UNIVERSITÉ Concordia UNIVERSITY

MODEL DRIVEN SOFTWARE ENGINEERING

COEN 6312, WINTER 2016

DELIVERABLE: 4

STATE MACHINES

Submitted By:

Desai Parth Mukeshbhai 27397364

Jarin Manuvel Mathew 27168470

Manjot Singh Gill 27299699

Vivek Khatri 27292848

Dollar Kumar Bansal 27599188

Contents

Objective 3

State Diagram 3

State Transition: 3

Start State: 4

Final State 4

Guarded Transition 4

Unlabeled Transition 4

State Diagram for Class SalesOrder 5

Explanation 5

Action Specification Code in Java 6

State Diagram for Class Payment 8

Explanation 8

Action Specification Code in Java 8

State Diagram for Class OrderDelivery 11

Explanation 11

Action Specification Code in Java 11

State Diagram for Class Inventory Stock 13

Explanation 13

Action Specification Code in Java 13

References 15

Objective

The objective of this deliverable is to show and explore the action specific or dynamic states representing executable UML. The particular model tries to describe and define how our created system would react to certain events or scenarios. It covers the dynamic nature of the system and explains it through event diagrams and Action specific language (In our case, Java).

State Diagram

A state diagram is a graph consisting of states and the transition between these defined states or state diagrams are used to specify the sequencing/timing behavior of objects. Every state diagram in UML has a start state and a final state. There is always a state diagram for every class.

In a state diagram a state can be defined as a constraint or a situation in the life cycle of an object, in which a constraint holds, the object executes an activity or waits for an event. States can be of two types, simple or composite.

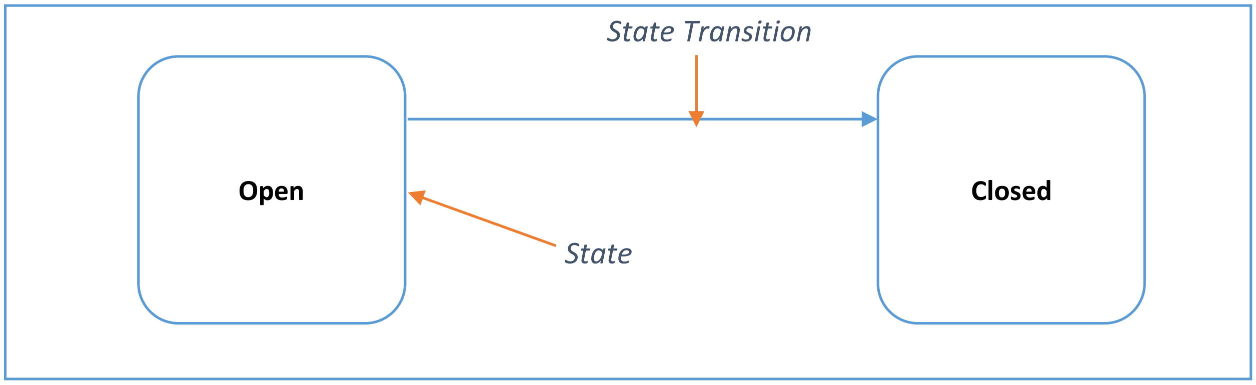


Figure 1: Sample State Diagram

State Transition:

A state transition can be defined as the progression in the state. Whenever there is a transition in the state that means the system is moving to a new state.

Example:

In figure 1 when there was a state transition that took place at the open state it progressed towards the closed state.

Start State:

The start state is the state that a new object will be in immediately following its creation. There can only be only one start state for a state machine.

