount for order.
Cash cs1Cash = new Cash(cs1O1Total);

//Sorting items by price, makes it look neater in receipt.
cs1.getOrder().getItems().sort(null);

//Finalizing transaction.
cs1.getOrder().finalizeOrder(cs1Cash, 200); //Finalizing transaction with less than required amou
cs1.getOrder().finalizeOrder(cs1Cash, cs1O1Total); //Finalizing transaction with correct amount t
```

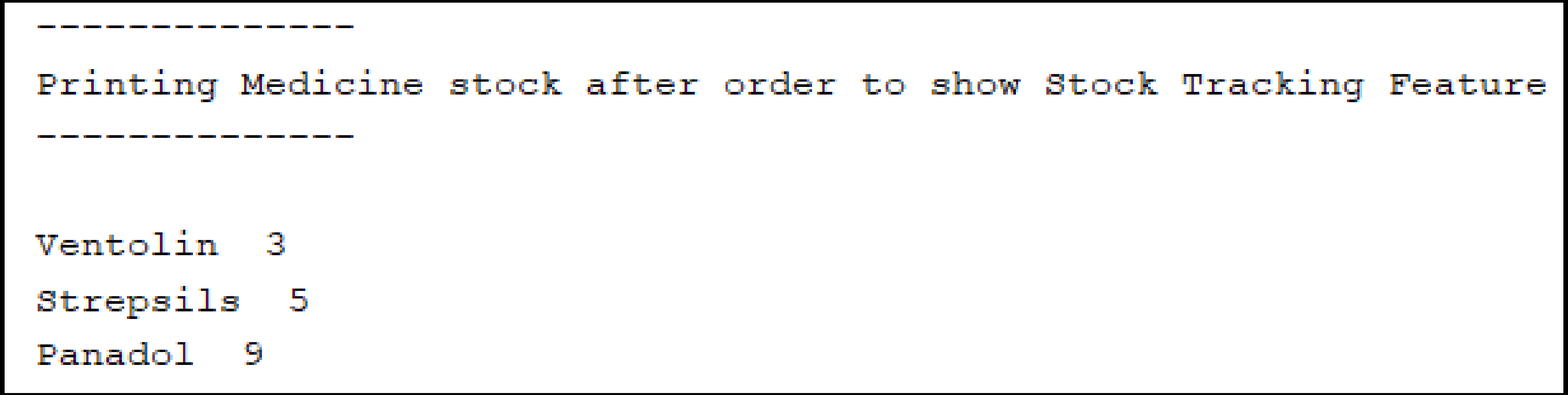


