CPSC 462 - Software Design

Instructor: Christopher T. Ryu, Ph.D., Professor

Homework #2

Submission Date : 10.17.12

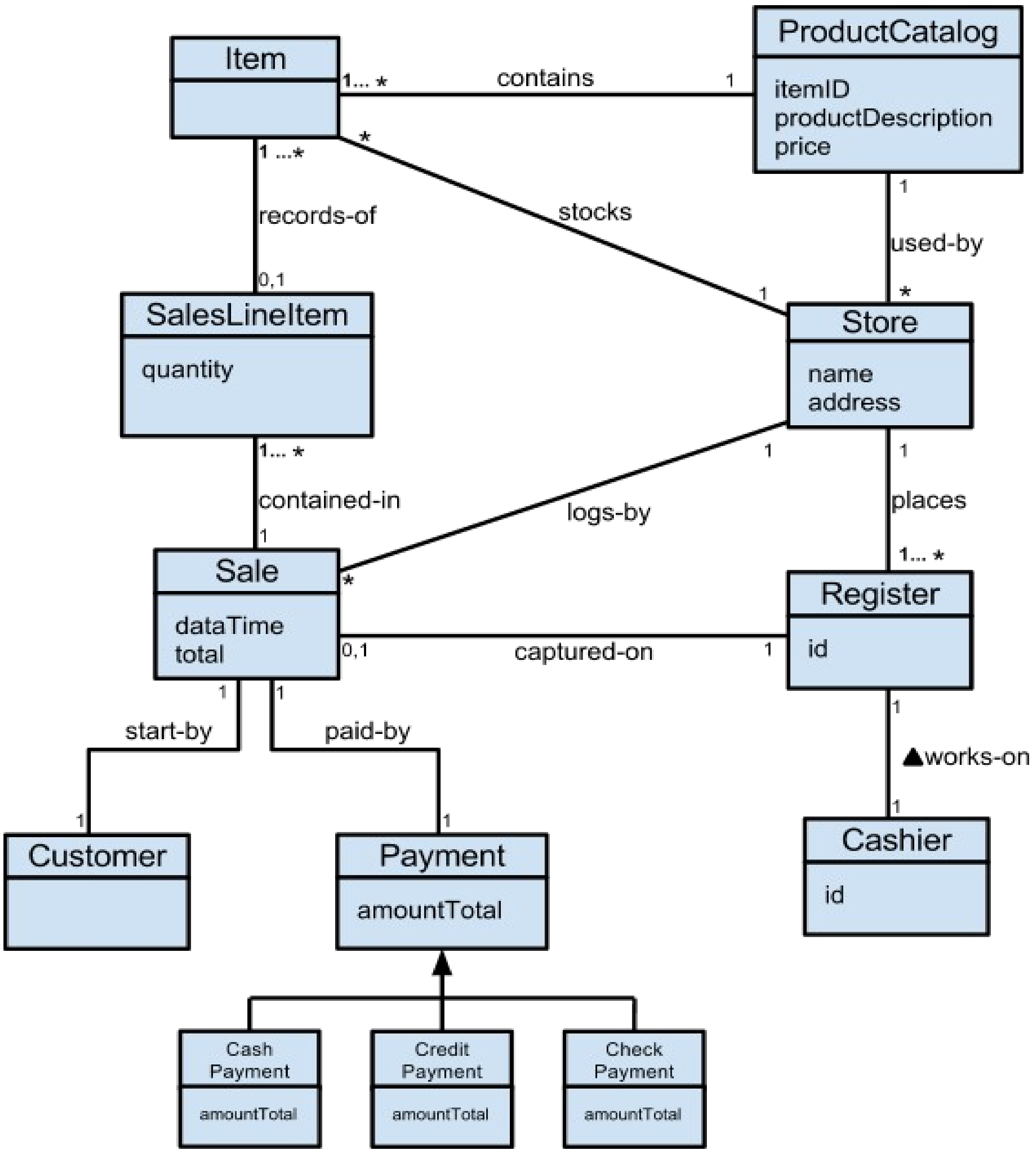
.

