Pepsi Machine UML

Java Programming - 152-116 – John Heckendorf

Keith Gallistel & Jan Young
5/5/2009

Descriptive Abstract

This paper provides a unified modeling language (UML) software development model of a Pepsi Machine. It contains all the UML components to create a Pepsi Machine program. This project can be used as a training resource.

The UML process presented in this paper assumes developers will utilize an agile object oriented development process. As a result, each use case in this UML project can be developed independently and simultaneously. This process will allow the system to be implemented in a reasonably short time frame. The initial artifacts of this UML model are use cases. All subsequent artifacts are directly attached to a specific use case. This documentation methodology facilitates the agile object oriented design process and provides a logical design flow.

MVC Reports

The MVC paradigm
By Jan Young

Model-View-Controller (MVC) programming is the application of three-way factoring. Objects of different classes take over the operations related to the application domain (the model), the display of the application's state, the view, and the user interaction with the model and the view, the controller. It is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller. The MVC, control flow is as follows: the user interacts with the user interface in some way, the controller handles the input event from the user interface, often via a registered handler or callback the controller notifies the model of the user action, possibly resulting in a change in the model's state, a view uses the model indirectly to generate an appropriate user interface the view gets its own data from the model, the model and controller have no direct knowledge of the view, and the user interface waits for further user interactions, which restarts the cycle.

The “View” renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. It is responsible for mapping graphics onto a device. A viewport typically has a one to one correspondence with a display surface and knows how to render to it. It attaches to a model and renders its contents to the display surface. There can be multiple viewports on the same model and each of these viewports can render the contents of the model to a different display surface.

A controller is the means by which the user interacts with the application. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input. The controller is responsible for mapping end-user action to application response. Java Swing doesn't use a single controller. It is common to create an anonymous action class for each event. The real controller is in a separate thread, the event dispatching thread. It catches and propagates the events to the view and model.

The model is used to manage information and notify observers when that information changes. It contains only data and functionality that are related by a common purpose. It is a collection of Java classes that forms a software application intended to store, and move, data. There is a single front end class that can communicate with any user interface.

Extreme Programming (XP) is actually a deliberate and disciplined approach to software development. XP is successful because it stresses customer satisfaction. The methodology is designed to deliver the software your customer needs when it is needed. This methodology also emphasizes team work. Managers, customers, and developers are all part of a team dedicated to delivering quality software. XP implements a simple, yet effective way to enable groupware style development.

MVC Written Report

By Keith Gallistel

Model-View-Control (MVC) and Extreme Programming are used as methodologies to create software. Each has its own benefits and drawbacks. MVC is well documented while Extreme Programming is more customers oriented in its ability to respond to design changes.

MVC is a pattern in software engineering. It was created by Trygve Reenskaug in 1979 while working on Smalltalk. It became big in the 1990s as a way to design software. In 2002, the W3C accepted it as part of the XForms standard. Its advantage is that it separates the business logic from the interface. Also, MVC reduces design complexity and increases the reuse of code.

The View component is the user interface. It also requests updates and allows the Control to select a view. In Java Swing applications, this is represented by the class that inherits Component.

The Control component is the events and actions. This is where actions are defined. Java Swing doesn't have one controller because each interface has its own events. Java Swing can use an event dispatching thread to handle events.

The Model component is the data and information. It is the aspect of the program that handles the calculation. It encapsulates the application and exposes functionality.

Extreme Programming is another pattern of software engineering. It focuses on customer demand and flexibility. This leads to fewer documents and overall design. Extreme Programming uses Unit Tests to check for bugs in the program. Its drawbacks are fewer documents to handle conflicts and unstable requirements.

Reference:

"Extreme Programming"; Wikipedia; http://en.wikipedia.org/wiki/Extreme_Programming

Jeffries, Ron; "What is Extreme Programming?"; 11/08/2001; XProgramming.com; http://www.xprogramming.com/xpmag/whatisxp.htm

"Model-view-controller"; Wikipedia; http://en.wikipedia.org/wiki/Model-view-controller

"Model-View-Contoller"; Java BluePrints; http://java.sun.com/blueprints/patterns/MVC-detailed.html

UML Reports

Use Cases

By Travis Welsh, Joe Bauer, and Shana Gardner

A use case is a diagram that captures what a system is supposed to do and who will be using it. It is the first and most basic of the seven UML diagrams and shows the interactions between a system's clients and its processes.

The syntax is simple, using stick figures to represent users, ovals to represent processes, and lines to show interactions between the two. For example, if a user has to log in to use a grade book, a use case diagram might have a stick figure labeled "User," and two ovals labeled "Login" and "Grade Book," with lines going from the user to each oval.

Activity Diagram

By Kevilus-Krueger

An activity diagram is a graphical representation of the business and operational workflow of a system.

There are several different elements that make up an activity diagram and some are as follows:

Initial node:

Description: Starting point of a diagram. Not required but makes reading it easier

Symbol: Filled in black circle

Activity Final Node:

Description: Denotes the end of all control flows within the activity. Can have multiple of these symbols

Symbol: Circle with a black circle in it

Activity:

Description: Representation of activities that can occur. These may be physical activities as well.

Symbol: Rounded rectangles

Flow:

Description: Lines representing the directional flow of the system from element to element

Symbol: Lines with arrows at the end pointing in the direction of the flow

Fork:

Description: Denotes beginning of parallel activity

Symbol: Black bar with a single flow going into it and several leaving

Join:

Description: Denotes the end of parallel activity

Symbol: Black bar with multiple flows going into it and one leaving

Decision:

Description: Location where a decision is made based on certain conditions

Symbol: A diamond with options written on either sides of the arrows emerging from the diamond where the options are written within box brackets, ex: [True]

Merge:

Description: Location where more than one incoming flows must reach before processing continues, based on any guards on the outgoing flow

Symbol: A diamond with several flows entering and one leaving

Partition:

Description: Indicates who or what is performing activities

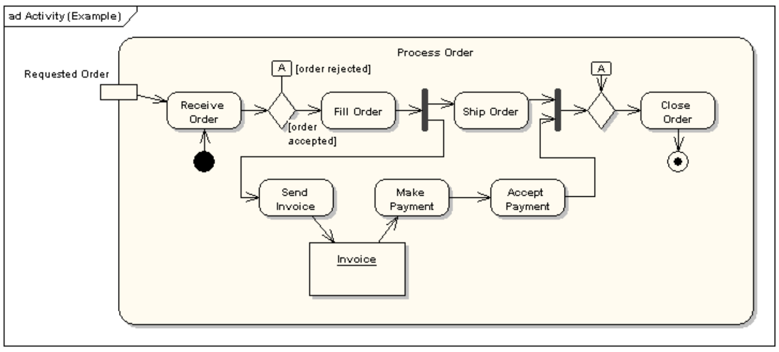
Symbol: Vertical or horizontal lines, or swim lanes, separating the activities

Flow Final:

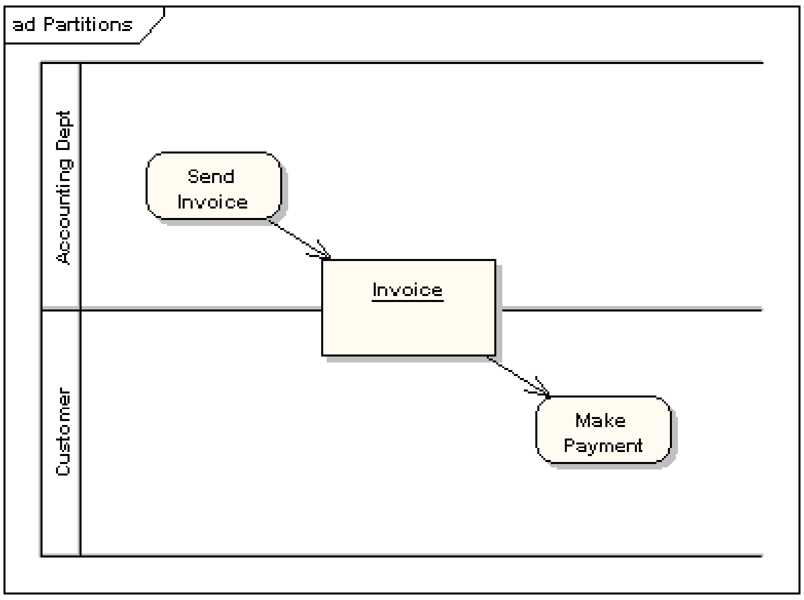
Description: Denotes the end of a single control flow

Symbol: Circle with an X in it

Diagram Example:



Partition/Swim Lanes Example:



Examples found at:

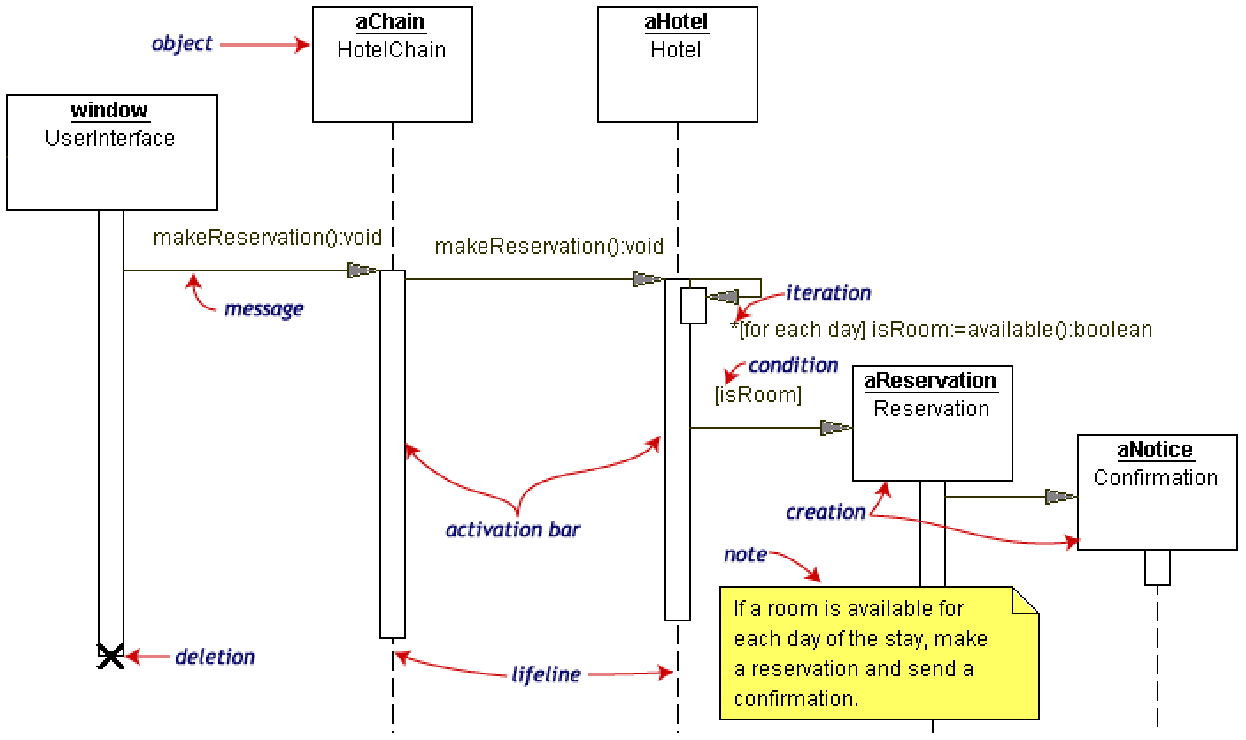
http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_activitydiagram.html

Sequence Diagrams

Catherine Gutowski & Sherry Oakes

By definition, Sequence Diagrams show the interaction between objects in a sequential order, visually depicting the flow of logic in a system. They are called sequence diagrams because of their sequential flow starting from left moving right and top to bottom. Sequencing is a good way to define methods that might be required to execute a process and because of the simplicity, Sequence Diagrams are a good tool to explain the system processes to laymen.

