NewGen POS

CS 462

Wednesday, Dec. 12, 2012

Wei Jen Lin
Xinchao Liu
Michael Robertson
Bryan Tamada
Wee Siang Wong

Software Development Process

Development Process

- Scrum
- XP

Software Tools

- MS Office (Visio, Word, etc.)
- Java IDE (Eclipse, NetBeans, XCode, etc.)
- GUI Framework (Qt)
- Version Control (GitHub)
- Testing (JUnit)

Database

- MySQL

Software Development Process

Use-Case:

Process Sale

ID: 001

Name: Process Sale

Description:

The POS system will record purchased items, calculate a total along with product names, customer payment method, update store inventory and print out a summary of purchased items in the form of a receipt for the customer.

Actors: cashier

Pre-condition:

cashier ready

Basic steps:

1. customer checkout with goods
2. cashier start a new sale
3. cashier enter items
4. system present item name and price

cashier repeat step 3-4 until record all goods

5. system present total price with tax
6. cashier asks for payment
7. customer pays and system handles payment
8. system logs sales
9. system prints receipt
10. customer leaves with receipt and goods

Post-condition:

Amount compute correctly, payment authorized, sales saved, inventory updated, receipt generated.

Priority: high

Special requirements:

large font text to guarantee screen visibility
quick payment authorization response

Memo (open issues):

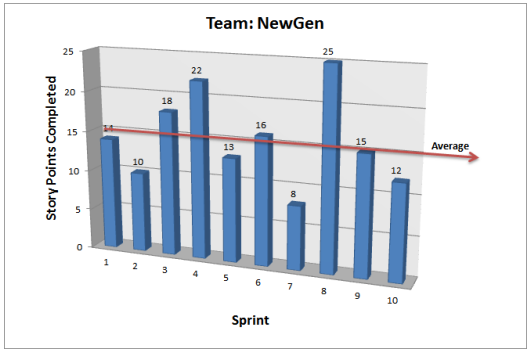
need to support manager's override operation

Software Development Process

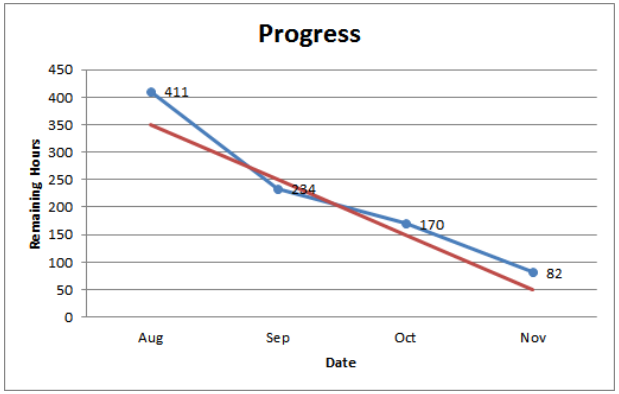
- **Sprint Backlog**

Who	Description				
	Total Estimated Hours:	350	250	150	50
	Environment				
Michel	Install Git	4	2	2	
Michel	Setup GitHub	4	4	4	4
Will	Setup JDK	4	2	2	
Will	Install QT	4	2	2	
Lin	Setup MYSQL	4	4	4	4
	Database				
Bryan	Design Database	20	16	12	
Leo	Create MYSQL tables	15	12	8	
Michel	Connect POS system to database	8	4	4	4
Will	Add sales records in database	8	2		
	Design Model				
Leo	Domain Model	24	12	8	
Bryan	Sequence Diagram	24	12	8	
Lin	CRC models	24	16	12	
Will	Design System based on GRASP RDD pattern	16	8	8	
Michel	Check OO Design Principles	16	8	8	8
	Implementation				
Michel	Polymorphic Operation Example	12	8		
Will	Add test template	12	8	4	2
Leo	Add input testing for productID and qty	12	4	4	
Bryan	Move main method	12	4	4	2
Leo	Make class variables private	12	4	4	2
Bryan	Incorporating product catalog, register, store objects	12	8		
All	Build Objects	40	40	40	24
Will	Receipt Class overhaul	8	8		
Michel	Make payment class working	16	8	8	8
	Test				
Leo	Test using test template	24	12	8	8
Bryan	Test all classes seprately	24	12	8	8
Lin	Test objects functionality	24	8	2	2
Will	Report test results	12	2	2	2
Michel	Remove extra methods	8	2	2	2
Leo	Clean up all codes	8	2	2	2