Team: NewGen

Name: Michael Robertson	**Email:** mirob2005@gmail.com
Name: Xinchao Liu	**Email:** lookfor@188.com
Name: Wee Siang Wong	**Email:** willydk@gmail.com
Name: Bryan Tamada	**Email:** btamada@gmail.com
Name: Wei Jen Lin	**Email:** littlemike44@hotmail.com

1. Domain Model

Here’s the domain model for use-case “Process Sale”:



2. Candidate Domain Objects in Three Categories

There are 9 candidate domain objects in the Domain Model.

Group them into three categories shown below:

a. Application Category:

- Sale
- Sales Line Item
- Payment

b. Business/Domain Category:

- Customer
- Cashier

c. Architecture Category:

- Store
- Product Catalog
- Item
- Register

3. Six Essential Characteristics for Major Class

Register Class

Retained Information:
This class and its attributes are vital to the function of the system because it handles information on the products and sales transactions. Its operations are the very core of the whole system as it is responsible for the creation and ending of sales transactions, payment and acquiring product descriptions. As a result it will be useful during the analysis phase because of the attributes and functions has.

Multiple Attributes:
The Register class provides multiple attributes such as ProductCatalog and Sale emphasizing the focus on major information. We can be assured that these attributes fit well with this class and should not be distributed or transferred to other classes because this classes expresses the core concept of our system.

Common Attributes:
We can clearly define the attributes for this class and they apply to all instances within its class hierarchy.

Needed Service:
All of the operations provided in this class manipulates the attributes in their own way. For example, endSale() and makePayment() both invoke methods of the Sale attribute, makeNewSale() creates a new instance of Sale, and enterItem() retrieves a product's description and invokes a method of Sale.

Common Operations:
A set of operations have been clearly defined for the Register class and apply to all of its instances.

Essential requirements:
Provides ease of use for external entities, such as the cashier in this case, to produce and/or consume information using this class to start and complete the sales transaction and provide the customer with a receipt listing product descriptions and quantities purchased.