In general, the diagram conveys this information along the horizontal and vertical dimensions: the vertical dimension shows, top down, the time sequence of messages/calls as they occur, and the horizontal dimension shows, left to right, the object instances that the messages are sent to.



Using the example above, the sequencing components are:

1. Objects (Lifeline notation elements) designated as a square at the top with a vertical dashed line. Objects are typically underlined if not a role.

2. Messages are the horizontal arrows between the lifelines with the descriptions written above each arrow. Solid arrows with full heads are synchronous calls; recurring events that happen regularly. The solid arrows with open heads are asynchronous calls; events that happen intermittently. Last, the dashed arrows with stick open heads are return messages.

3. Activation bars are clear rectangles drawn on top of the lifelines to represent that processes are being performed in response to the message.

4. When an object is to be removed from memory (destroyed), an 'X' is drawn on the lifeline and the lifeline terminates.

5. The solid circle designates a message received from outside the diagram.

6. Notes can be designated in a rectangle with a folded corner.

7. Loops (not shown above) can be designated with an open (not filled in) circle or as a frame surrounding the looping messages.

8. More advanced documentation may include Alternatives, Options, Gates, Parallels, Breaks and even References to other diagrams. These depictions are not covered in this paper.

Sequence diagramming can be very inclusive in systems where the number of objects or processes is limited or fixed. However, in more complex systems with a large about of objects and asynchronous messaging, sequence diagramming cannot fully reflect the system's behavior. In these cases, sequence diagramming is used more for developing simple overviews of the system such as a Use Case.

Class Diagrams
By Janelle Young and Keith Gallistel

Definition:

Class diagrams are what allow UML to do modeling. A class diagram is a diagram that shows classes and interfaces. Also it shows collaborations and relationships among those classes and interfaces.

Explanation:

The purpose of the class diagram is to show the types being modeled within the system. There are three "classifiers" within the diagram. These are a class, an interface, and a package.

Class:

A class is a rectangle containing three compartments stacked vertically. The top compartment shows the class's name. The middle compartment lists the class's attributes. The bottom compartment lists the class's operations (methods).

Interface:

An interface is a variation of a class. It provides only a definition of business functionality of a system. A separate class implements the actual business functionality. An interface shares the same features as a class; except that the methods are only declared in the interface and will be implemented by the class implementing the interface.

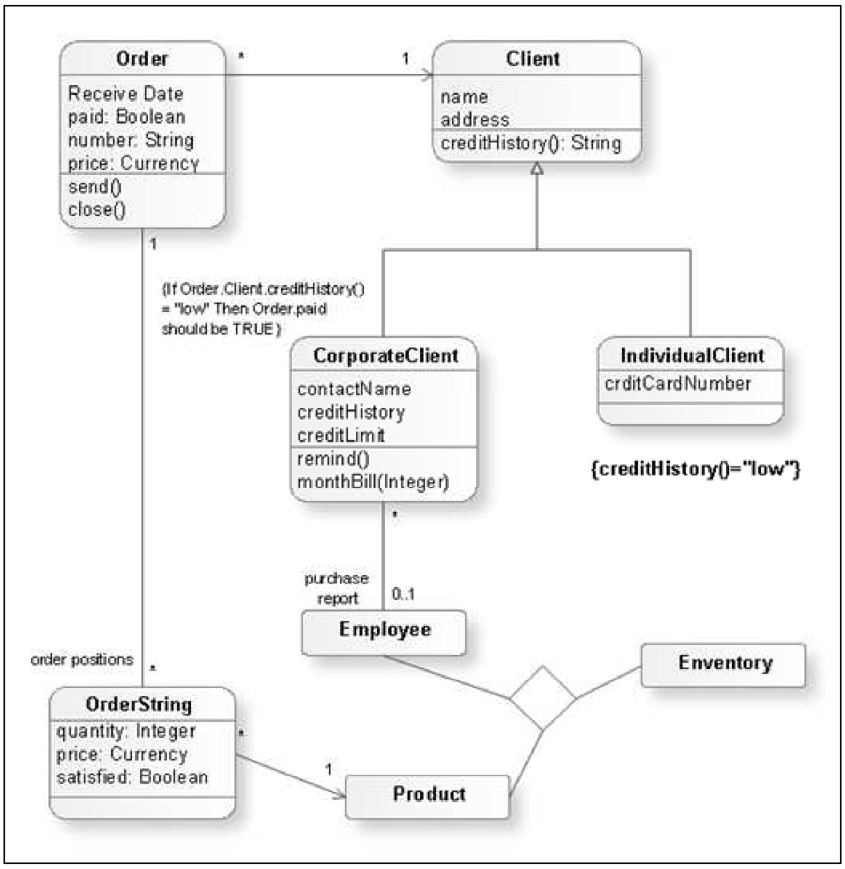
Package:

A package provides the ability to group together classes and/or interfaces that are related.

Grouping these design elements in a package element provides for better readability of class diagrams, especially complex class diagrams.

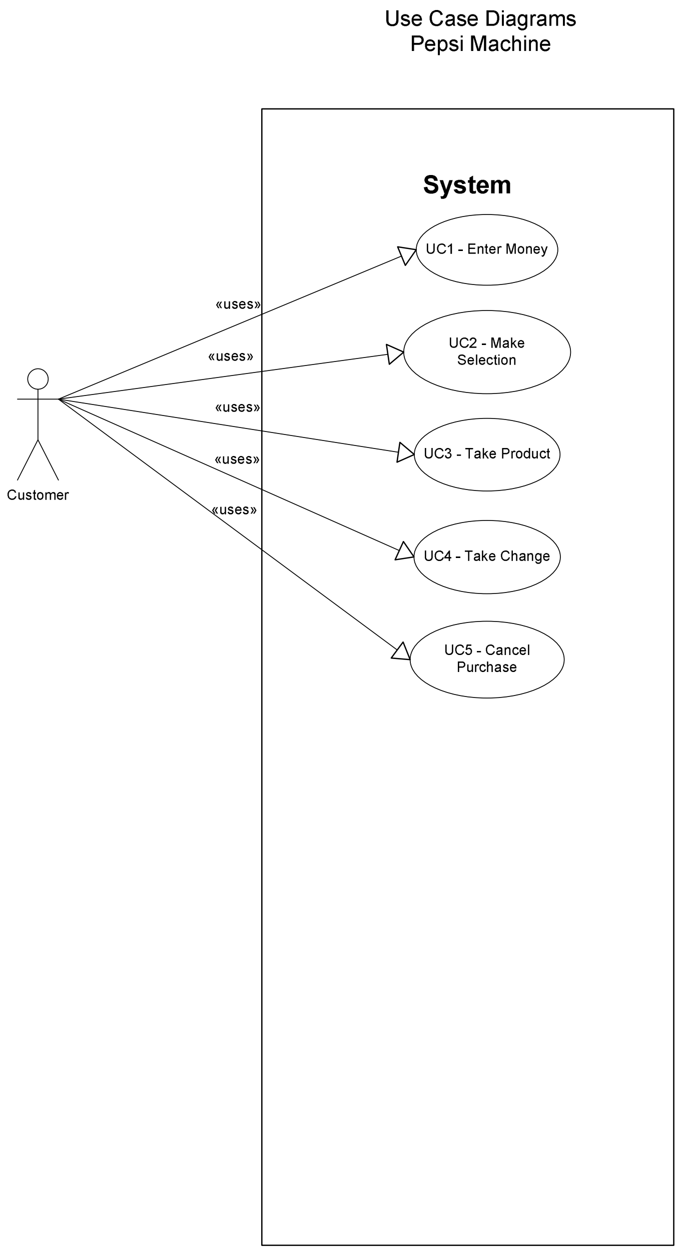
Example:

The following example shows a class diagram. “Order” is an example of a class. “Receive Data” in “Order” is an example of an interface.



Example: From http://www.conceptdraw.com/products/img/ScreenShots/cd5/uml/UML_Class-diagram.gif

Use Case


Use Case Diagrams
Pepsi Machine
System
UC1 - Enter Money
UC2 - Make Selection
UC3 - Take Product
UC4 - Take Change
UC5 - Cancel Purchase
«uses»
«uses»
«uses»
«uses»
«uses»
Customer

Activity Diagrams

Use Case 1 (see Figure 1)

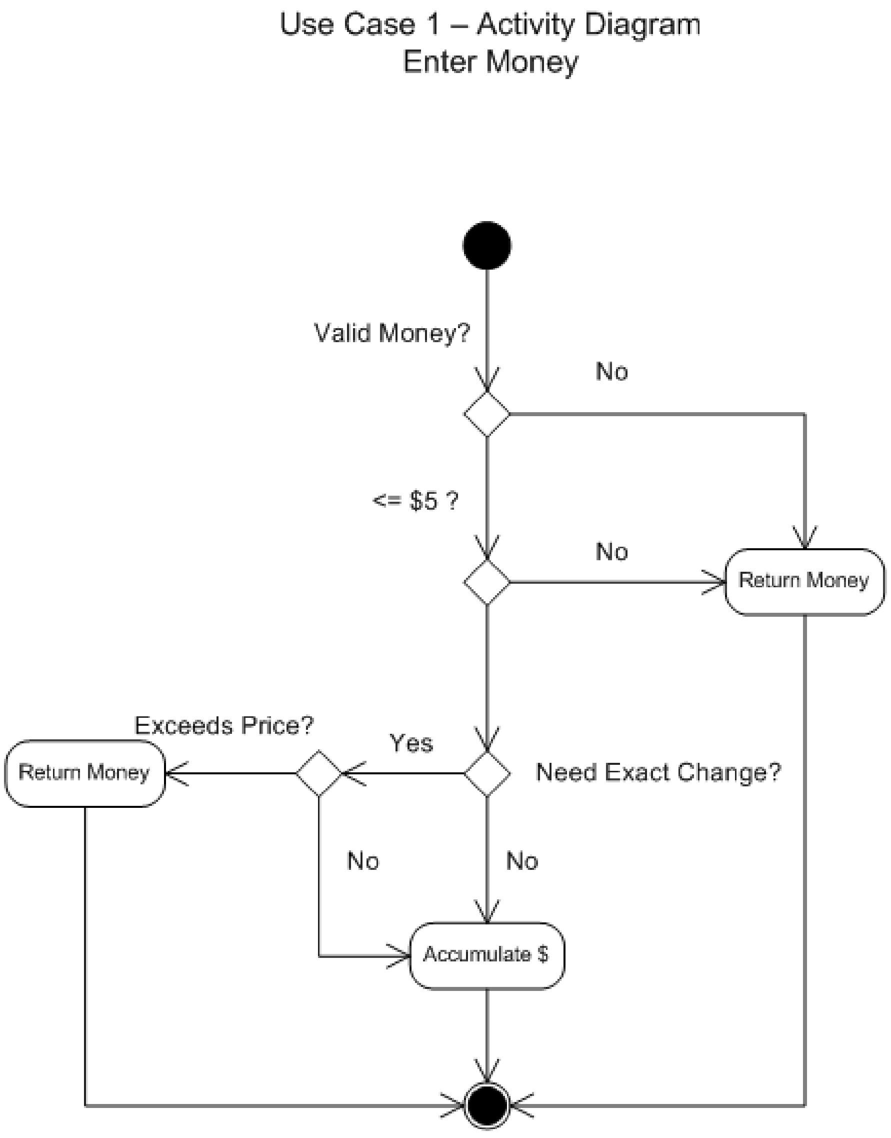


Figure 1

Use Case 2 (see Figure 2)