- **Sprint Velocity Chart**

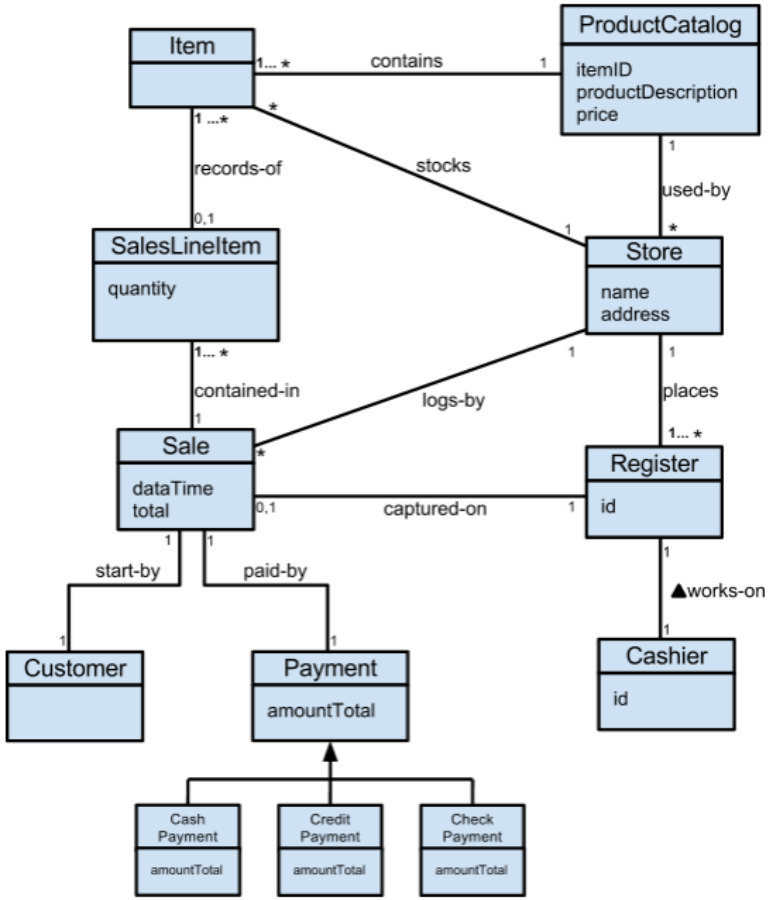


- **Burn-down Chart**



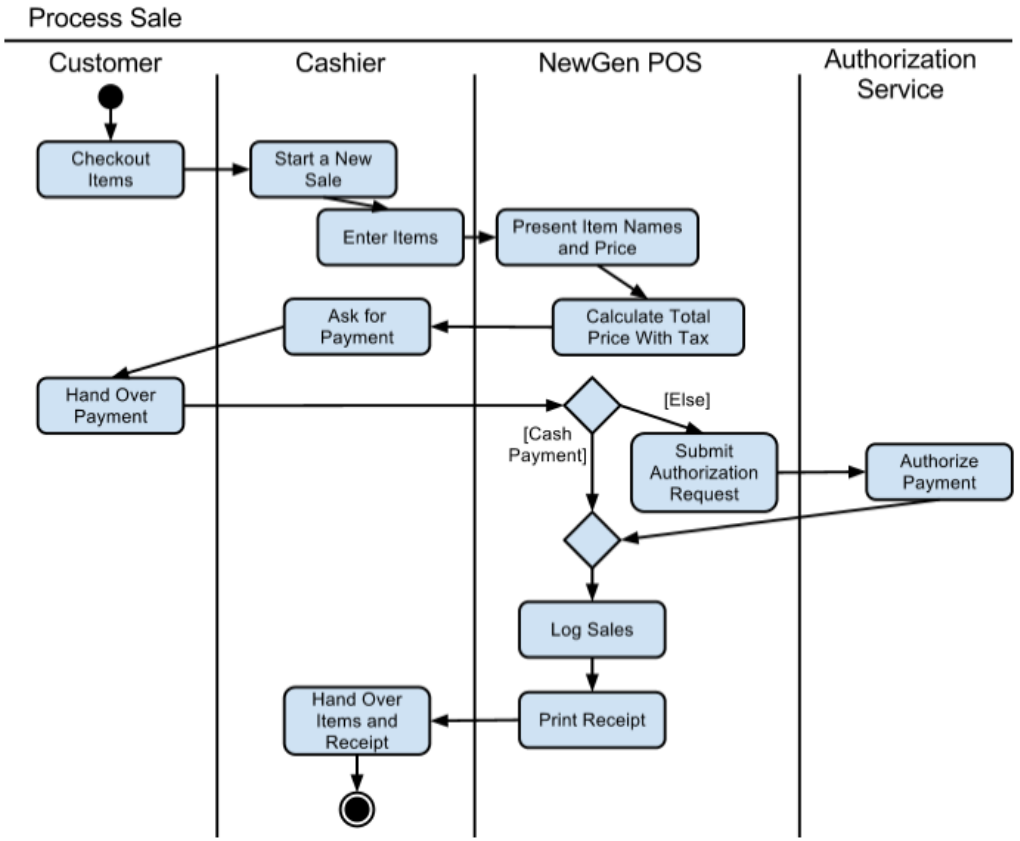
System Architecture

Domain Model



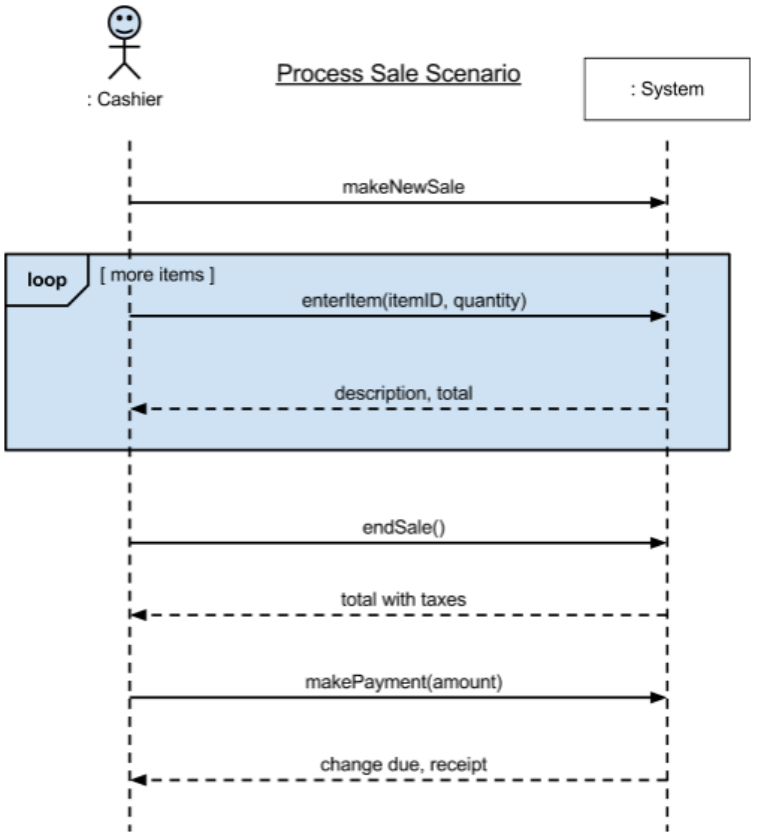
System Architecture

Activity Diagram



System Architecture

System Operations



System Architecture

CRC Model

Class: Store	
Responsibility:	**Collaborator:**
maintains product catalog	ProductCatalog
stores items	Item
logs sale orders	Sale
manages register status	Register

Class: ProductCatalog	
Responsibility:	**Collaborator:**
manages item id/price	Item, ItemID, Money
stores item descriptions	Item
genarates catalog for store	Store

Class: Item	
Responsibility:	**Collaborator:**
define item id/price	ProductCatalog
define stock status	Store