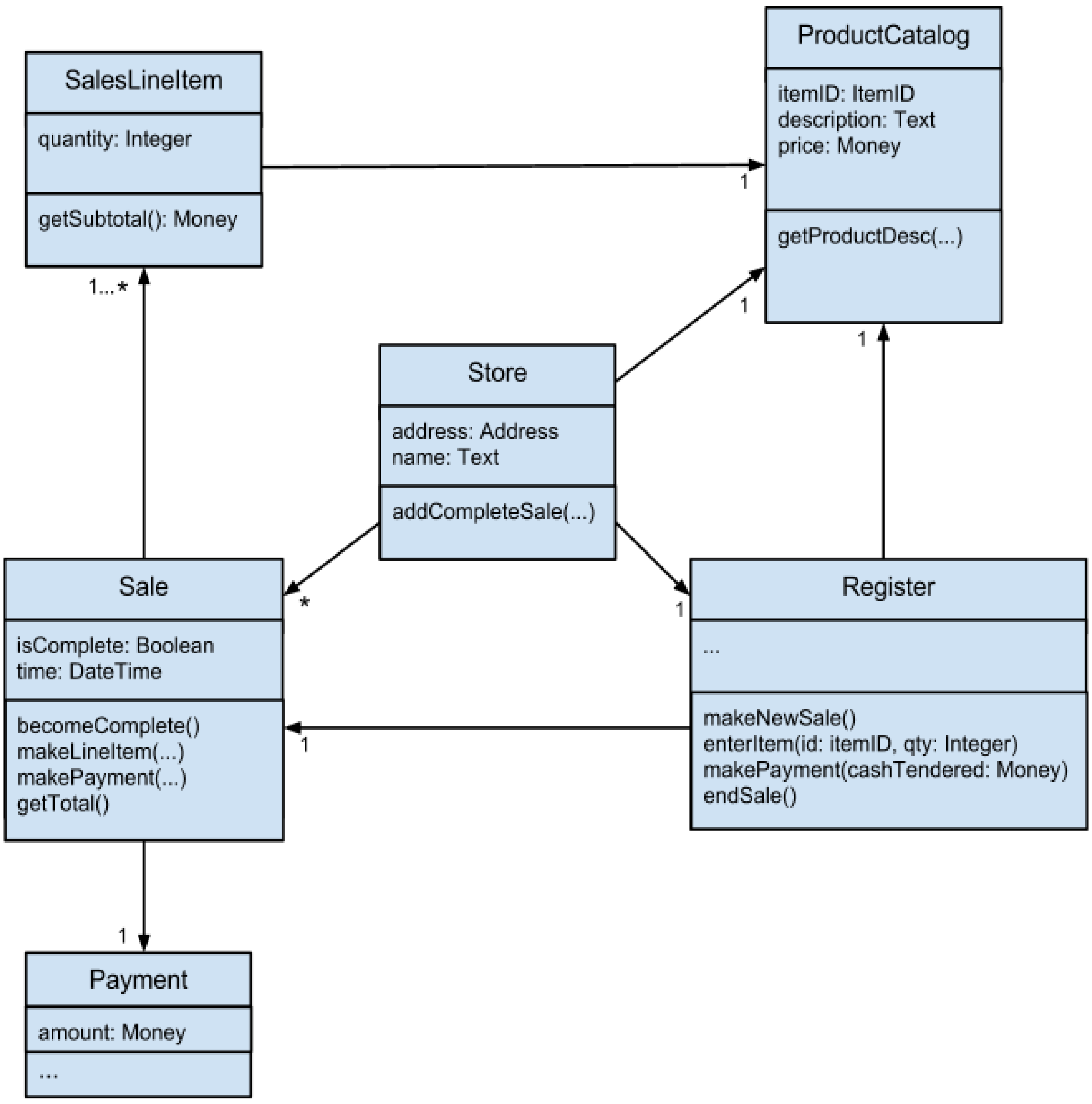
4. Class Hierarchy Conformance Requirements

Payment Class Hierarchy

The subclasses Cash Payment, Credit Payment and Check Payment are mutually overlapping with the Payment class as they all deal with the transaction of a payment by the customer. This is to say that we can group these 3 classes as members (or subclasses) of the Payment class, which we will label as the superclass since it is the more general of the classes, to identify their general relationship. Moreover, the Domain Model satisfies the IS-A rule by illustrating the superclass and subclass' set membership conformance due to this membership and how each member has a unique total amount to display, which was inherited from the Payment class. The subclasses represent variations of similar concepts, have similar attributes, in this case the amountTotal attribute, and associations that can be factored out and expressed in the superclass. The subclasses' concepts are similar in that they are responsible for distinguishing and processing different forms of payments chosen by the customer. Furthermore, their association with the superclass is how they are handled differently; compared to Cash Payment, Check Payment and Credit Payment need to be authorized as a precondition before the sale is complete. The 100% rule is satisfied because the superclass' attributes, operations, and associations are available to the subclasses as we can see they are defined methods in the Domain Model.