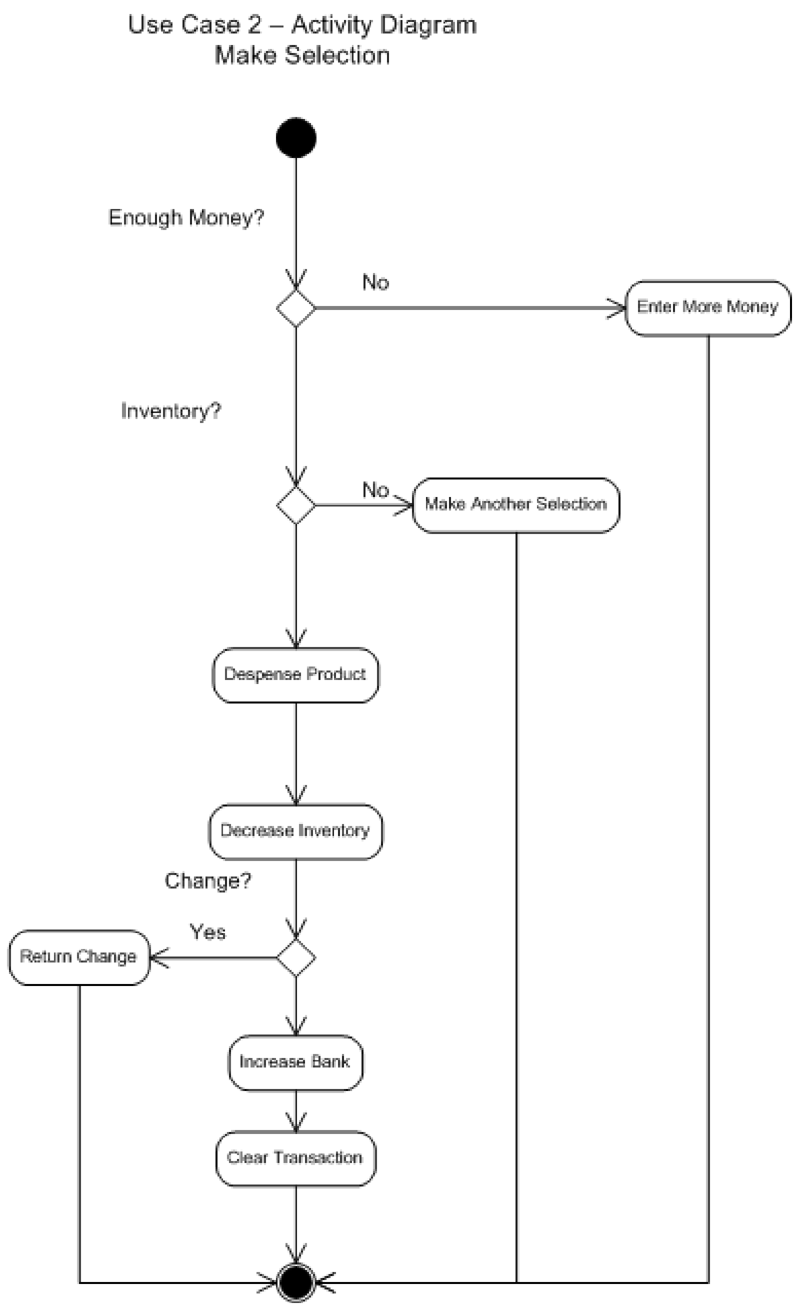


Figure 2

Use Case 3 (see Figure 3)

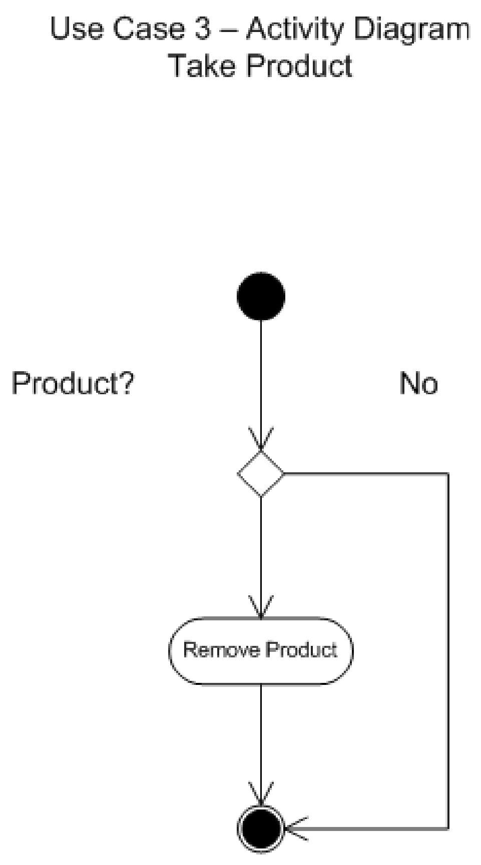


Figure 3

Use Case 4 (see Figure 4)

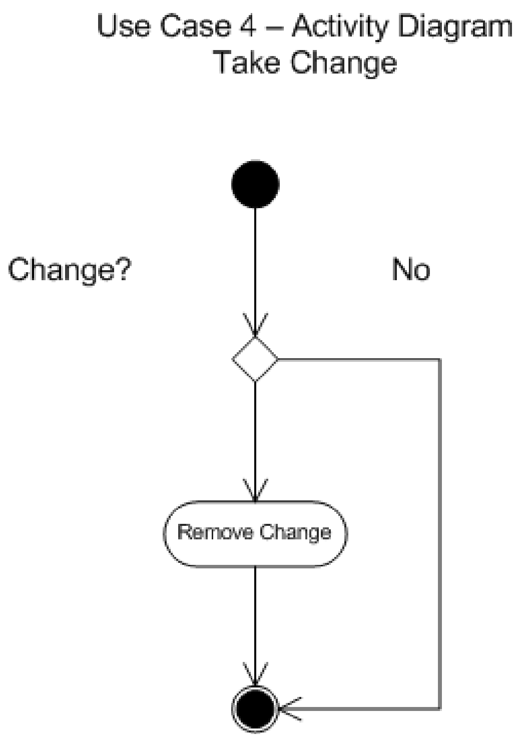


Figure 4

Use Case 5 (see Figure 5)

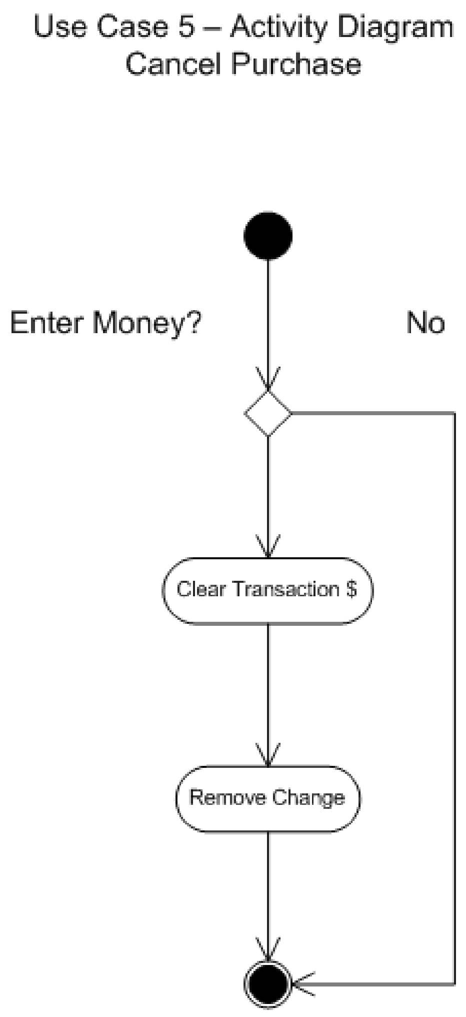


Figure 5

Sequence Diagrams

Use Case 1 (see Figure 6)

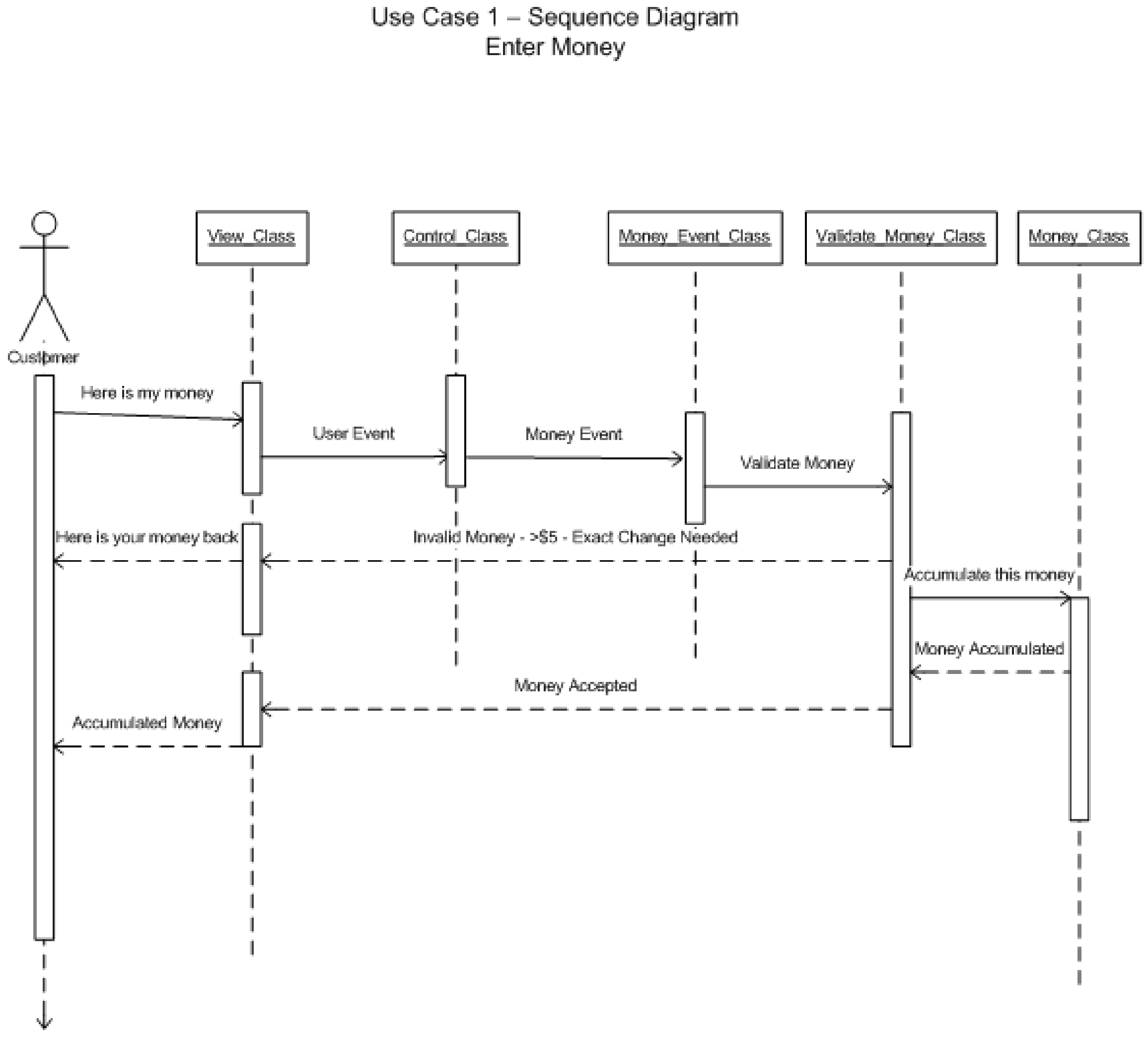


Figure 6

Use Case 2 (see Figure 7)