```
--------------
Customer Making an Order
--------------

[Item{'Panadol'}, Item{'Vitamin D3'}, Item{'Premium Diapers'}]
[Item{'Panadol'}, Item{'Vitamin D3'}, Item{'Premium Diapers'}, Item{'Bobana Shampoo'}]
Your total is 379.97
Cash transaction created. Please finalize order with $379.97
Invalid Payment
$200.0 is not enough.Your cart is still saved. Finalize order with correct amount to complete transaction.
Thank You! Cash Payment Complete
------------------
Receipt
------------------
Panadol 40.0
Bobana Shampoo 59.99
Premium Diapers 129.99
Vitamin D3 149.99
Total 379.97
------------------
```

Full Demo: Stock Tracking Feature

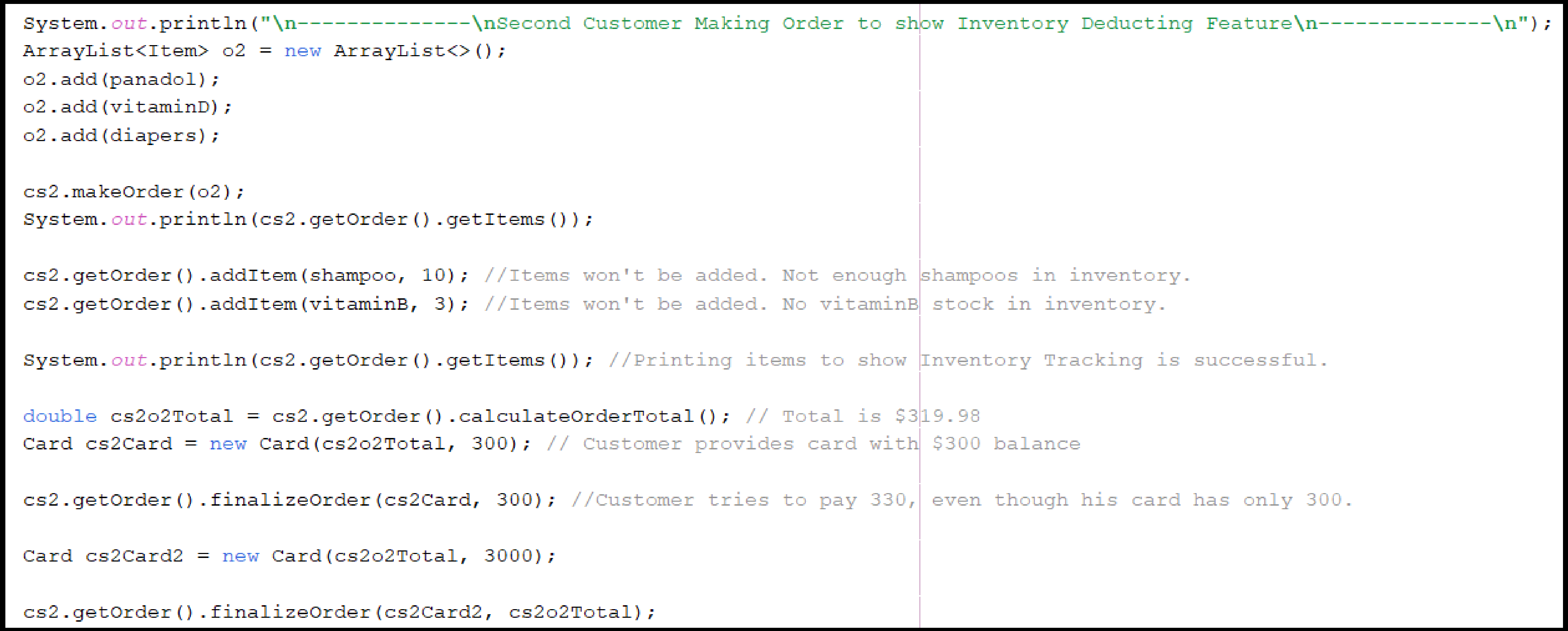


```java
System.out.println("\n--------------\nPrinting Medicine stock after order to show Stock Tracking Feature\n--------------\n");
inventory.showMedStock();;
```

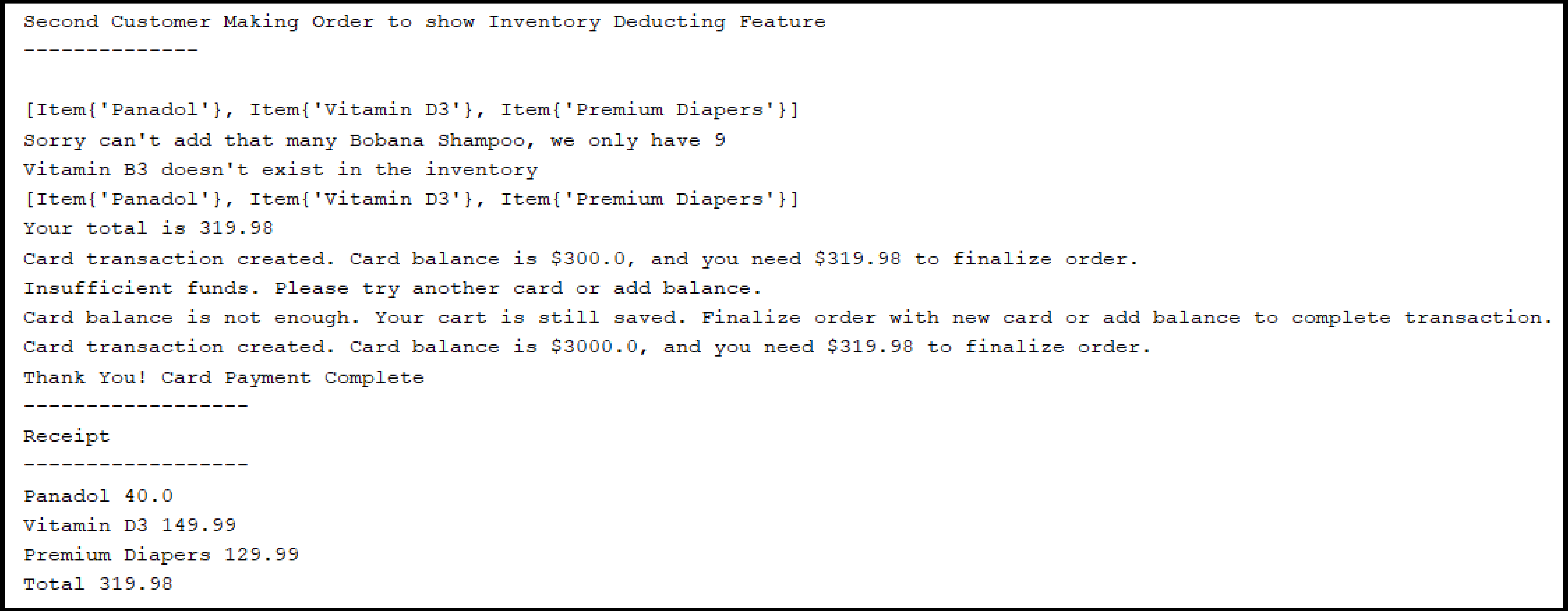