Class: ItemID	
Responsibility:	**Collaborator:**
keeps item ID in right format	

Class: Sale	
Responsibility:	**Collaborator:**
adds SaleLineItem	SaleLineItem
logs sale date/time	Store
logs sale total payment	Payment
defines sale status	Interface, Register

Class: SaleLineItem	
Responsibility:	**Collaborator:**
shows item id/price	Item
shows item quantity	Sale

Class: Payment	
Responsibility:	**Collaborator:**
defines payment total amount	Sale
computes amount for change	Money
verifies payment	Register

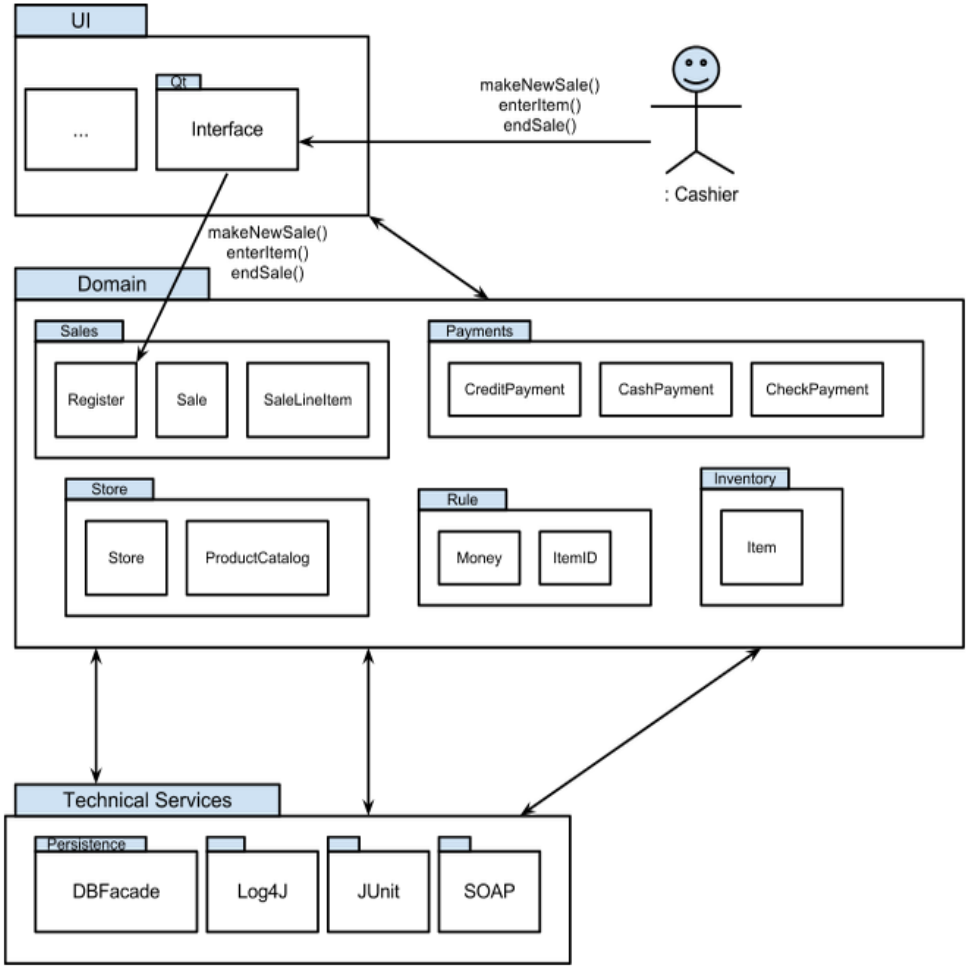
Class: Register	
Responsibility:	**Collaborator:**
define register status	Store
starts a new sale	Sale
adds item to sale	SaleLineItem, ItemID
accepts tendered money	Payment, Money

Class: Money	
Responsibility:	**Collaborator:**
converts price to right format	

Class: Interface	
Responsibility:	**Collaborator:**
displays sale detail	Sale, SaleLineItem
displays price	Money
clears cart	
accepts register's commands	Register