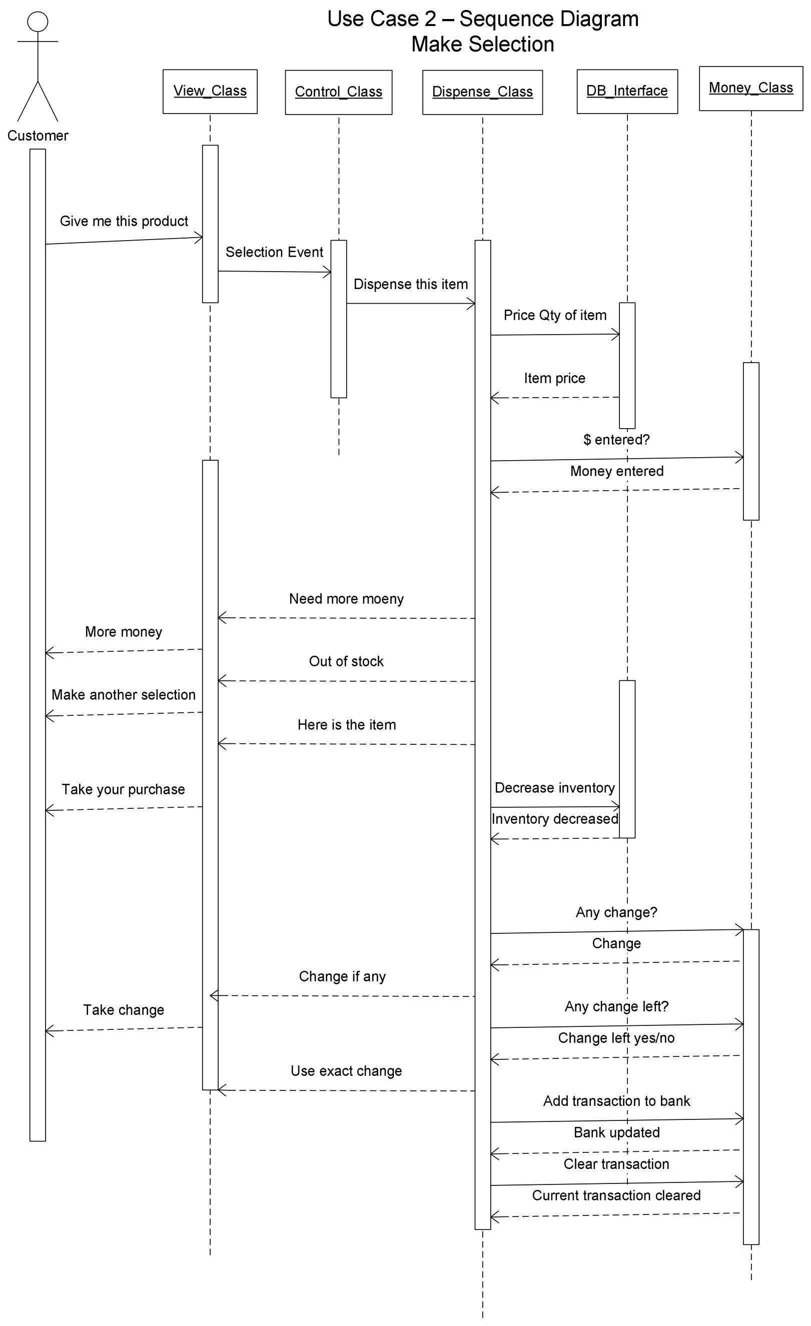


Figure 7

Use Case 3 (see Figure 8)

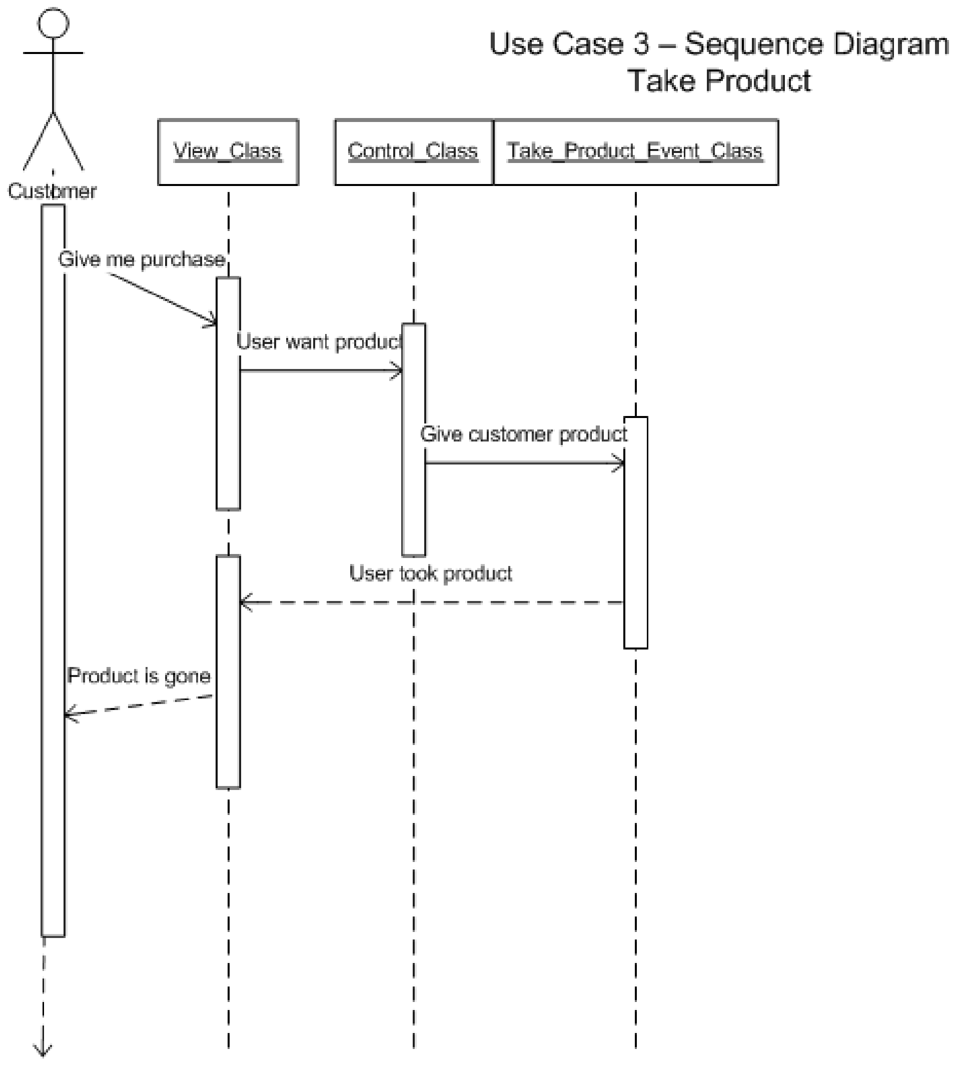


Figure 8

Use Case 4 (see Figure 9)

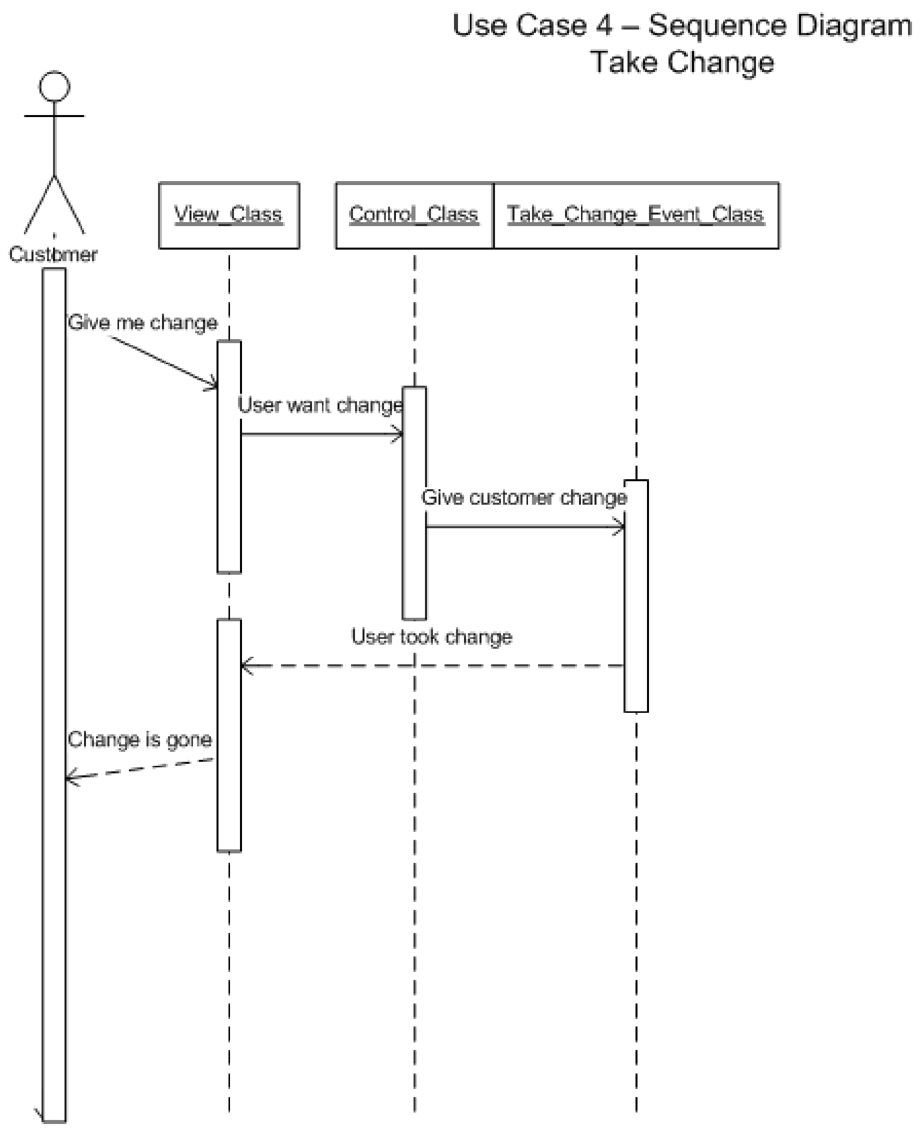


Figure00209

Use Case 5 (see Figure 10)