```
--------------
Printing Medicine stock after order to show Stock Tracking Feature
--------------

Ventolin  3
Strepsils  5
Panadol  9
```

Full Demo: Inventory Deducting Feature

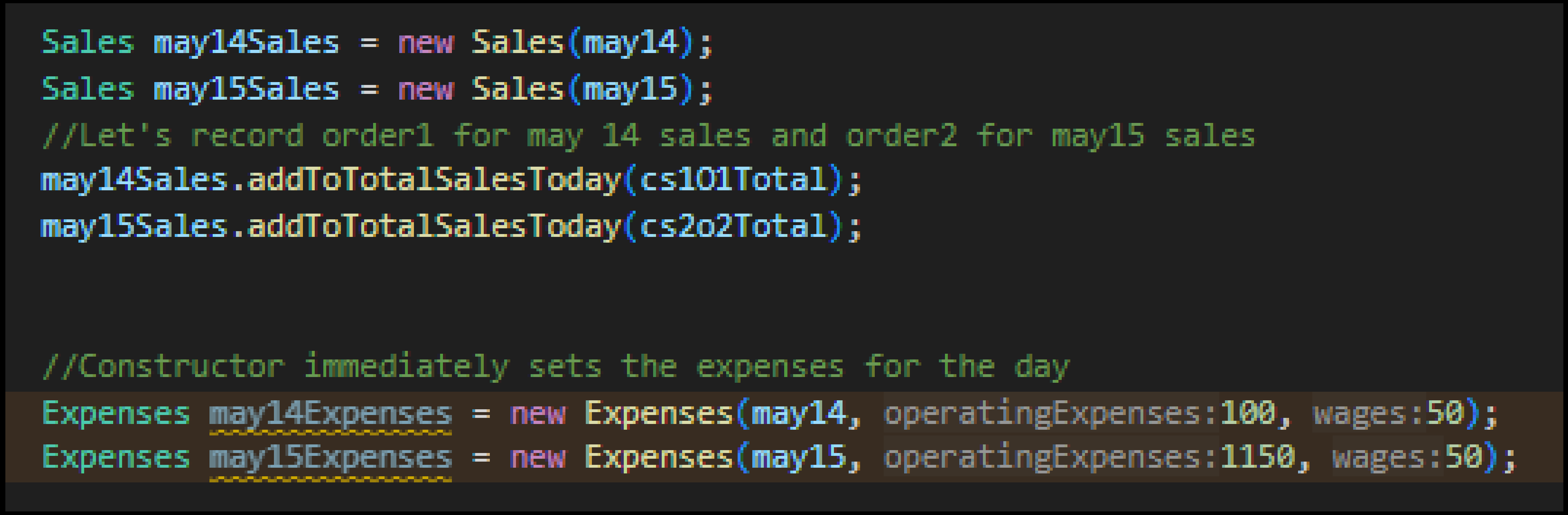


```java
System.out.println("\n--------------\nSecond Customer Making Order to show Inventory Deducting Feature\n--------------\n");
ArrayList<Item> o2 = new ArrayList<>();
o2.add(panadol);
o2.add(vitaminD);
o2.add(diapers);

cs2.makeOrder(o2);
System.out.println(cs2.getOrder().getItems());

cs2.getOrder().addItem(shampoo, 10); //Items won't be added. Not enough shampoos in inventory.
cs2.getOrder().addItem(vitaminB, 3); //Items won't be added. No vitaminB stock in inventory.

System.out.println(cs2.getOrder().getItems()); //Printing items to show Inventory Tracking is successful.

double cs2o2Total = cs2.getOrder().calculateOrderTotal(); // Total is $319.98
Card cs2Card = new Card(cs2o2Total, 300); // Customer provides card with $300 balance

cs2.getOrder().finalizeOrder(cs2Card, 300); //Customer tries to pay 330, even though his card has only 300.