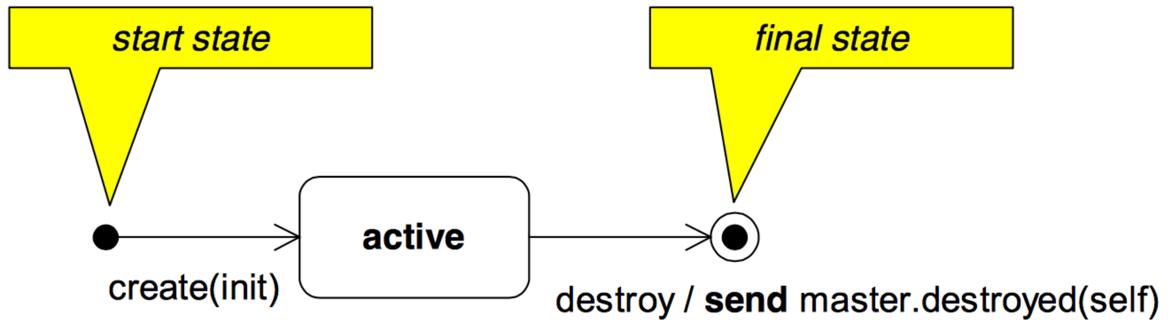


Figure 2: Example

Final State

Final state is the state which shows the destruction of the object or the object is going out of existence.

Guarded Transition

A guarded transition can be defined as a shorthand notation that says any event that is taking place the guard condition must also be true for the transition to take place.

Unlabeled Transition

An unlabeled transition is the one in which a transition will take place when the activity (processing, below) completes.