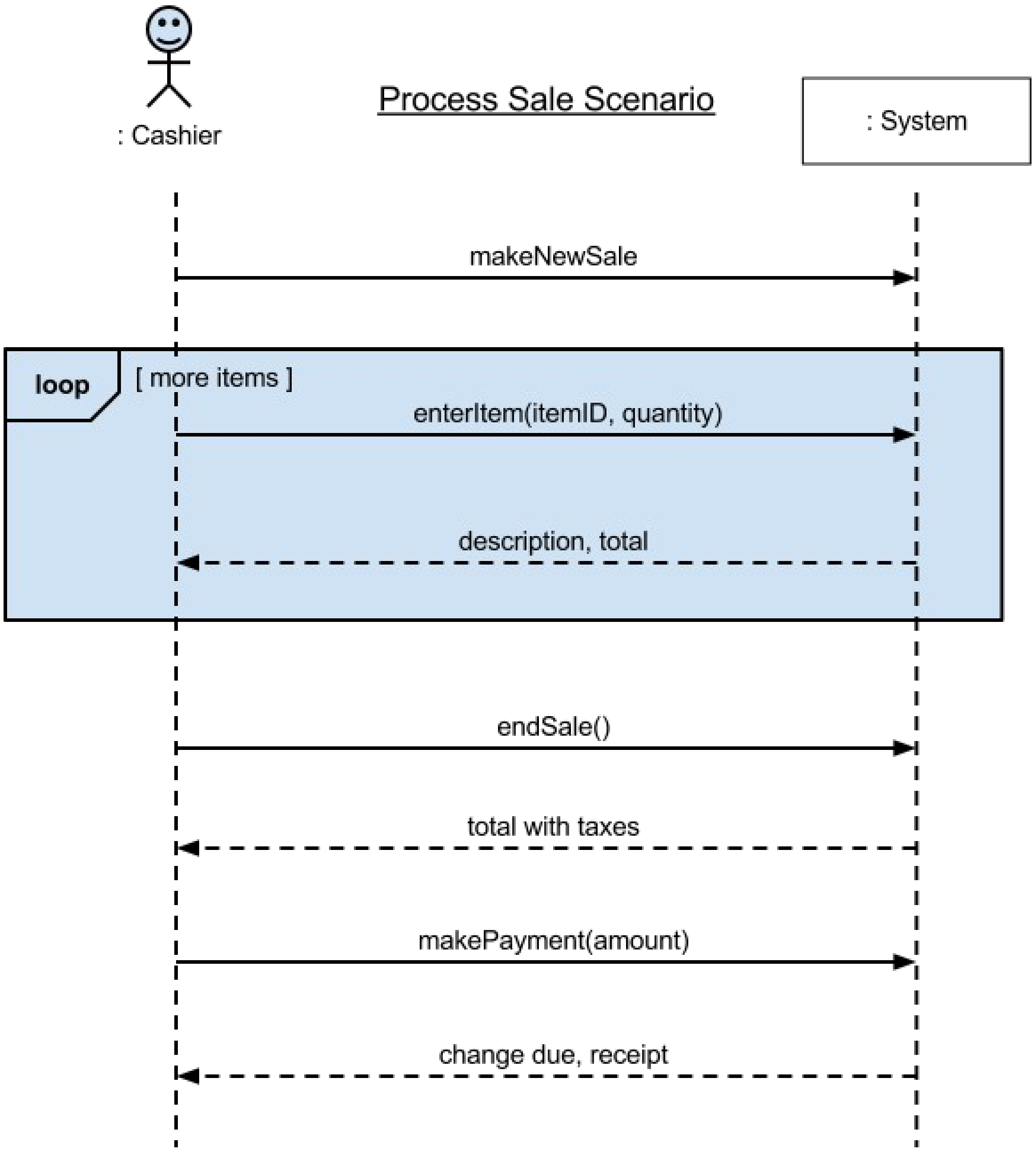
5. ER Diagram for Process Sale Scenario

This is the ER Diagram(using Class Diagram) for Process Sale Scenario:



6. System Sequence Diagram

This is the SSD for Process Sale Scenario:



7. Operation Contract Defined

Operation: submitPayment()

Cross reference: Process Sale

Preconditions: Cashier has input all items checked out by the Customer. The point of sale system has correctly calculated the total price with tax. Cashier informs the Customer of the total price and the Customer decides on a payment type, such as cash, credit, or check payment.