Card cs2Card2 = new Card(cs2o2Total, 3000);

cs2.getOrder().finalizeOrder(cs2Card2, cs2o2Total);
```

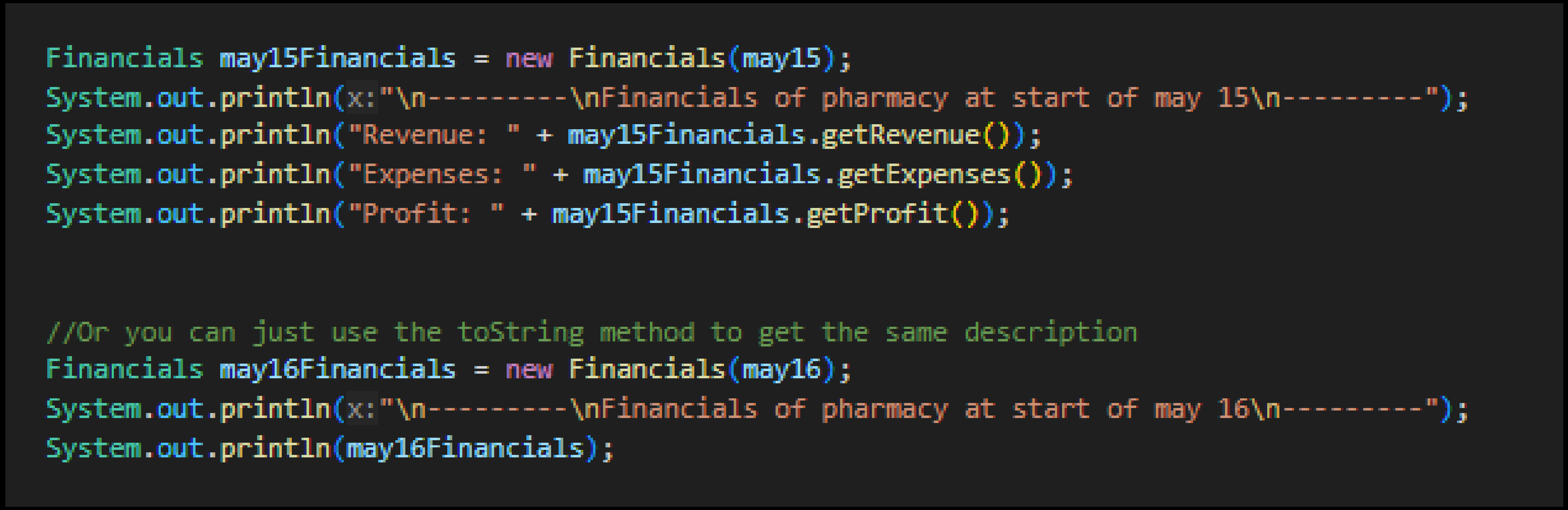


```
Second Customer Making Order to show Inventory Deducting Feature
--------------