State Diagram for Class SalesOrder

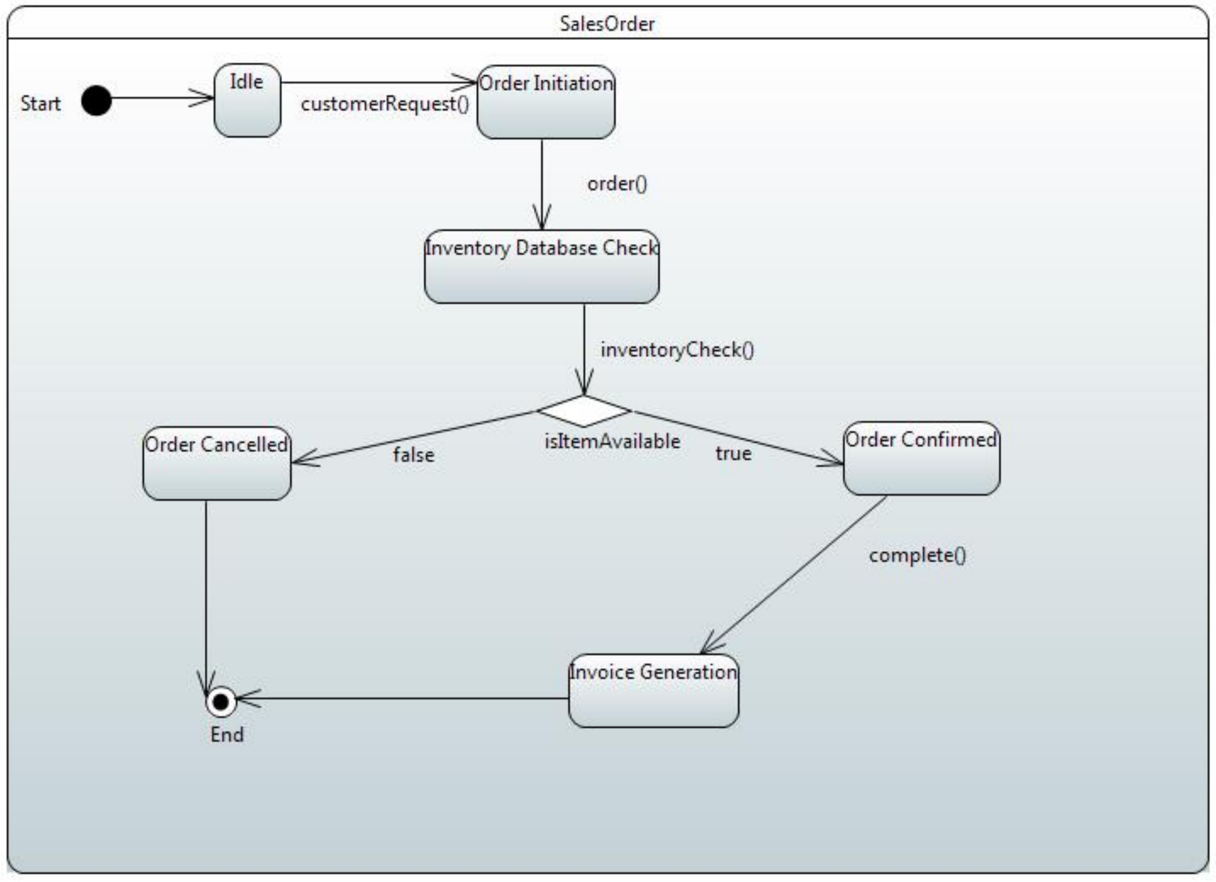


Figure 3: State Diagram of Class Sales Order

Explanation

System is in idle state initially. A customer request initiates the order. Inventory database is then checked to see if all the ordered Items are available in the database. Inventory database checks the two given scenarios:-

Scenario 1 – All the items listed in the order are available in the inventory. In this case the order is confirmed and an invoice is generated, which leads to the end of Sales order.

Scenario 2 – In case all or some of the items listed in the order are not available in the inventory, the order is then said to be cancelled and the leads to the end of Sales order.

Action Specification Code in Java

```
import java.sql.Date;
import java.util.Calendar;
import java.util.Scanner;
public class order {
	static int orderNum = 0 ;
	static Date orderDate;
	static String custId;
	static String empId;
	static int itemId;
	static int orderAmount;
	static String shippingAddress;
	static Date shippingDate;
	static int orderStatus;
	static int noOfItems;
	public static boolean isItemAvailable(int itemId)
	{
		boolean present = false;
		// make present = true if item present
		if(present)
			return true ; // if item available
		else
			return false;
		//code for checking if the item present in inventory or not
	}
	public static void invoiceGeneration()
	{
		//code for generating the invoice
	}
	public static void createOrder()
	{
		Scanner sc = new Scanner(System.in);
		System.out.println("*************Enter Order details*************/n");
		System.out.println("Enter shipping address");
		String s = sc.nextLine();
		shippingAddress = s;
		System.out.println("Enter item ID ");
		int id = sc.nextInt();
		itemId = id;
		System.out.println("Enter customer id");
```

```
            String cid = sc.nextLine();
            custId = cid;
            System.out.println("Enter order amount ");
            int amt = sc.nextInt();
            orderAmount = amt;
            System.out.println("Enter employee id");
            String eid = sc.nextLine();
            empId = eid;
            System.out.println("Enter number of items");
            int num = sc.nextInt();
            noOfItems = num;
            orderNum++;
            Calendar cal = Calendar.getInstance();
            orderDate = (Date) cal.getTime();
            }

public static boolean inventoryCheck(int itemId)
{
            boolean itemPresent = false;
            if(isItemAvailable(itemId))
            {
                        itemPresent = true;
            }
            return itemPresent;
}

public static void main(String args[])
{
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter Order details");
            createOrder();
            if (inventoryCheck(itemId))
            {
                        System.out.println("Order Confirmed");
                        invoiceGeneration();
            }
            else
            {
                        System.out.println("Item not present in Inventory. Order Cancelled.");
            }
```

```
        }
}
```