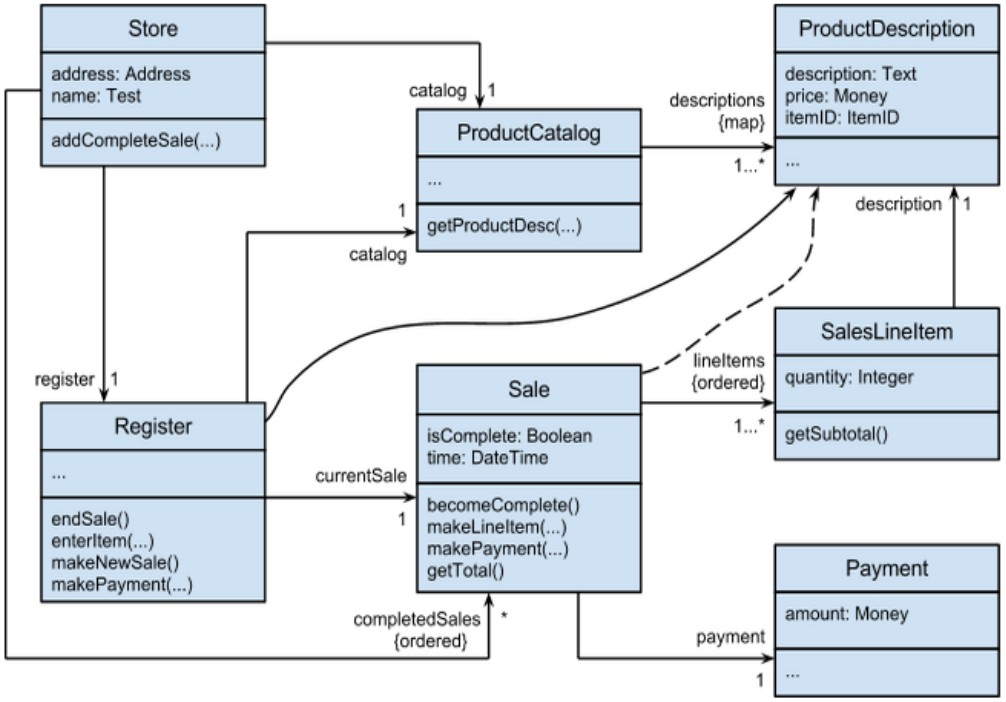
System Architecture

Architecture Diagram



Design Decisions

Design Class Diagram



Design Decisions

GRASP RDD Pattern

- **Creator**
- **Information Expert**
- **Low Coupling**
- **High Cohesion**
- **Controller**
- **Polymorphism**
- **Pure Fabrication**
- **Indirection**
- **Protection Variations**

Design Decisions

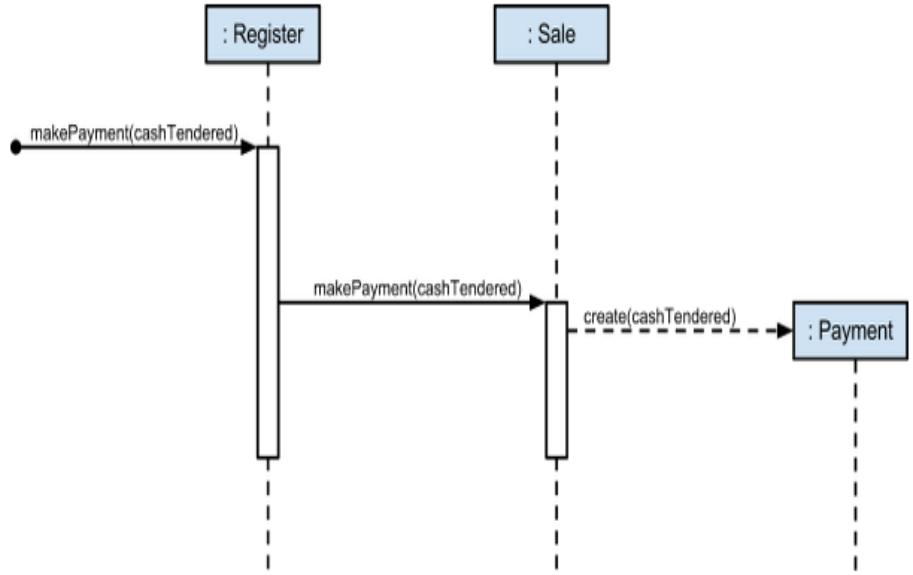
OO Design Principles

- **Command-Query Separation Principle**
- **Law of Demeter**
- **Single-Responsibility Principle**
- **Open-Closed Principle**
- **Liskov Substitution Principle**
- **Dependency-Inversion Principle**
- **Interface Segregation Principle**

