# Homework 4: Java to UML Exercise

# Objective

This exercise enables the student to be able to map between simple Java programs and UML class and sequence diagrams. It provides Java source code and asks the students to create class and sequence diagrams that correspond to the code..

Note: this application and code are based on Craig Larman’s *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall, 1998 (1st Edition) and 2004 (3rd Edition).

# Application

Consider a “point of sale terminal,” which is a cash register in a store that is connected to a database of product information that includes product price, description, and available quantity. During a sale, a clerk will enter into the register the product identity information (a universal product code, available from a bar-code scanner, for example) and quantity for each item sold. Product description and price information will be available to the clerk at the register. After all items are entered, the register will provide the sale total and accept notification that payment has been provided. The register will then reduce the quantity of sold products from the product information in the database.

Consider that you are joining a project team developing the Point of Sale Terminal (POST) product, and you are given the following Java code, which is part of an early increment toward building the product. You note that it is poorly documented. To better understand what the code does, you decide to create UML class and sequence diagrams from the code. You create class diagrams from the class declarations, and you create sequence diagrams by tracing the object interactions that implement key methods.

# Task

1. Create a UML class diagram that covers all the attributes, operations, and obvious relationships between the classes.
2. Create a UML sequence diagram for starting up the store
   1. Invoke the Store() constructor and trace it to completion, collaborating with other objects.
3. Create a UML sequence diagram for entering an item into a sale
   1. Invoke the Register.enterItem() method and trace it to completion, collaborating with other objects.
4. Create a UML sequence diagram for making a payment after all sale items have been entered into the register
   1. Invoke the Register.makePayment() method and trace it to completion, collaborating with other objects.
   2. Assume that the sale is complete and that the amount of money tendered is sufficient to cover the total cost.

# Java Code

## Class Store

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* @maintainer Scott Hawker

\*/

**public** **class** Store {

**private** ProductCatalog productCatalog;

**private** Register register;

**public** Store() {

productCatalog = **new** ProductCatalog();

register = **new** Register( productCatalog );

}

**public** Register getRegister() {

**return** register;

}

**public** **void** addSale() {

// To Be Defined

}

}

## Class Register

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* @maintainer Scott Hawker

\*/

**import** java.util.\*;

**public** **class** Register {

**private** ProductCatalog productCatalog;

**private** Sale sale;

**public** Register( ProductCatalog catalog ) {

productCatalog = catalog;

}

// when a sale becomes complete, the system goes to a

// state where it can accept payment

**public** **void** endSale() {

sale.becomeComplete();

}

**public** **void** enterItem( **int** upc, **int** quantity ) {

**if** ( isNewSale() ) {

sale = **new** Sale();

}

ProductDescription desc =

productCatalog.getDescription( upc );

sale.makeLineItem( desc, quantity );

}

**public** **void** makePayment( **float** cashTendered ) {

sale.makePayment( cashTendered );

sale.getBalance();

}

**private** **boolean** isNewSale() {

**return** ( sale == **null** ) || (sale.isComplete() );

}

}

## Class ProductCatalog

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* @maintainer Scott Hawker

\*/

**import** java.util.\*;

**public** **class** ProductCatalog {

**private** Hashtable productDescriptions;

**private** ProductDescription productDescription;

**public** ProductCatalog() {

productDescriptions = **new** Hashtable();

ProductDescription ps = **new** ProductDescription( 100, 1,

"product 1" );

productDescriptions.put( **new** Integer( 100 ), ps );

ps = **new** ProductDescription( 200, 1, "product 2");

productDescriptions.put( **new** Integer( 200 ), ps );

}

**public** ProductDescription getDescription( **int** upc ) {

**return** (ProductDescription) productDescriptions.get(

**new** Integer( upc ) );

}

}

## Class ProductDescription

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* @maintainer Scott Hawker

\*/

**public** **class** ProductDescription {

**private** **int** upc = 0;

**private** **float** price = 0;

**private** String description = "";

**public** ProductDescription( **int** upc, **float** price,

String description ) {

**this**.upc = upc;

**this**.price = price;

**this**.description = description;

}

**public** **int** getUPC() {

**return** upc;

}

**public** **float** getPrice() {

**return** price;

}

**public** String getDescription() {

**return** description;

}

}

## Class Sale

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* @maintainer Scott Hawker

\*/

**import** java.util.\*;

**public** **class** Sale {

**private** Vector lineItems;

**private** Date date;

**private** **boolean** isComplete;

**private** Payment payment;

**public** Sale() {

lineItems = **new** Vector();

date = **new** Date();

isComplete = **false**;

}

**public** **float** getBalance() {

**return** payment.getAmount() - total();

}

**public** **void** becomeComplete() {

isComplete = **true**;

}

**public** **boolean** isComplete() {

**return** isComplete;

}

**public** **void** makeLineItem( ProductDescription spec, **int** quantity )

{

lineItems.addElement( **new** SaleLineItem( spec, quantity) );

}

**private** **float** total() {

**float** total = 0;

Enumeration e = lineItems.elements();

**while** (e.hasMoreElements() ) {

total += ((SaleLineItem) e.nextElement()).subtotal();

}

**return** total;

}

**public** **void** makePayment( **float** cashTendered ) {

payment = **new** Payment( cashTendered );

}

}

## Class SaleLineItem

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* **@author** Scott Hawker

\*/

**public** **class** SaleLineItem {

**private** **int** quantity;

**private** ProductDescription productDesc;

**public** SaleLineItem( ProductDescription desc, **int** quantity) {

**this**.productDesc = desc;

**this**.quantity = quantity;

}

**public** **float** subtotal() {

**return** quantity \* productDesc.getPrice();

}

}

## Class Payment

/\*

\* Created on Apr 3, 2006

\* Modified on March 27, 2007

\*

\* Based on Chapter 24 of Craig Larman's Applying UML and Patterns,

\* 1st Edition, 1998.

\*/

**package** post;

/\*\*

\* **@author** Craig Larman

\* @maintainer Scott Hawker

\*

\*/

**public** **class** Payment {

**private** **float** amount;

**public** Payment( **float** cashTendered ) {

**this**.amount = cashTendered;

}

**public** **float** getAmount() {

**return** amount;

}

}