State Diagram for Class Payment

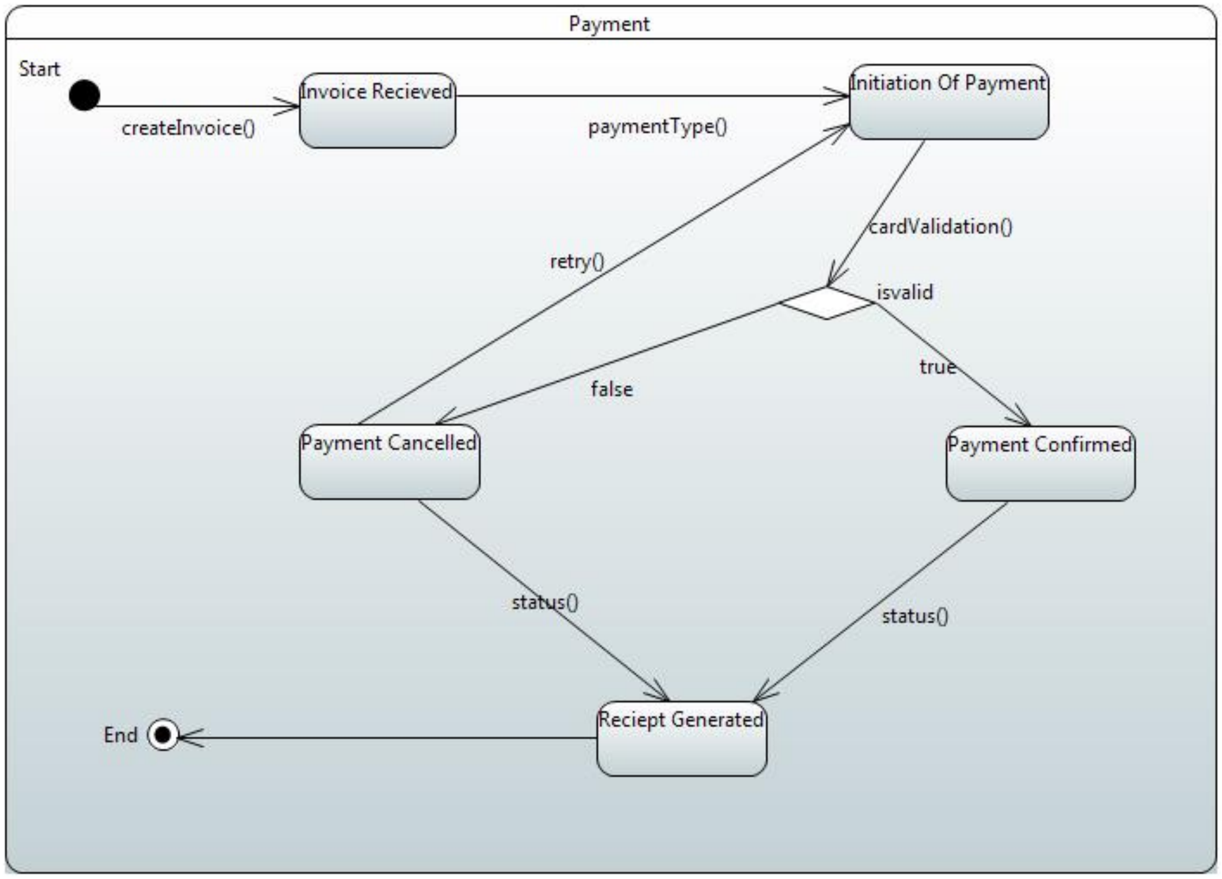


Figure 4: State Diagram of Class Payment

Explanation

Once an Invoice is generated by the cashier, the payment system receives the invoice and asks for the type/mode of payment. Once the mode of payment is specified, the payment is then initiated. This further leads to two main scenarios:-

Scenario 1 – The mode of payment is valid. In this the case, the payment is then confirmed and a receipt is generated for the payment.