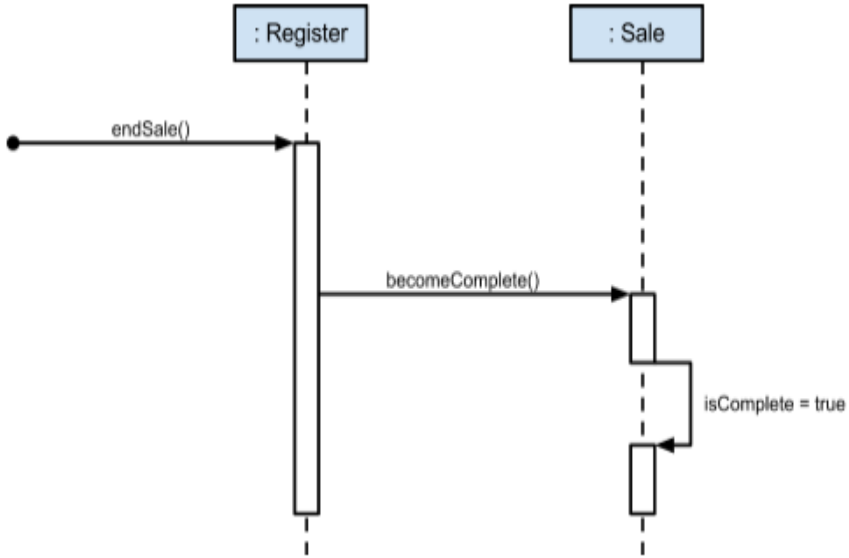
Design Decisions

Object Design Models

- Sequence Diagram for makePayment



- Sequence Diagram for endSale



Design Decisions

Design Patterns

- **Factory**
- **Singleton**
- **Facade**
- **Adapter**
- **Strategy**
- **Observer**

Implementation

Codes

```
142     public void makeNewSale(){
143         currentSale = new Sale();
144     }
145     public void makePayment(int paymentMethod){
146         PaymentFactory factoryObj = new PaymentFactory(paymentMethod);
147         Payment payment = factoryObj.getPayment(this.paymentAmount);
148         currentSale.makePayment(payment);
149     }
150     //Cash Payment
151     public boolean makeCashPayment(int paymentMethod, String input){
152         try{
153             Double paymentInput = Double.parseDouble(input);
154             this.paymentAmount = new Money(paymentInput);
155             this.total = currentSale.getTotal();
156         }
157         catch(NumberFormatException e){
158             Ui_NewGenPOS.setText("Payment amount must contain ONLY numbers and must NOT be blank! Try Again!");
159             return false;
160         }
161         if(this.total.checkEquals(new Money(0))){
162             Ui_NewGenPOS.setText("Cart is empty, add item and try again!");
163             return false;
164         }
165 
166         //Ensure payment is >= the total
167         boolean success = adapter.verifyPayment(this.paymentAmount, this.total);
168         if(success){
169             this.makePayment(paymentMethod);
170         }
171         return success;
172 
173     }
174     //Credit Payment
175     public boolean makeCreditPayment(int paymentMethod, String inputAmount, String inputCardNumber,
176             String inputYear, String inputMonth, String inputName){
177         try{
178             Double paymentInput = Double.parseDouble(inputAmount);
179             this.paymentAmount = new Money(paymentInput);
180             this.total = currentSale.getTotal();
181         }
182         catch(NumberFormatException e){
183             Ui_NewGenPOS.setText("Payment amount must contain ONLY numbers and must NOT be blank! Try Again!");
184             return false;
185         }
186 
187         if(this.total.checkEquals(new Money(0))){
188             Ui_NewGenPOS.setText("Cart is empty, add item and try again!");
189             return false;
190         }
191 
192         boolean success;
193         success = adapter.verifyPayment(this.paymentAmount, this.total, inputName,
194                 inputCardNumber, inputMonth, inputYear);
195 
196         if(success){
197             this.makePayment(paymentMethod);
198             this.customerName = inputName;
199             this.cardNumber = "XXXXXXXXXXXX"+inputCardNumber.substring(12);
200         }
201         return success;
```