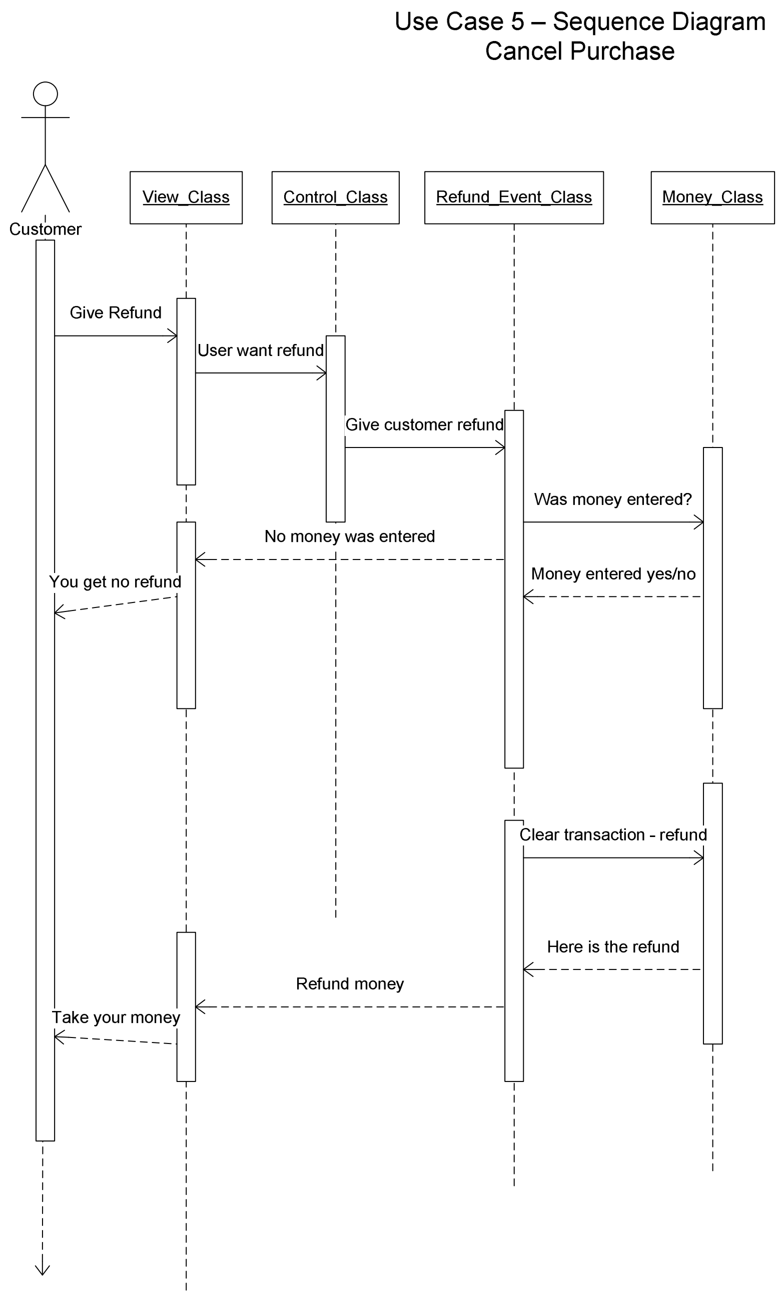


Figure 10

Class Diagram (see Figure 11)

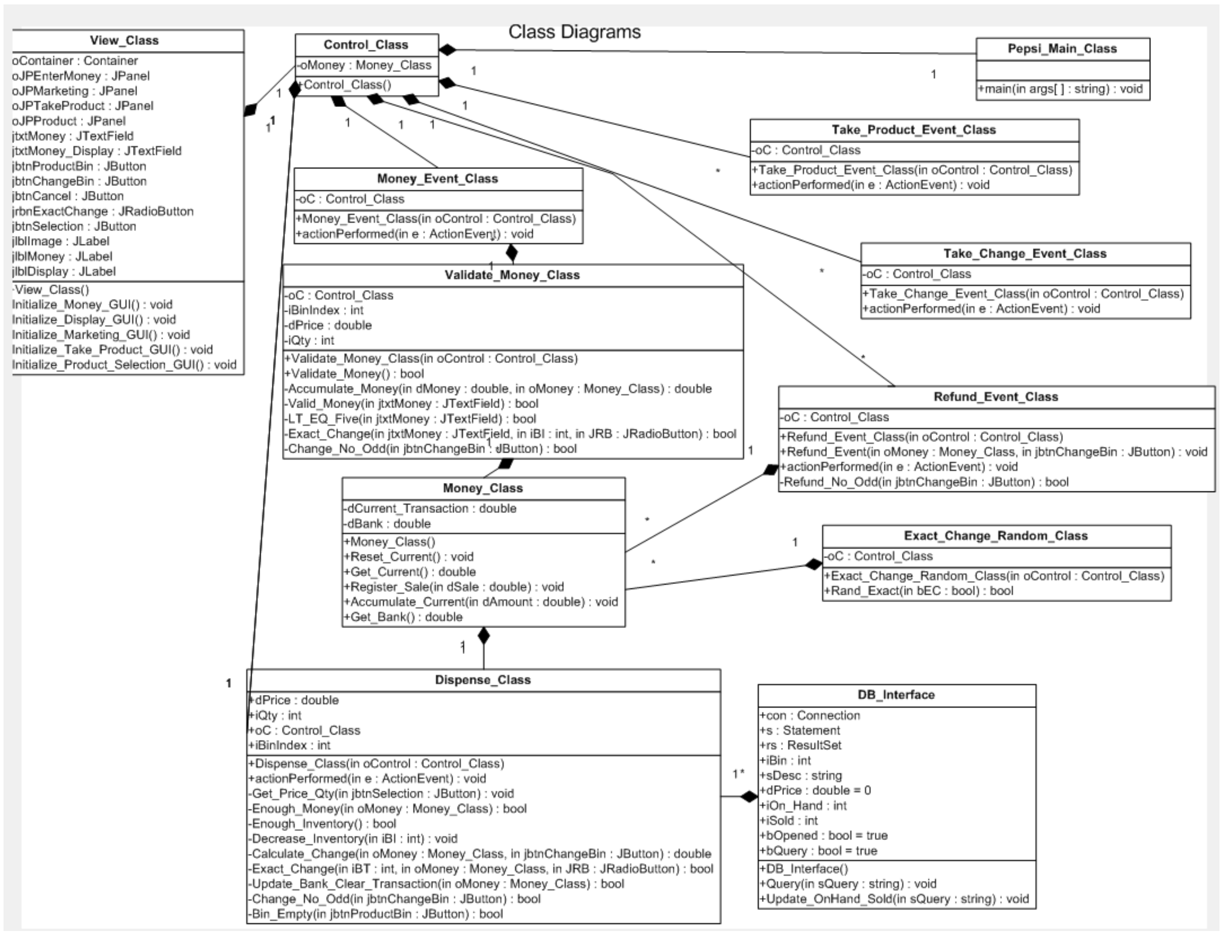
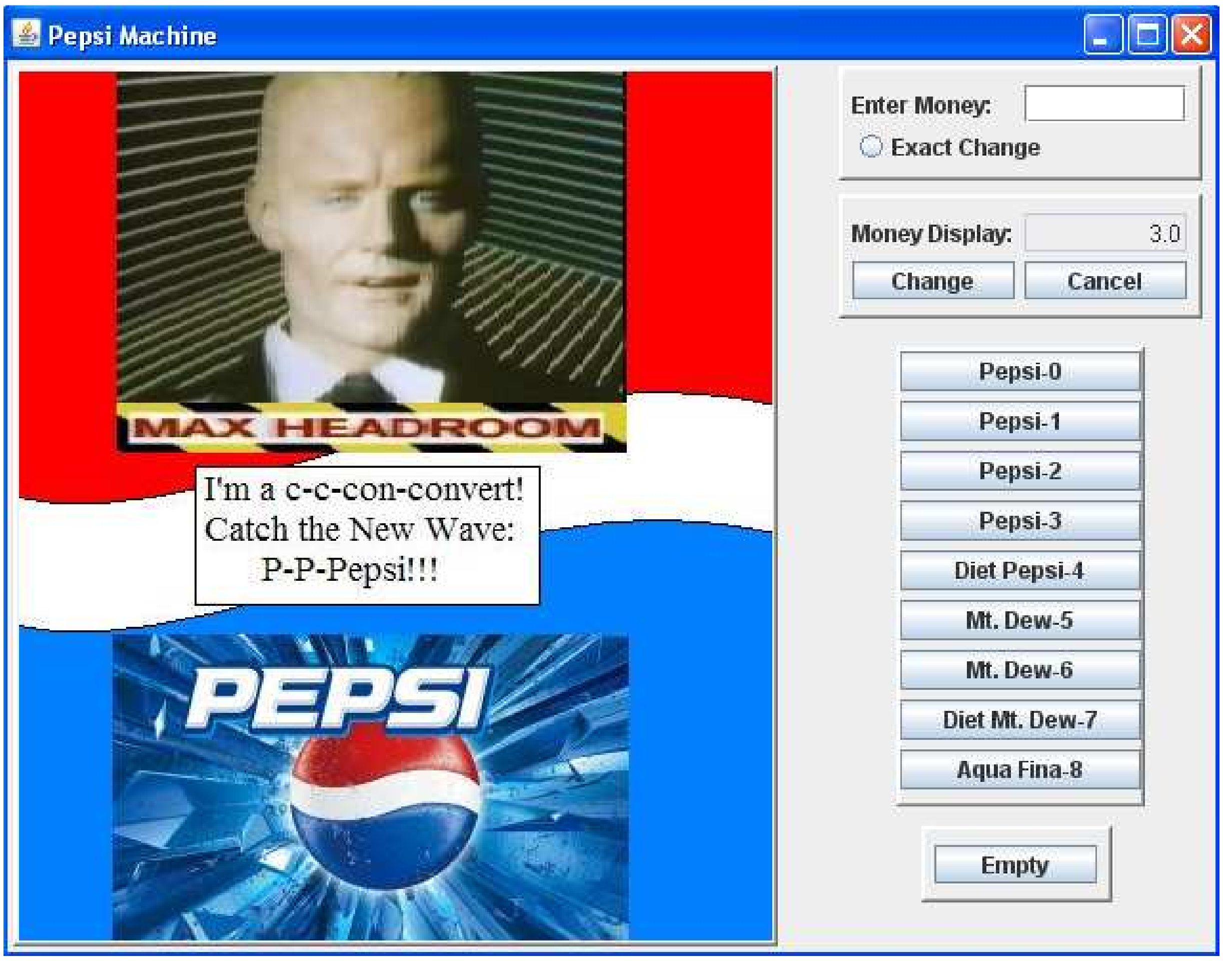