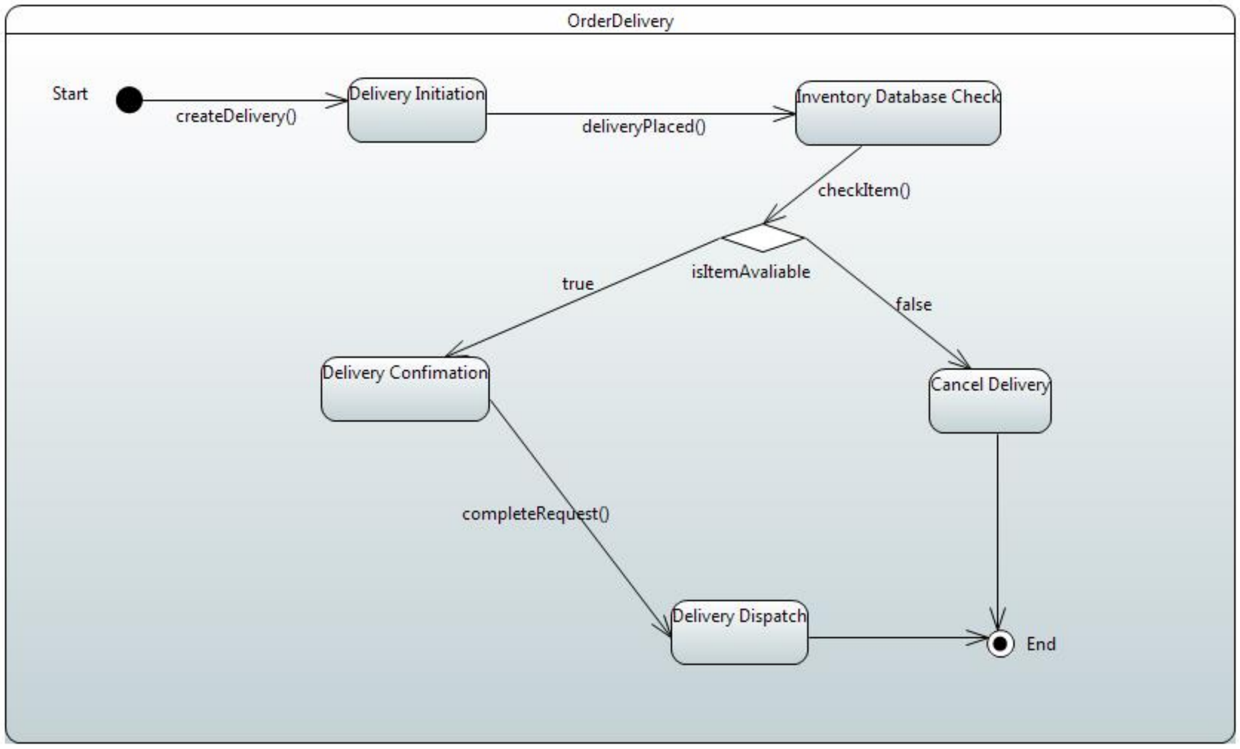
Scenario 2 – If the mode of payment is invalid, then either the payment is retried again or the payment is declared as failed and a receipt is generated to state that.

Action Specification Code in Java

```
import java.sql.Date;
```

```
            }


import java.util.Scanner;
public class payment {
        public static boolean createInvoice()
        {
                boolean success = false;
                // code for invoice creation
                // if successfully creates invoice then set success = true
                if(success)
                        return true;
                else
                        return false;
        }
        public static void generateReciept()
        {
                //code for generation of receipt
        }
        public static void inputDebitCardInfo()
        {
                //take debit card info
        }
        public static void inputCreditCardInfo()
        {
                //take credit card info
        }
        public static boolean paymentType(String PaymentType)
        {
                boolean success  = false;

                switch(PaymentType)
                {
                case("Debit"): inputDebitCardInfo();
                if(validateCard())
                        success = true;

                case("Credit"):inputCreditCardInfo();
                if(validateCard())
                        success = true;
                }
                return success;
        }
        public static boolean validateCard()
        {
                return false;
                //If valid then return true
                // else return false
        }
        public static void main(String args[])
        {
                int Bill_No;
                int Delivery_No;
                Float PaymentAmount;
                Scanner sc = new Scanner(System.in);
                Date Date;  // Edate is a defined structure
                boolean payment_success;
                if(createInvoice())
                {
                        System.out.println("Enter the Payment Type");
                        String PaymentType = sc.nextLine();
                        int number_of_try = 0; // number of try is limited to 3
                        do{
                                payment_success = paymentType(PaymentType);
                                number_of_try++;

                        }while(payment_success || number_of_try < 3);


                        generateReciept();
                }
        }
}
```