Postconditions: Changes were received by the customer if there were any. The authorization request was submitted by the system if the customer pays with credit or check. Sale was logged. Receipt was generated by the POS system and Cashier hands it to the customer. Customer leaves with the receipt and items purchased.

8. Two Defined Classes by JAVA

First class:

```
public class Register {
    private ProductCatalog catalog;
    private Sale currentSale;

    public Register(ProductCatalog catalog)
    {
        this.catalog = catalog;
    }
    public void endSale()
    {
        currentSale.becomeComplete();
    }
    public void enterItem( ItemID id, int quantity)
    {
        ProductDescription desc = catalog.getProductDescription(id);
        currentSale.makeLineItem(desc,quantity);
    }
    public void makeNewSale()
    {
        currentSale = new Sale();
    }
    public void makePayment(Money cashTendered)
    {
        currentSale.makePayment(cashTendered);
    }
}
```

Second Class:

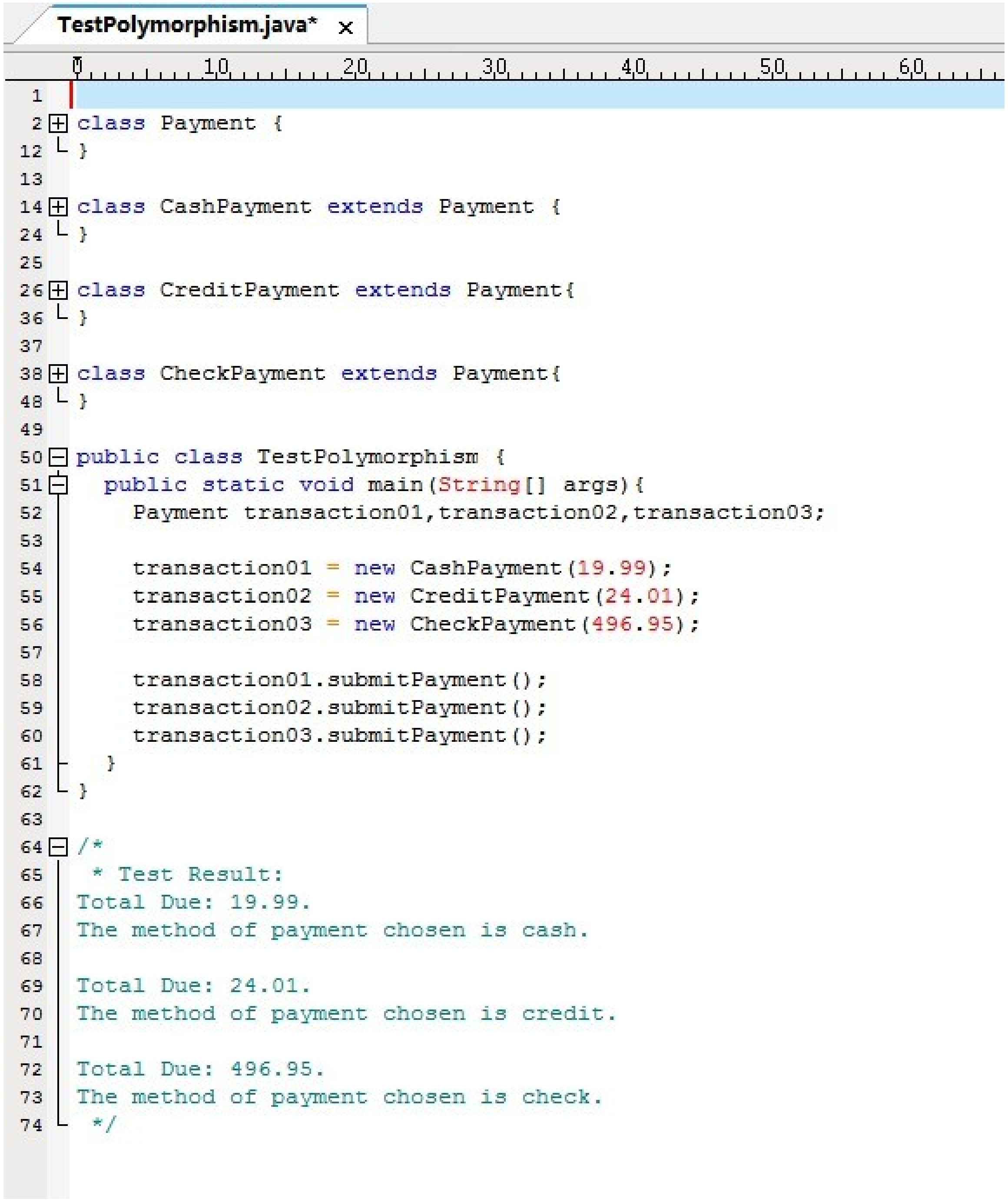
```
public class ProductDescription{
   private ItemID id;
   private Money price;
   private String description;
   public ProductDescription(ItemID id, Money price, String description)
   {
      this.id = id;
      this.price = price;
      this.description = description;
   }
   public ItemID getITemID()
   {
      return id;
   }
   public Money getPrice()
   {
      return price;
   }
   public String getDescription()
   {
      return description;
   }
}
```