CS462-Project / NewGenPOS / src / **newgenpos** /

Interface update done, total display changes when checkbox is clicked

mirob2005 authored 7 days ago

DBFacade.java	7 days ago	Facade Design Pattern Added
IPricingStrategy.java	7 days ago	Interface update done, total dis
IVerifyPayment.java	8 days ago	Adding a 2nd test to Money,Sa
ItemID.java	a month ago	All objects except payment cla
Main.java	a month ago	Receipt overhaul done, partial
Money.java	9 days ago	2 tests done for HW5.2 [mirob
Payment.java	a month ago	Add item dialog added with scr
PaymentFactory.java	7 days ago	Added factory design pattern [
Pricing.java	7 days ago	Working on getting discount to
ProductCatalog.java	7 days ago	Facade Design Pattern Added
ProductDescription.java	a month ago	All objects except payment cla
Register.java	7 days ago	Interface update done, total dis
Sale.java	7 days ago	Interface update done, total dis
SalesLineItem.java	a month ago	All objects except payment cla
SeniorDiscountPricing.java	7 days ago	Interface update done, total dis
StandardPricing.java	7 days ago	Interface update done, total dis
Store.java	7 days ago	Added singleton design pattern
Ui_AddItem.java	7 days ago	Facade Design Pattern Added
Ui_CashDialog.java	a month ago	Add item dialog added with scr
Ui_CheckDialog.java	a month ago	Add item dialog added with scr
Ui_CreditDialog.java	a month ago	Add item dialog added with scr
Ui_NewGenPOS.java	7 days ago	Interface update done, total dis
Ui_ReceiptDialog.java	a month ago	Cash Payment is done, fully w
guiWidgetHandler.java	7 days ago	Interface update done, total dis
verifyCash.java	8 days ago	Adapter Design Pattern done,,
verifyCheck.java	7 days ago	Fixed verifyCheck so that all re
verifyCredit.java	8 days ago	Adapter Design Pattern done,,
verifyPaymentAdapter.java	8 days ago	Adapter Design Pattern done,,

Implementation

Test Case

MoneyTest.java (portion of the code)

```
public class MoneyTest {
   /**
   * Test of calcTotal method, of class Money.
   */
   @Test
   public void test1CalcTotal() {
       System.out.println("Testing Method: Money.calcTotal");
       double tax = 1.08;
       Money subTotal = new Money(9.99);
       //The total should be $10.79
       Money expResult = new Money((9.99*1.08));
       Money result = subTotal.calcTotal(tax);
       System.out.println("Result was: "+result.getFormatted());
       System.out.println("Expected Result is: "+expResult.getFormatted());
       assert(result.checkEquals(expResult));
       System.out.println("Test 1 of Money.calcTotal passed!");
   }
}
```

SaleTest.java (portion of the code)

```
public class SaleTest {
   // initialize sale

   public SaleTest() {
   }

   @Before
   public void setUp() {
       this.testProductItemID = new ItemID(111111);
       this.testProductPrice = new Money(9.99);
       this.testProductStock = 99;
       this.testProductDesc = new ProductDescription(this.testProductItemID,
       this.testProductName, this.testProductPrice, this.testProductStock);
   }

   @After
   public void tearDown() {
   }

   /**
   * Test of calcSubTotal method, of class Sale.
   */
   @Test
   public void test1CalcSubTotal() {
       System.out.println("Testing Method: Sale.calcSubTotal");
       int qty = 5;
       SalesLineItem item = new SalesLineItem(this.testProductDesc,qty);
       Sale instance = new Sale();
       instance.calcSubTotal(item, qty);
       //Subtotal should be $49.95
       Money expectedTotal = new Money(49.95);
       assert(instance.getSubTotal().checkEquals(expectedTotal));
       System.out.println("Test 1 of Sale.calcSubTotal passed!");

   }
}
```

Demo