State Diagram for Class OrderDelivery



Explanation

Once a delivery is created, the delivery process initiates and a delivery is placed. The inventory database is then checked to seek if all the items to be delivered are available. This further leads to two main scenarios:-

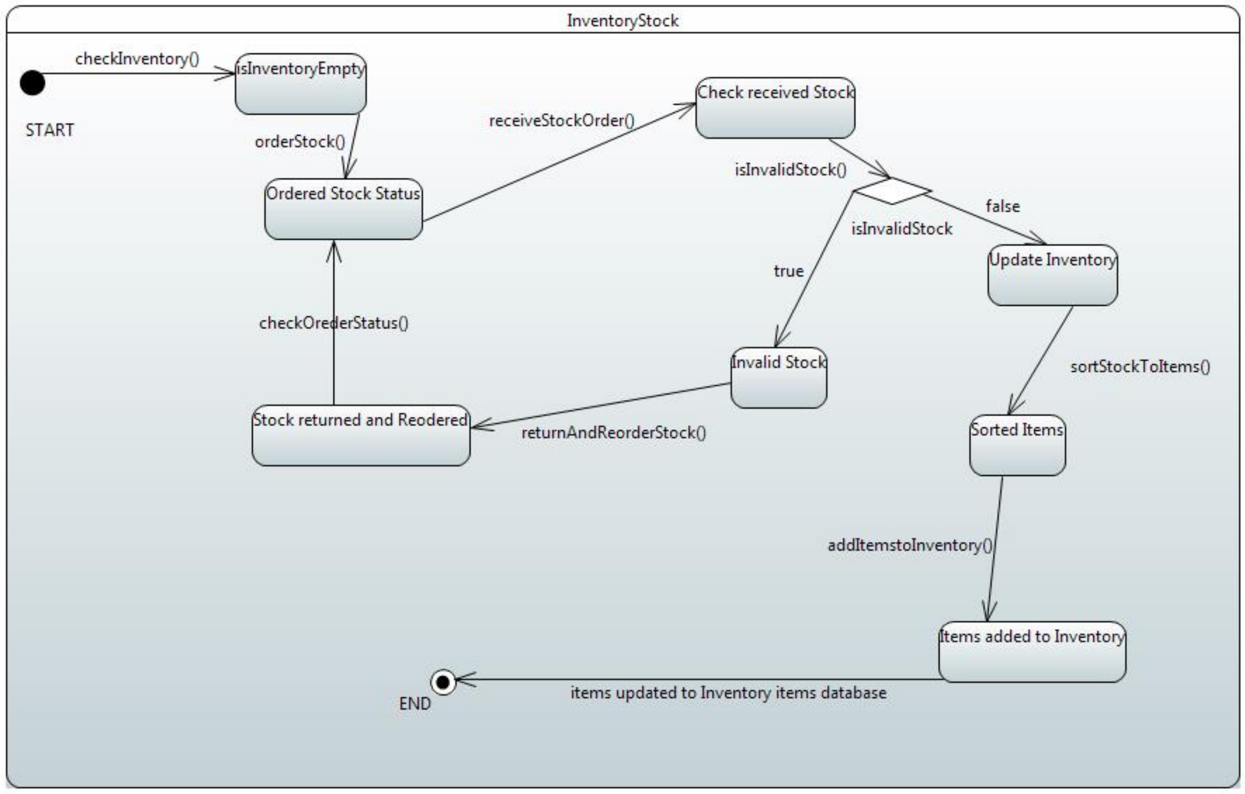
Scenario 1 – In case all the items required for the delivery are available, the delivery is confirmed and dispatched, hence leading to the end of Order Delivery.

Scenario 2 – In case all the items required for Delivery are not available, the delivery is cancelled and the process of Order Delivery ends.