Operation contract for enterItem method:

Operation: enterItem(itemID:integer, quantity: integer)
Cross reference: Process Sale
Precondition: There is a sale underway, cashier ready, customer ready to checkout items
Postcondition: A SalesLineItem instance was created. Item description was displayed on the system screen. Item added to cart. Total price with tax was calculated.

9. Polymorphic Operation

Payment class has been identified as a polymorphic operation. Here's the test result:

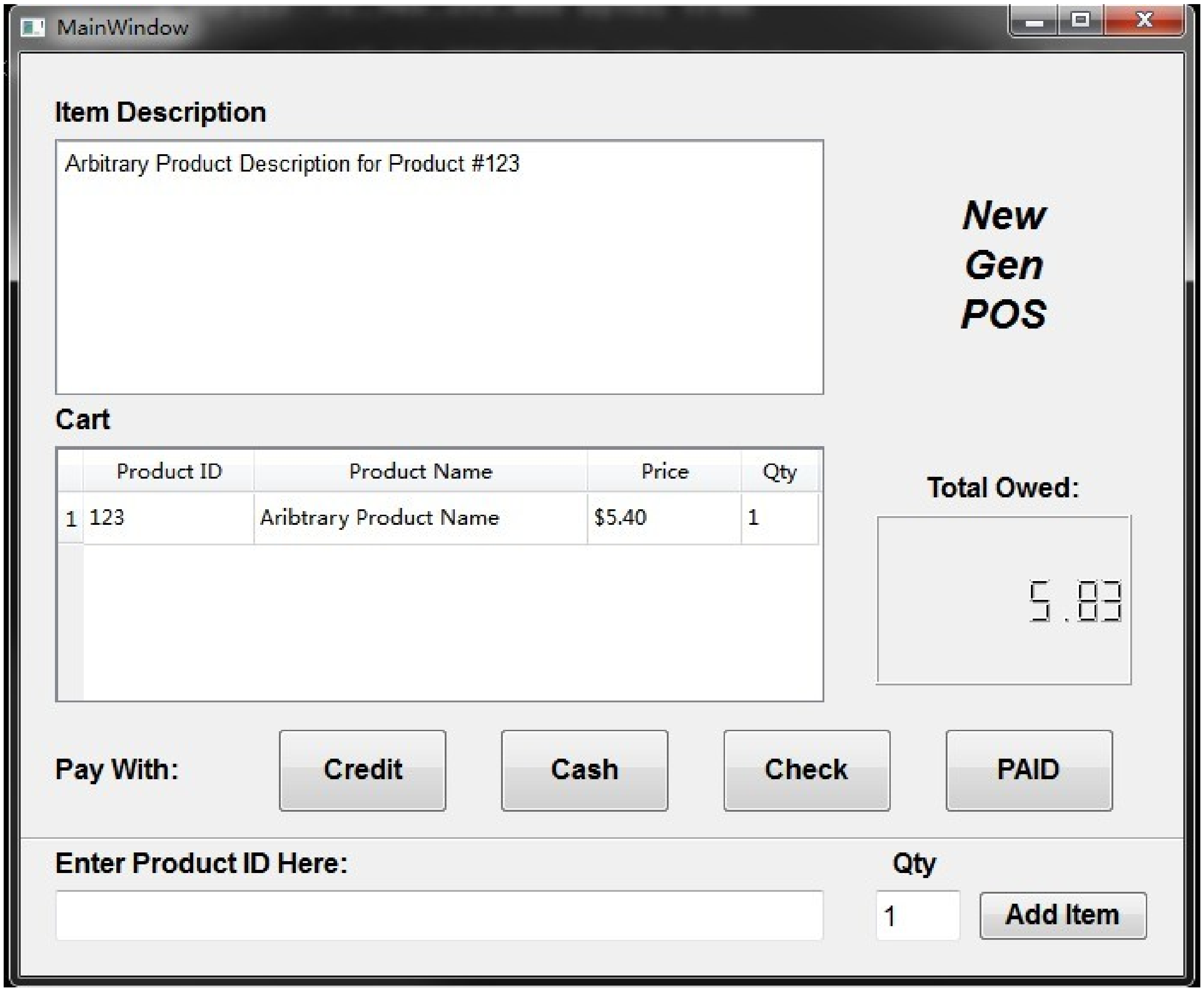


```

class Payment {
}

class CashPayment extends Payment {
}

class CreditPayment extends Payment{
}

class CheckPayment extends Payment{
}

public class TestPolymorphism {
  public static void main(String[] args){
    Payment transaction01,transaction02,transaction03;

    transaction01 = new CashPayment(19.99);
    transaction02 = new CreditPayment(24.01);
    transaction03 = new CheckPayment(496.95);

    transaction01.submitPayment();
    transaction02.submitPayment();
    transaction03.submitPayment();
  }
}

/*
 * Test Result:
Total Due: 19.99.
The method of payment chosen is cash.

Total Due: 24.01.
The method of payment chosen is credit.

Total Due: 496.95.
The method of payment chosen is check.
 */
```

Full Code can be seen at: https://github.com/miro2005/CS462-Project/blob/master/TestPolymorphism/src/TestPolymorphism.java

10. Create User Interface

User Interface was created with Java and Qt.