Class Diagrams
View_Class
oContainer : Container
oJPEnterMoney : JPanel
oJPMarketing : JPanel
oJPTakeProduct : JPanel
oJPProduct : JPanel
jtxtMoney : JTextField
jtxtMoney_Display : JTextField
jbtnProductBin : JButton
jbtnChangeBin : JButton
jbtnCancel : JButton
jrbnExactChange : JRadioButton
jbtnSelection : JButton
jlblImage : JLabel
jlblMoney : JLabel
jlblDisplay : JLabel
-View_Class()
Initialize_Money_GUI() : void
Initialize_Display_GUI() : void
Initialize_Marketing_GUI() : void
Initialize_Take_Product_GUI() : void
Initialize_Product_Selection_GUI() : void
Control_Class
-oMoney : Money_Class
+Control_Class()
Pepsi_Main_Class
+main(in args[ ] : string) : void
Take_Product_Event_Class
-oC : Control_Class
+Take_Product_Event_Class(in oControl : Control_Class)
+actionPerformed(in e : ActionEvent) : void
Money_Event_Class
-oC : Control_Class
+Money_Event_Class(in oControl : Control_Class)
+actionPerformed(in e : ActionEvent) : void
Take_Change_Event_Class
-oC : Control_Class
+Take_Change_Event_Class(in oControl : Control_Class)
+actionPerformed(in e : ActionEvent) : void
Validate_Money_Class
-oC : Control_Class
-iBinIndex : int
-dPrice : double
-iQty : int
+Validate_Money_Class(in oControl : Control_Class)
+Validate_Money() : bool
-Accumulate_Money(in dMoney : double, in oMoney : Money_Class) : double
-Valid_Money(in jtxtMoney : JTextField) : bool
-LT_EQ_Five(in jtxtMoney : JTextField) : bool
-Exact_Change(in jtxtMoney : JTextField, in iBI : int, in JRB : JRadioButton) : bool
-Change_No_Odd(in jbtnChangeBin : JButton) : bool
Refund_Event_Class
-oC : Control_Class
+Refund_Event_Class(in oControl : Control_Class)
+Refund_Event(in oMoney : Money_Class, in jbtnChangeBin : JButton) : void
+actionPerformed(in e : ActionEvent) : void
-Refund_No_Odd(in jbtnChangeBin : JButton) : bool
Money_Class
-dCurrent_Transaction : double
-dBank : double
+Money_Class()
+Reset_Current() : void
+Get_Current() : double
+Register_Sale(in dSale : double) : void
+Accumulate_Current(in dAmount : double) : void
+Get_Bank() : double
Exact_Change_Random_Class
-oC : Control_Class
+Exact_Change_Random_Class(in oControl : Control_Class)
+Rand_Exact(in bEC : bool) : bool
Dispense_Class
+dPrice : double
+iQty : int
+oC : Control_Class
+iBinIndex : int
+Dispense_Class(in oControl : Control_Class)
+actionPerformed(in e : ActionEvent) : void
-Get_Price_Qty(in jbtnSelection : JButton) : void
-Enough_Money(in oMoney : Money_Class) : bool
-Enough_Inventory() : bool
-Decrease_Inventory(in iBI : int) : void
-Calculate_Change(in oMoney : Money_Class, in jbtnChangeBin : JButton) : double
-Exact_Change(in iBT : int, in oMoney : Money_Class, in JRB : JRadioButton) : bool
-Update_Bank_Clear_Transaction(in oMoney : Money_Class) : bool
-Change_No_Odd(in jbtnChangeBin : JButton) : bool
-Bin_Empty(in jbtnProductBin : JButton) : bool
DB_Interface
+con : Connection
+s : Statement
+rs : ResultSet
+iBin : int
+sDesc : string
+dPrice : double = 0
+iOn_Hand : int
+iSold : int
+bOpened : bool = true
+bQuery : bool = true
+DB_Interface()
+Query(in sQuery : string) : void
+Update_OnHand_Sold(in sQuery : string) : void
1
*
1*

Figure 11

GUI


Pepsi Machine
MAX HEADROOM
I'm a c-c-con-convert! Catch the New Wave: P-P-Pepsi!!!
PEPSI
Enter Money:
Exact Change
Money Display: 3.0
Change
Cancel
Pepsi-0
Pepsi-1
Pepsi-2
Pepsi-3
Diet Pepsi-4
Mt. Dew-5
Mt. Dew-6
Diet Mt. Dew-7
Aqua Fina-8
Empty

Program Code

View_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/

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

public class View_Class extends JFrame
{
	final static long serialVersionUID = 5;

	final int iBT = 9;
	boolean bExact = false;

	//Layout
	Container oContainer;
	JPanel oJPEnterMoney;
	JPanel oJPMarketing;
	JPanel oJPTakeProduct;
	JPanel oJPProduct;

	//Product Layout
	GridLayout oGL1;

	//Objects
	JLabel jlblImage;
	JLabel jlblMoney;
	JLabel jlblDisplay;
	JTextField jtxtMoney;
	JTextField jtxtMoneyDisplay;
	JButton jbtnChangeBin;
	JButton jbtnCancel;
	JButton jbtnProductBin;
	JRadioButton jrbnExactChange;

	//Soda Buttons
	JButton[] oBT = new JButton[iBT];
	String sBT[] = {"Pepsi-0", "Pepsi-1", "Pepsi-2", "Pepsi-3", "Diet Pepsi-4", "Mt. Dew-5", "Mt. Dew-6", "Diet Mt. Dew-7", "Aqua
Fina-8"};
	Color oDefault;
```

```
public View_Class()
{
        super("Pepsi Machine");
        oContainer = getContentPane();
        oContainer.setLayout(null);

        //Initialize methods
        Initialize_Money_GUI();
        Initialize_Display_GUI();
        Initialize_Marketing_GUI();
        Initialize_Take_Product_GUI();
        Initialize_Product_Selection_GUI();

        setSize(640,500);
        setVisible(true);

}

//Initialize Money GUI
private void Initialize_Money_GUI()
{
        oJPEnterMoney = new JPanel();
        oJPEnterMoney.setLayout(null);
        oJPEnterMoney.setBorder(new BevelBorder(BevelBorder.RAISED));

        jlblMoney = new JLabel("Enter Money:");
        jlblMoney.setBounds(7, 10, 85, 20);
        oJPEnterMoney.add(jlblMoney);

        jtxtMoney = new JTextField(7);
        jtxtMoney.setHorizontalAlignment(JTextField.RIGHT);
        jtxtMoney.setBounds(97, 10, 85, 20);
        oJPEnterMoney.add(jtxtMoney);

        Exact_Change_Random_Class oECRC = new Exact_Change_Random_Class();
        bExact = oECRC.Rand_Exact(bExact);

        jrbnExactChange = new JRadioButton("Exact Change", bExact);
        jrbnExactChange.setBounds(7, 32, 105, 20);
        oJPEnterMoney.add(jrbnExactChange);

        oJPEnterMoney.setBounds(435, 3, 190, 60);
        oContainer.add(oJPEnterMoney);
}

//Initialize Display GUI
private void Initialize_Display_GUI()
{
        oJPEnterMoney = new JPanel();
        oJPEnterMoney.setLayout(null);
        oJPEnterMoney.setBorder(new BevelBorder(BevelBorder.RAISED));

        jlblMoney = new JLabel("Money Display:");
        jlblMoney.setBounds(7, 10, 85, 20);
        oJPEnterMoney.add(jlblMoney);

        jtxtMoneyDisplay = new JTextField(7);
        jtxtMoneyDisplay.setHorizontalAlignment(JTextField.RIGHT);
        jtxtMoneyDisplay.setEditable(false);
        jtxtMoneyDisplay.setBounds(97, 10, 85, 20);
        oJPEnterMoney.add(jtxtMoneyDisplay);

        jbtnChangeBin = new JButton("Change");
        jbtnChangeBin.setBounds(7, 35, 85, 20);
        oJPEnterMoney.add(jbtnChangeBin);

        jbtnCancel = new JButton("Cancel");
        jbtnCancel.setBounds(97, 35, 85, 20);
        oJPEnterMoney.add(jbtnCancel);

        oDefault = jbtnCancel.getBackground();
        oJPEnterMoney.setBounds(435, 70, 190, 65);
        oContainer.add(oJPEnterMoney);
}

//Initialize Marketing GUI
```

private void Initialize_Marketing_GUI()

```
{
        oJPMarketing = new JPanel();
        oJPMarketing.setLayout(null);
        oJPMarketing.setBorder(new BevelBorder(BevelBorder.RAISED));

        Icon icoMarketImage = new ImageIcon(getClass().getResource("pepsi_grand.jpg"));
        jlblImage = new JLabel(icoMarketImage);
        jlblImage.setBounds(3,3,394,454);
        oJPMarketing.add(jlblImage);

        oJPMarketing.setBounds(3, 3, 400, 460);
        oContainer.add(oJPMarketing);
}

//Initialize Take Product GUI
```

private void Initialize_Take_Product_GUI()

```
{
        oJPTakeProduct = new JPanel();
        oJPTakeProduct.setLayout(null);
        oJPTakeProduct.setBorder(new BevelBorder(BevelBorder.RAISED));
        jbtnProductBin = new JButton("Empty");
        jbtnProductBin.setBounds(5, 5, 120, 30);
        oJPTakeProduct.add(jbtnProductBin);

        oJPTakeProduct.setBounds(465, 400, 130, 40);
        oContainer.add(oJPTakeProduct);
    }

    //Initialize Product Selection GUI
    private void Initialize_Product_Selection_GUI()
    {
        oJPProduct = new JPanel();
        oGL1 = new GridLayout (9,1,5,5);
        oJPProduct.setLayout(oGL1);
        oJPProduct.setBorder(new BevelBorder(BevelBorder.RAISED));

        for(int x= 0; x < iBT; x++)
        {
            oBT[x] = new JButton(sBT[x]);
            oBT[x].setHorizontalAlignment(SwingConstants.CENTER);
            oJPProduct.add(oBT[x]);
        }
        oJPProduct.setBounds(465, 150, 130, 240);
        oContainer.add(oJPProduct);
    }
}
```

Control_Class

```
//NTC Java Programming 152-116
//Instructor: John Heckendorf
//Student: Jan Young & Keith Gallistel
//Lab ID:<Pepsi>

public class Control_Class extends View_Class
{
    final static long serialVersionUID = 5;

    final int iBT = 9;

    Money_Class oMoney = new Money_Class();

    //Control Class
    public Control_Class()
    {
        //Use Case 1 - Enter Money
        Money_Event_Class oMEC = new Money_Event_Class(this);
        jtxtMoney.addActionListener(oMEC);
```

```
		//Use Case 2 - Make Selection
		Dispense_Class oDC = new Dispense_Class(this);
		//For loop for generated buttons
		for(int x= 0; x < iBT; x++)
		{
			oBT[x].addActionListener(oDC);
		}

		//Use Case 3 - Take Product
		Take_Product_Event_Class oTPC = new Take_Product_Event_Class(this);
		jbtnProductBin.addActionListener(oTPC);

		//Use Case 4 - Take Change
		Take_Change_Event_Class oTCC = new Take_Change_Event_Class(this);
		jbtnChangeBin.addActionListener(oTCC);

		//Use Case 5 - Cancel Purchase
		Refund_Event_Class oREC = new Refund_Event_Class(this);
		jbtnCancel.addActionListener(oREC);