Action Specification Code in Java

```
package com.inventory;
import java.sql.Date;
import java.util.Calendar;
import java.util.Scanner;
public class orderDelivery extends SalesOrder {
static int orderDeliveryNo = 0 ;
static Date orderDate;
static String custId;
static String empId;
static int itemId;
static int orderAmount;
static String shippingAddress;
static Date shippingDate;
static int orderStatus;
static int noOfItems;
public static boolean isItemAvailable(int itemId)
{
	boolean present = false;
	// make present = true if item present
	if(present)
		return true ; // if item available
	else
		return false;
	//code for checking if the item present in inventory or not
}

public static void deliveryDispatch()
{
	//code for dispatching the order
}

public static void createDelivery()
{
	createOrder();
	//code for dispatching the order
}

public static boolean checkItem(int itemId)
{
	boolean itemPresent = false;

	if(isItemAvailable(itemId))
	{
		itemPresent = true;
	}

	return itemPresent;
}

public static void main(String args[])

{

		createDelivery();

	if (checkItem(itemId))
	{
		System.out.println("Delivery Confirmed");
		deliveryDispatch();
	}
	else
	{
		System.out.println("Item not present in Inventory. Delivery 
Cancelled.");
	}
}
}
```

State Diagram for Class Inventory Stock



Explanation

The first step is to check the inventory. After that the system can enter into any of the following three scenarios -

Scenario 1: If the Inventory is low the Stock Manger updates the inventory by ordering the items which are low and inventory stock is updated update functionality.

Scenario 2: Stock which is not available anymore can be removed using the Remove functionality in the system.

Scenario 3: If some new Item is to added into the inventory it is done so by using the Add functionality in the system.

Action Specification Code in Java

```
package com.inventory;
import java.util.Scanner;
public class inventoryStock {
	public static boolean isInventoryEmpty(int product_id)
	{
		boolean notEmpty = false;
		// code for checking Inventory for Stock

		// if stock is not available then empty return = true
		if(notEmpty)
			return true;
		else
			return false;
	}
	public static boolean add_stock(int product_id, String product_qty) {
		// TODO Auto-generated method stub

		boolean success  = false;

		// Code for adding a stock into Inventory
		// if product_id is present then make success = true
		// Perform a check operation If Product id is not available then adding 
a stock as new entry.

		// If stock address successfully then return true otherwise false value

		return success;
	}
	public static boolean update_stock(int product_id, String product_qty) {
		// TODO Auto-generated method stub

		boolean success  = false;

		// Code for updating a stock into Inventory

		// Perform a check operation If Product id is available then Updates an 
existing stock qty.

		// If stock address successfully then return true otherwise false value

		return success;
	}
	public static boolean remove_stock(int product_id) {
		// TODO Auto-generated method stub

		boolean success  = false;

		// Code for removing or deleting a stock from Inventory

		// If stock address successfully then return true otherwise false value

		return success;
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int inven_operation;
		boolean msg = false;
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter operation");
		System.out.println("1 : Add");
		System.out.println("2 : Update");
```

```
            System.out.println("3 : Delete");
            inven_operation = sc.nextInt();
            System.out.println("Enter the Stock Id");
            int product_id = sc.nextInt();

            System.out.println("Enter the Stock Qty");
            String product_qty = sc.nextLine();


            switch(inven_operation)
            {
            case(1):
                  if(isInventoryEmpty(product_id)){
                        msg = add_stock(product_id,product_qty);
                  }else{
                        System.out.println("Stock Already available. Please
perform an Update Operation");
                  }

            case(2):

                  msg = update_stock(product_id,product_qty);

            case(3):

                  msg = remove_stock(product_id);

            if(msg)
                  System.out.println("Operation Performed Successfully");
            else
                  System.out.println("Operation Not Performed Successfully");

            }

      }
}
```

References

[1] Class Notes Dr. Abdel Wahab which is available on Concordia moodle

[2] http://www.sts.tu-harburg.de/teaching/ws-99.00/OOA+D/StateDiagrams.pdf

[3]http://www.omg.org/news/meetings/workshops/presentations/eai_2001/tutorial_monday/tockey_tutorial/6-States,_Actions,_&_Activities.pdf