Arbitrary Product Description for Product #123

	Product ID	Product Name	Price	Qty
1	123	Aribtrary Product Name	$5.40	1

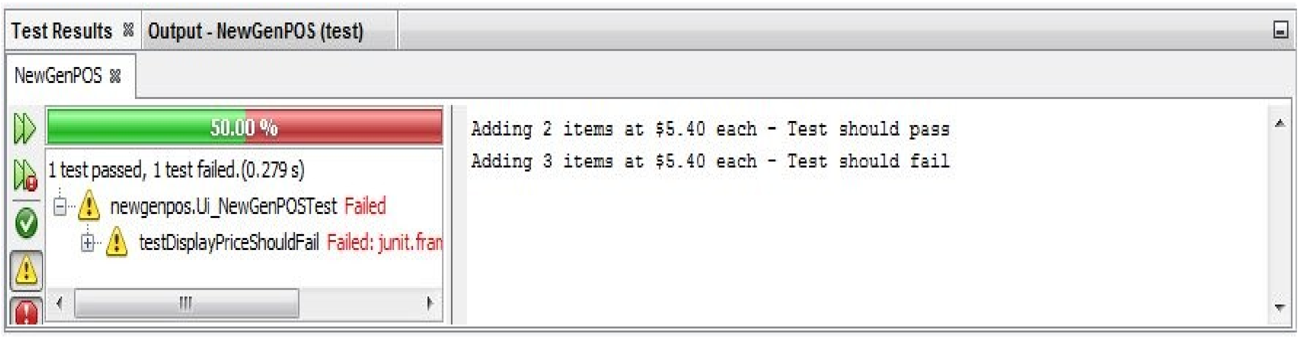
5.83

1

Full code can be seen at: https://github.com/miro2005/CS462-Project/blob/master/NewGenPOS/src/newgenpos/Ui_NewGenPOS.java

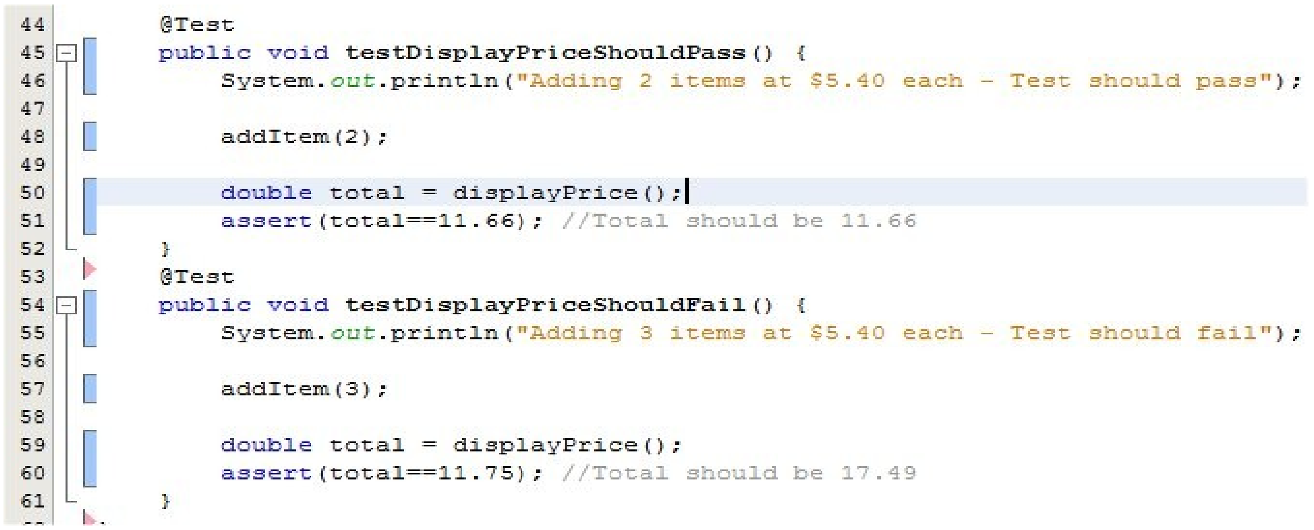
11. Test Cases for User Interface

Full code for the test cases can be seen at: https://github.com/miro2005/CS462-Project/blob/master/NewGenPOS/test/newgenpos/Ui_NewGenPOSTest.java



1 test passed, 1 test failed.(0.279 s)
newgenpos.Ui_NewGenPOSTest Failed
testDisplayPriceShouldFail Failed: junit.fran

```
Adding 2 items at $5.40 each - Test should pass
Adding 3 items at $5.40 each - Test should fail
```



```
    @Test
    public void testDisplayPriceShouldPass() {
        System.out.println("Adding 2 items at $5.40 each - Test should pass");

        addItem(2);

        double total = displayPrice();
        assert(total==11.66); //Total should be 11.66
    }
    @Test
    public void testDisplayPriceShouldFail() {
        System.out.println("Adding 3 items at $5.40 each - Test should fail");

        addItem(3);

        double total = displayPrice();
        assert(total==11.75); //Total should be 17.49
    }
```