[Item{'Panadol'}, Item{'Vitamin D3'}, Item{'Premium Diapers'}]
Sorry can't add that many Bobana Shampoo, we only have 9
Vitamin B3 doesn't exist in the inventory
[Item{'Panadol'}, Item{'Vitamin D3'}, Item{'Premium Diapers'}]
Your total is 319.98
Card transaction created. Card balance is $300.0, and you need $319.98 to finalize order.
Insufficient funds. Please try another card or add balance.
Card balance is not enough. Your cart is still saved. Finalize order with new card or add balance to complete transaction.
Card transaction created. Card balance is $3000.0, and you need $319.98 to finalize order.
Thank You! Card Payment Complete
------------------
Receipt
------------------
Panadol 40.0
Vitamin D3 149.99
Premium Diapers 129.99
Total 319.98
```

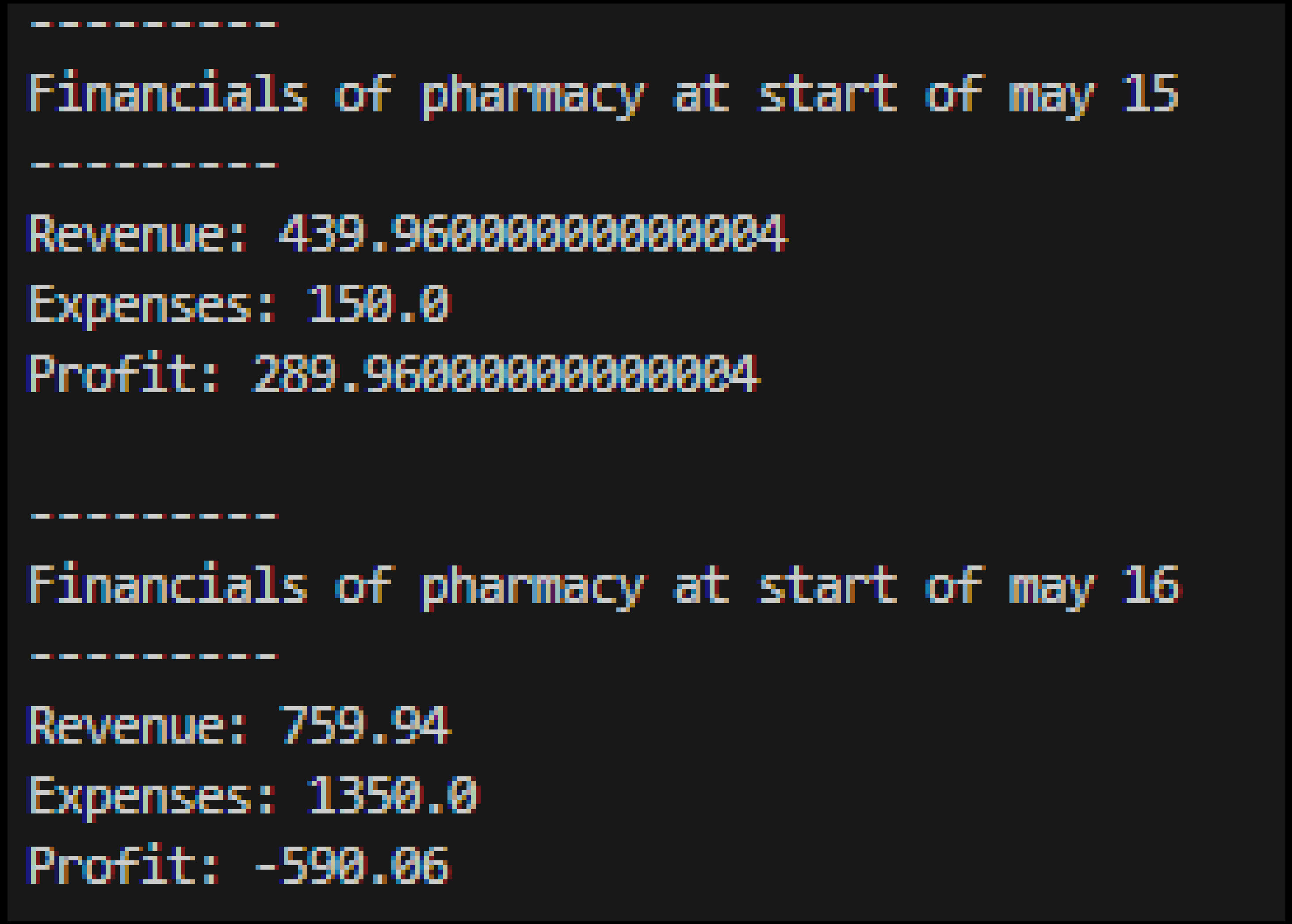
Full Demo: Financials



```java
Sales may14Sales = new Sales(may14);
Sales may15Sales = new Sales(may15);
//Let's record order1 for may 14 sales and order2 for may15 sales
may14Sales.addToTotalSalesToday(cs1o1Total);
may15Sales.addToTotalSalesToday(cs2o2Total);


//Constructor immediately sets the expenses for the day
Expenses may14Expenses = new Expenses(may14, operatingExpenses:100, wages:50);
Expenses may15Expenses = new Expenses(may15, operatingExpenses:1150, wages:50);
```



```java
Financials may15Financials = new Financials(may15);
System.out.println(x:"\n---------\nFinancials of pharmacy at start of may 15\n---------");
System.out.println("Revenue: " + may15Financials.getRevenue());
System.out.println("Expenses: " + may15Financials.getExpenses());
System.out.println("Profit: " + may15Financials.getProfit());


//Or you can just use the toString method to get the same description
Financials may16Financials = new Financials(may16);
System.out.println(x:"\n---------\nFinancials of pharmacy at start of may 16\n---------");
System.out.println(may16Financials);
```



```
---------
Financials of pharmacy at start of may 15
---------
Revenue: 439.96000000000004
Expenses: 150.0
Profit: 289.96000000000004


---------
Financials of pharmacy at start of may 16
---------
Revenue: 759.94
Expenses: 1350.0
Profit: -590.06
```

THANK YOU