		//End Of Job Class
		EOJ_Class oEOJ = new EOJ_Class(this);
		this.addWindowListener(oEOJ);
	}
}
```

Main_Pepsi_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>
*/

//Main application class
import javax.swing.*;

public class Pepsi_Main_Class
{
	final static long serialVersionUID = 5;

	//Method which launches the application
	public static void main(String arg[])
	{
		Control_Class oControl = new Control_Class();
		oControl.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Money_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/
public class Money_Class
{
	final static long serialVersionUID = 5;

	private double dCurrent_Transaction = 0;
	private double dBank = 50;

	//Instantiated Class
	public Money_Class()
	{
	}

	//Get money in bank
	public double Get_Bank()
	{
		double dB = 0;
		dB = dBank;
		return dB;
	}

	//Get amount of current money customer set
	public double Get_Current()
	{
		double dCurrent = 0;
		dCurrent = dCurrent_Transaction;
		return dCurrent;
	}

	//Accumulate the new current amount with old current amount
	public void Accumulate_Current(double dAmount)
	{
		dCurrent_Transaction += dAmount;
	}

	//Set current to zero
	public void Reset_Current()
	{
		dCurrent_Transaction = 0;
	}
	//Add current to bank
	public void Register_Sale(double dSale)
	{
		dBank += dSale;
	}
}
```

Money_Event_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/
import javax.swing.*;
import java.awt.event.*;

public class Money_Event_Class extends JFrame implements ActionListener
{
	final static long serialVersionUID = 5;

	Control_Class oC;

	//Instantiated Class
	public Money_Event_Class(Control_Class oControl)
	{
		oC = oControl;
	}

	//Method that will process GUI events
	public void actionPerformed(ActionEvent e)
	{
		boolean bGood = true;
		Validate_Money_Class oVMC = new Validate_Money_Class(oC);
		bGood = oVMC.Validate_Money();
	}
}
```

Dispense_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/
import javax.swing.*;
import java.util.*;
import java.awt.Color;
import java.awt.event.*;

public class Dispense_Class extends JFrame implements ActionListener
{
	final static long serialVersionUID = 5;

	//variables
	Control_Class oC;

	int iBinIndex = 0;
	double dPrice = 0;
	int iQty = 0;

	//Instantiated Class
	public Dispense_Class(Control_Class oControl)
	{
		oC = oControl;
	}

	//Method that will process GUI events
	public void actionPerformed(ActionEvent e)
	{
		boolean bEnoughMoney = false;
		boolean bEnoughInventory = false;
		boolean bExactChange = false;
		boolean bProductBinEmpty = false;
		boolean bChangeBinNoOdd = false;

		for(int x = 0; x < oC.iBT; x++)
		{
			if(e.getSource() == oC.oBT[x])
			{
				iBinIndex = x;
			}
		}
		Get_Price_Qty(iBinIndex);

		bEnoughMoney = Enough_Money(oC.oMoney);
		bEnoughInventory = Enough_Inventory();
		bExactChange = Exact_Change(iBinIndex, oC.oMoney, oC.jrbnExactChange);
		bProductBinEmpty = Bin_Empty(oC.jbtnProductBin);
		bChangeBinNoOdd = Change_No_Odd(oC.jbtnChangeBin);

		if(bEnoughMoney && bEnoughInventory && bExactChange && bProductBinEmpty && bChangeBinNoOdd)
		{
			Decrease_Inventory(iBinIndex);
			oC.jbtnProductBin.setText(oC.sBT[iBinIndex].toString());
			Calculate_Change(oC.jbtnChangeBin, oC.oMoney);
```

```
                oC.bExact = Update_Bank_Clear_Transaction(oC.oMoney);
                oC.jtxtMoneyDisplay.setText(null);
        }
}

//Accesses database to get the price and quantity of product
private void Get_Price_Qty(int iBI)
{
        String sBI = "";
        String sQ = "";

        sBI = Integer.toString(iBI);
        sQ ="SELECT * FROM Products WHERE pm_bin_number = " + sBI + ";";

        DB_Interface oDBI = new DB_Interface();
        oDBI.Query(sQ);

        dPrice = oDBI.dPrice;
        iQty = oDBI.iOn_Hand;
}

//Checks if there is enough money in current
private boolean Enough_Money(Money_Class oMoney)
{
        boolean bEMoney = true;

        if(dPrice > oMoney.Get_Current())
        {
                bEMoney = false;
        }
        return bEMoney;
}

//Checks if quantity is greater than one
private boolean Enough_Inventory()
{
        boolean bEInv = true;

        if(iQty < 1)
        {
                bEInv = false;
        }
        return bEInv;
}

//Checks if product bin is empty
private boolean Bin_Empty (JButton jbtnProductBin)
{
        boolean bBin = true;

        if(jbtnProductBin.getText() != "Empty")
        {
                jbtnProductBin.setBackground(Color.RED);
                bBin = false;
        }
        return bBin;
}

//Checks if change bin contains garbage
private boolean Change_No_Odd(JButton jbtnChangeBin)
{
        boolean bVM = false;
        double dData = 0;

        if(jbtnChangeBin.getText() != "Change")
        {
                try
                {
                        dData = Double.parseDouble(jbtnChangeBin.getText());
                        bVM = true;
                }
                catch(NumberFormatException e)
                {
                        bVM = false;
                        jbtnChangeBin.setBackground(Color.RED);
                }
        }
        else
        {
                bVM = true;
        }
        return bVM;
}

//Decreases the on-hand inventory in the database
private void Decrease_Inventory(int iBI)
{
        String sBI = "";
        String sQ = "";

        sBI = Integer.toString(iBI);
        sQ ="SELECT * FROM Products WHERE pm_bin_number = " + sBI + ";";

        DB_Interface oDBI = new DB_Interface();
        oDBI.Update_OnHand_Sold(sQ);

}
```

```
//Calculates the change
private void Calculate_Change(JButton jbtnChangeBin, Money_Class oMoney)
{
	String sCBin;
	double dCBin;
	double dCurrent;
	double dNewCurrent;
	String sNewCurrent;
	double dCalc;
	String sCalc;

	sCBin = jbtnChangeBin.getText();
	dCurrent = oMoney.Get_Current();

	try
	{
		Formatter oF = new Formatter();

		dCalc = dCurrent - dPrice;
		oF.format("%,.2f", dCalc);
		sCalc = oF.toString();

		if(dCalc >= 0)
		{
			if(sCBin == "Change" && dCalc != 0.0)
			{
				jbtnChangeBin.setText(sCalc);
			}

			if(sCBin != "Change")
			{
				Formatter oH = new Formatter();

				dCBin = Double.parseDouble(jbtnChangeBin.getText());
				dNewCurrent = dCalc + dCBin;
				oH.format("%,.2f", dNewCurrent);
				sNewCurrent = oH.toString();
				jbtnChangeBin.setText(sNewCurrent);
			}
		}
	}
	catch(NumberFormatException e)
	{
	}
}

//Test for Exact Change
```

```
private boolean Exact_Change(int iBI, Money_Class oMoney, JRadioButton JRB)
{
        boolean bEC = false;
        double dData = 0;
        double dPrice = 0;

        String sBI = "";
        String sQ = "";

        sBI = Integer.toString(iBI);
        sQ ="SELECT * FROM Products WHERE pm_bin_number = " + sBI + ";";

        DB_Interface oDBI = new DB_Interface();
        oDBI.Query(sQ);

        dPrice = oDBI.dPrice;
        dData = oMoney.Get_Current();

        if(JRB.isSelected() == true)
        {
                if(dPrice == dData)
                {
                        bEC = true;
                }
        }
        else
        {
                bEC = true;
        }
        return bEC;
}

//Update the bank and clear current
private boolean Update_Bank_Clear_Transaction(Money_Class oMoney)
{
        boolean bExact = false;

        double dCurrent = 0;
        double dSale = 0;
        double dBank = 0;

        dCurrent = oMoney.Get_Current();
        dSale = dCurrent - dPrice;
        oMoney.Register_Sale(dSale);
        oMoney.Reset_Current();
        dBank = oMoney.Get_Bank();

        if(dBank < 5.00)
        {
                bExact = true;
            }
            return bExact;
        }
}
```

Take_Change_Event_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/

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

public class Take_Change_Event_Class extends JFrame implements ActionListener
{
        final static long serialVersionUID = 5;

        //variable
        Control_Class oC;

        //Instantiated Class
        public Take_Change_Event_Class(Control_Class oControl)
        {
                oC = oControl;
        }

        //Method that will process GUI events
        public void actionPerformed(ActionEvent e)
        {
                //Resets change bin
                if(oC.jbtnChangeBin.getText() != "Change")
                {
                        oC.jbtnChangeBin.setText("Change");
                }

                //Restores generic color to change bin button
                if(oC.jbtnChangeBin.getBackground() != oC.oDefault)
                {
                        oC.jbtnChangeBin.setBackground(oC.oDefault);
                }
        }
}
```

Validate_Money_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/

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

public class Validate_Money_Class
{
	final static long serialVersionUID = 5;

	Control_Class oC;

	int iBinIndex = 0;
	double dPrice = 0;
	int iQty = 0;

	//Instantiated Class
	public Validate_Money_Class(Control_Class oControl)
	{
		oC = oControl;
	}

	//Validate Money Method
	public boolean Validate_Money()
	{
		boolean bGood = true;

		boolean bValid = false;
		boolean bLEFive = false;
		boolean bExactChange = false;
		boolean bChangeBinNoOdd = false;
		boolean bInvalid = false;

		double dMoneyConvert = 0;
		double dTotAcum = 0;

		Formatter oF = new Formatter();

		bValid = Valid_Money(oC.jtxtMoney);
		bLEFive = LT_EQ_Five(oC.jtxtMoney);
		bExactChange = Exact_Change(iBinIndex, oC.jtxtMoney, oC.jrbnExactChange);
```

```
          bChangeBinNoOdd = Change_No_Odd(oC.jbtnChangeBin);
          bInvalid = Change_Full_Money_Odd(oC.jtxtMoney, oC.jbtnChangeBin);

          if(bValid && bLEFive && bExactChange)
          {
                    dMoneyConvert = Double.parseDouble(oC.jtxtMoney.getText());
                    dTotAcum = Accumulate_Money(dMoneyConvert, oC.oMoney);
                    oF.format("%,.2f", dTotAcum);
                    oC.jtxtMoneyDisplay.setText(oF.toString());
                    oC.jtxtMoney.setText(null);
          }
          else
          {
                    if(bChangeBinNoOdd && bInvalid)
                    {
                              bGood = false;
                              oC.jbtnChangeBin.setText(oC.jtxtMoney.getText());
                              oC.jtxtMoney.setText(null);
            }
                    else
                    {
                              bGood = false;
                              oC.jbtnChangeBin.setBackground(Color.RED);
                    }
          }
          return bGood;
}

//Validate whether money is good
```
private boolean Valid_Money(JTextField JTF)
```
{
          boolean bVM = false;
          double dData = 0;

          try
          {
                    dData = Double.parseDouble(JTF.getText());
                    bVM = true;
          }
          catch(NumberFormatException e)
          {
                    bVM = false;
          }
          return bVM;
}

//Validate if money is less than or equal to $5.00
```
private boolean LT_EQ_Five(JTextField JTF)
```
{
        boolean bLEF = false;
        double dData = 0;

        try
        {
                dData = Double.parseDouble(JTF.getText());
        }
        catch(NumberFormatException e)
        {
                bLEF = false;
        }

        if(dData <= 5)
        {
                bLEF = true;
        }
        return bLEF;
}

//Validate whether you need to use exact change
```

private boolean Exact_Change(int iBI, JTextField JTF, JRadioButton JRB)

```
{
        boolean bEC = false;
        double dData = 0;
        double dPrice = 0;

        String sBI = "";
        String sQ = "";

        sBI = Integer.toString(iBI);
        sQ ="SELECT * FROM Products WHERE pm_bin_number = " + sBI + ";";

        DB_Interface oDBI = new DB_Interface();
        oDBI.Query(sQ);
        dPrice = oDBI.dPrice;

        try
        {
                dData = Double.parseDouble(JTF.getText());
        }
        catch(NumberFormatException e)
        {
        }

        if(JRB.isSelected() == true)
        {
                if(dPrice == dData)
                {
                        bEC = true;
                }
            }
            else
            {
                bEC = true;
            }
            return bEC;
}
```

//Validate if change is money or garbage

private boolean Change_No_Odd(JButton jbtnChangeBin)

```
{
            boolean bVM = false;
            double dData = 0;

            if(jbtnChangeBin.getText() != "Change")
            {
                try
                {
                    dData = Double.parseDouble(jbtnChangeBin.getText());
                    bVM = true;
                }
                catch(NumberFormatException e)
                {
                    bVM = false;
                }
            }
            else
            {
                bVM = true;
            }
            return bVM;
}
```

//Validate if change bin is full and money is garbage

private boolean Change_Full_Money_Odd (JTextField JTF, JButton JBTN)

```
{
            boolean bOdd = false;
            double dData = 0;

            if(JBTN.getText() != "Change")
            {
                try
                {
                    dData = Double.parseDouble(JTF.getText());
                    bOdd = true;
                }
                catch(NumberFormatException e)
                {
                        bOdd = false;
                    }
                }
                else
                {
                    bOdd = true;
                }
                return bOdd;
        }

        //Accumulate the money in the bank
        private double Accumulate_Money(double dMoney, Money_Class oMoney)
        {
                double dReturn = 0;
                oMoney.Accumulate_Current(dMoney);
                dReturn = oMoney.Get_Current();
                return dReturn;
        }
}
```

Take_Product_Event_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/

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

public class Take_Product_Event_Class extends JFrame implements ActionListener
{
        final static long serialVersionUID = 5;

        //variable
        Control_Class oC;

        //Instantiated Class
        public Take_Product_Event_Class(Control_Class oControl)
        {
                oC = oControl;
        }

        //Method that will process GUI events
        public void actionPerformed(ActionEvent e)
        {
                //Resets product bin
```

```
		if(oC.jbtnProductBin.getText() != "Empty")
		{
			oC.jbtnProductBin.setText("Empty");
		}

		//Restores generic color to product bin button
		if(oC.jbtnProductBin.getBackground() != oC.oDefault)
		{
			oC.jbtnProductBin.setBackground(oC.oDefault);
		}
	}
}
```

Exact_Change_Random_Class

```
/*
NTC Java Programming 152-116
Instructor: John Heckendorf
Student: Jan Young & Keith Gallistel
Lab ID: <Pepsi>  & 46314
Project:pepsi.java
*/
import java.util.Random;

public class Exact_Change_Random_Class
{
	final static long serialVersionUID = 5;
```

public Exact_Change_Random_Class()

```
	{
	}

	//This sets the Exact Change button randomly or if bank is less than $5.00
```

public boolean Rand_Exact(boolean bEC)

```
	{
		boolean bExact = false;
		double dBank = 0;
		Random randomNumbers = new Random();
		int iNum = randomNumbers.nextInt(10);

		if(iNum > 5)
		{
			bExact = true;
		}
		Money_Class oMoney = new Money_Class();
		dBank = oMoney.Get_Bank();

		if(dBank < 5)
		{
			bExact = true;
		}
		return bExact;
	}
}
```

EOJ_Class

```
//NTC Java Programming 152-116
//Instructor: John Heckendorf
//Student: Jan Young & Keith Gallistel
//Lab ID:<Pepsi>

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

public class EOJ_Class extends JFrame implements WindowListener
{
	final static long serialVersionUID = 5;

	Control_Class oC;
```

public EOJ_Class(Control_Class oControl)

```
	{
		oC = oControl;
	}
	public void windowActivated(WindowEvent e)
	{
	}
	public void windowDeactivated(WindowEvent e)
	{
	}
	public void windowOpened(WindowEvent e)
	{
	}
	public void windowClosing(WindowEvent e)
	{
		System.exit(0);
	}

	public void windowClosed(WindowEvent e)
	{
	}

	public void windowIconified(WindowEvent e)
	{
	}

	public void windowDeiconified(WindowEvent e)
	{
	}
```

```
}
```

DB_Interface

```
/*	==============================================
	JAVA DB_Interface
	==============================================
	Java coding co-authored by Clifford Berg & John Heckendorf
	Documentation authored by Clifford Berg & John Heckendord
	Code was modified by Heck for the Java Pepsi UML project 04/17/2007
*/

import java.sql.*;
import javax.swing.*;

public class DB_Interface
{
	//Connection object
	Connection con;

	//SQL statement object
	Statement s;

	//Data set object
	ResultSet rs;

	//Fields are declared public
	//Directly accessible by the object
	//Will maintain state
	public int iBin = 0;
	public String sDesc = "";
	public double dPrice = 0;
	public int iOn_Hand = 0;
	public int iSold = 0;

	public boolean bOpened = true;
	public boolean bQuery = true;

//	---------------------------------------------------------------
//	Constructor Method:
//		-Set up ODBC datasource  = PepsiMachine to database pepsi_machine.mdb
//		-Set up connection object and SQL statement object for querying
//	---------------------------------------------------------------
	public DB_Interface()
	{
		ClearData();

		try
		{
				Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
				//JavaDB3 is the DSN ODBC data source name you assigned when you selected this database
				//	Start, setting, control panel, adminstration tools, data sources (ODBC),
				//	System DSN, add, double click MS Acess *.mdb, Give your DSN a name, Select (find it), OK, OK

				con = DriverManager.getConnection("jdbc:odbc:PepsiMachine","","");
				//-------------------------------------------------------
				// Must include these ResultSet parameters to enable ResultSet to be updatable!
				//-------------------------------------------------------
				s = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
			ResultSet.CONCUR_UPDATABLE);
			}
			catch(SQLException err)
			{
				JOptionPane.showMessageDialog(null,"Error #1 - " + err.toString( )
					"Error", JOptionPane.ERROR_MESSAGE);
				bOpened = false;
			}

			catch(Exception err)
			{
				JOptionPane.showMessageDialog(null,"Error #2 - " + err.toString( ),
					"Error", JOptionPane.ERROR_MESSAGE);
				bOpened = false;
			}
		}
//		----------------------------------------------------------------
//		Run query based on user selection on tbl_products to extract  pertinent values from ResultSet.
//		----------------------------------------------------------------
		public void Query(String sQuery)
		{
			bQuery = true;
			ClearData();

			try
			{
				rs = s.executeQuery(sQuery);

			//	Other rs methods = afterLast( ), beforeFirst( ), first( ), last( ), previous( )
			//	rs.next() Must be called after a query, it sets the point to the beginning of the result set
			//	unless, you use the absolute statement
//				while(rs.next()) //Get/loop through all records in a multiple record result set
//				{
					rs.absolute(1); //In a one record result set, set the pointer to the first/only record
					iBin = rs.getInt("pm_bin_number");
					sDesc = rs.getString("pm_description");
					dPrice = rs.getDouble("pm_price");
					iOn_Hand = rs.getInt("pm_on_hand");
					iSold = rs.getInt("pm_sold");
//				}

			}
		catch(NumberFormatException err)
		{
			JOptionPane.showMessageDialog(null,"Error #10 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(SQLException err)
		{
			JOptionPane.showMessageDialog(null,"Error #11 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(Exception err)
		{
			JOptionPane.showMessageDialog(null,"Error #12 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		if(bOpened)
			CloseDB();
	}
//	----------------------------------------------------------------
//	Run query based on user selection on it will update ResultSet first and subsequently
//	the database. This method is safer than directly updating the database
//	----------------------------------------------------------------
	public void Update_OnHand_Sold(String sQuery)
	{
		bQuery = true;
		ClearData();
		//Update - change the result set then update the database
		//Safer logic structure
		// Command to directly update the database s.executeUpdate(sString);

		try
		{
			rs = s.executeQuery(sQuery);

			rs.absolute(1);
			iBin = rs.getInt("pm_bin_number");
			sDesc = rs.getString("pm_description");
			dPrice = rs.getDouble("pm_price");
			iOn_Hand = rs.getInt("pm_on_hand");
			iSold = rs.getInt("pm_sold");

			rs.absolute(1);
			rs.updateInt("pm_on_hand", rs.getInt("pm_on_hand") - 1);
			rs.updateInt("pm_sold", rs.getInt("pm_sold") + 1);
			rs.updateRow();

		}
		catch(NumberFormatException err)
		{
			JOptionPane.showMessageDialog(null,"Error #20 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(SQLException err)
		{
			JOptionPane.showMessageDialog(null,"Error #21 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(Exception err)
		{
			JOptionPane.showMessageDialog(null,"Error #22 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		if(bOpened)
			CloseDB();
	}

	//Reset database values to their original values
	public void Reset_OnHand_Sold(String sQuery)
	{
		bQuery = true;
		ClearData();

		//Update - change the result set then update the database
		//Safer logic structure
		// Command to directly update the database s.executeUpdate(sString);

		try
		{
			rs = s.executeQuery(sQuery);

			while(rs.next())
			{
				rs.updateInt("pm_on_hand", 25);
				rs.updateInt("pm_sold", 0);
				rs.updateRow();
			}

		}
		catch(NumberFormatException err)
		{
			JOptionPane.showMessageDialog(null,"Error #20 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(SQLException err)
		{
			JOptionPane.showMessageDialog(null,"Error #21 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(Exception err)
		{
			JOptionPane.showMessageDialog(null,"Error #22 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		if(bOpened)
			CloseDB();
	}

	//Reset database values to their original values
	public void Update_Query(String sQuery)
	{
		bQuery = true;
		int iRows = 0;

		ClearData();

		//Update - change the result set then update the database
		//Safer logic structure
		// Command to directly update the database s.executeUpdate(sString);

		try
		{
			iRows = s.executeUpdate(sQuery);
		}
		catch(NumberFormatException err)
		{
			JOptionPane.showMessageDialog(null,"Error #20 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(SQLException err)
		{
			JOptionPane.showMessageDialog(null,"Error #21 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}

		catch(Exception err)
		{
			JOptionPane.showMessageDialog(null,"Error #22 - " + err.toString( ),
				"Error", JOptionPane.ERROR_MESSAGE);
			bQuery = false;
		}
		if(bOpened)
			CloseDB();

	}
//	---------------------------------------------------------------
//	Close the database connection and SQL statement objects
//	---------------------------------------------------------------
```

```
        private void CloseDB()
        {
                try
                {
                        s.close( );
                        con.close( );
                }
                catch(SQLException err)
                {
                        JOptionPane.showMessageDialog(null,"Error #100 - " + err.toString( ),
                                "Error", JOptionPane.ERROR_MESSAGE);
                }
                catch(Exception err)
                {
                        JOptionPane.showMessageDialog(null,"Error #101 - " + err.toString( ),
                                "Error", JOptionPane.ERROR_MESSAGE);
                }
                bOpened = false;
        }
//      ---------------------------------------------------------------
//      Clear any possible populated data fields prior to querying
//      ---------------------------------------------------------------
        public void ClearData()
        {
                iBin = 0;
                sDesc = "";
                dPrice = 0;
                iOn_Hand = 0;
                iSold = 0;
        }
}//----------end of DB_Interface class----------
```