Project 2: Warehouse Management Interface

Stage 1

CSCI 430 – Object Oriented Programming

Saint Cloud State University

Group 11

Brent Clapp

Group Lead:
Mark Christenson

Sabin Basnet

Table of Contents

Completed Code

```java
/*****************************************************************************
**
OpeningState.java
Provides Interface to chose the users role

    - typeing exit can be used to quit any state
    - higher authorities may log into lower permissions
        + after exiting the user will return to the origional role

Responsible individual: ALL

Options:
    1. Client
    2. Clerk
    3. Manager
    q. quit

*****************************************************************************
**/
import Source_Code.*;
import java.util.*;
import java.io.*;
import java.lang.*;

public class OpeningState {
    final static String MAINMENU =  ""+
            "SELECT STATE          \n\t"+
              "1. Client Menu State\n\t"+
              "2. Clerk Menu State\n\t"+
              "3. Manager Menu State\n\t" +
              "q. Quit the program\n\n";
    final static String FILENAME = "WareData";

  public static void main(String args[]){
      Warehouse warehouse = openWarehouse(FILENAME);
      Scanner s = new Scanner(System.in);
      boolean notDone = true;
      while(notDone){
          System.out.println(MAINMENU);
          String choice = s.nextLine();
          switch(choice){
            case "1":
              ClientMenuState.processInput(warehouse);
              break;
            case "2":
              ClerkMenuState.processInput(warehouse);
              break;
            case "3":
              ManagerMenuState.performMenu(warehouse);
              break;
            case "exit":
            case "q":
                notDone = false;
                break;
            default:
              System.out.println("ERROR: invalid option, exiting program");
          }//end switch
      }//end while
      saveChanges(FILENAME, warehouse);
    }//end main


/******************************************************************************
*
    saveChanges
    Saves any changes made to the warehouse.

******************************************************************************
/
    public static void saveChanges(String file, Warehouse warehouse){
          if(warehouse.saveData(file))
                System.out.println("Saved successfully");
          else
            System.out.println("Save failed. Error occured");
    }//end saveChanges

    /************************************************************
    openWarehouse
    Given a filename, attempts to open the warehouse file
    If not found, then it creates a new warehouse
    Returns a warehouse object.
    ************************************************************/
    private static Warehouse openWarehouse(String file){
          Warehouse w;
          try{
            w = Warehouse.retrieveData(file);
            if(w == null){
                System.out.println("Empty file. Creating new warehouse.");
                w = Warehouse.instance();
            } else
```

```
                System.out.println("Warehouse successfully read from file.");
        } catch(IOException ioe){
            w = Warehouse.instance();
            System.out.println("Warehouse file not found. Creating new warehouse");
        }

        return w;
    }//end openWarehouse
}
```

```
/******************************************************************
ClientMenuState.java
Responsible individual: Sabin Basnet

1.  In the ClientMenuState, the Context has stored the ClientID for the current client; all operations are for that ClientID. The state will have operations for the following:
(a) Show client details. The state invokes a method on Facade to get the Client object and then gets the client details. Note that the ClientID is available in the Context.
(b) Show list of products with sale prices.  The state invokes a method on Facade to get an iterator, and then extracts the needed information.
(c) Show client transactions. The state invokes a method on Facade to get the Client object and then gets the transaction details for the client. Note that the ClientID is available in the Context.
(d) Edit client's shopping cart. Change quantities of products in the shopping cart. Facade provides the iterator.
(e) Add to client's shopping cart. Actor provides the product id and quantity; invoke method on FaÃ§ade.
(f) Display client waitlist.
(g) Logout. System transitions to the previous  state, which has to be remembered in the context. (If previous state was the OpeningState, it goes there; otherwise it goes to ClientMenuState.)
******************************************************************/

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

import Source_Code.*;

public class ClientMenuState{
    private static Warehouse warehouse;
    private static ClerkMenuState clerkMenuState;
  final static String MAINMENU =  ""+
        "CLIENT MENU OPTIONS                                        \n\t"+
          "a. Show Client detail                            (DISPLAYCLIENTDETAILS)\n\t"+
          "b. Show list of products                         (DISPLAYPRODUCTSLIST)\n\t"+
          "c. Show client transactions                      (DISPLAYCLIENTTRANSACTIONS)\n\t"+
          "d. Edit client's shopping cart and change the quantities of product on it (DISPLAYSHOPPINGCART)\n\t"+
          "e. Add client shopping cart                      (ADDSHOPPINGCART)\n\t"+
          "f. Display the client waitlist                   (DISPLAYCLIENTWAITLIST)\n\t"+
          "g. Logout                                        \n\n";


```

```
/*****************************************************************************
*

/*********************************************************************
getProductId
Prompts user for product id, retrieves it and returns it
*********************************************************************/
public static int getProductId(){
   System.out.print("Please enter a product id: ");
   Scanner s = new Scanner(System.in);
   return s.nextInt();
}//end getProductId()


/*****************************************************************************
**
displayClientDetails
Displays all Product objects in the system.

*****************************************************************************
*/
private static void displayClientsDetails(){
   try {
      int clientId = clerkMenuState.getClientId();
      Iterator it = warehouse.getClients();
      while(it.hasNext() )
         System.out.println(it.next().toString());
   }
   catch (Exception e){ System.out.println("ERROR: displayClientDetails() in
ClientMenuState " + e);}
}//end displayAllProducts


/*****************************************************************************
**
displayProductList
Displays all client objects in the system.

*****************************************************************************
*/
private static void displayProductList(){
   try {
      Iterator it = warehouse.getProducts();
      while(it.hasNext() )
         System.out.println(it.next().toString());
   }
```

```
      catch (Exception e){ System.out.println("ERROR: getProducts() in ClientMenuState " + e);}
   }//end displayAllClients


/***************************************************************************
 displayClientTransactions()
 Prompts the user for a client id.
 Asks the user if they'd like to display detailed data (This includes the items
                     that were charged for)
 Displays the date and relevant data for each invoice in the client's history

***************************************************************************/
 private void displayClientTransaction(){
   int clientId = clerkMenuState.getClientId();
   Iterator invoiceIt;
   boolean choice;
   if(!warehouse.verifyClient(clientId)){
     System.out.println("Error, invalid client id. Aborting operation");
     return;
   }//end if
   System.out.print("Would you like to display detailed transactions? (Y|N) ");
   choice = new Scanner(System.in).next().equals("Y"); //true if Y
   invoiceIt = warehouse.getInvoiceIt(clientId);
   if(choice)
     while(invoiceIt.hasNext() )
       System.out.println(((invoiceIt.next())).toString());
   else
     while(invoiceIt.hasNext() )
       System.out.println(((invoiceIt.next())).toString());
 }//end displayInvoices()


/****************************************************************************
 showWaitList
 Gets the product id, then displays its wait list

****************************************************************************/
 private static void showWaitList(){
          try {
       int productID = getProductId();
       Iterator it;
       if(!warehouse.verifyProduct(productID)){
          System.out.println("Error, invalid product id. Aborting operation");
          return;
       }//end if
       it = warehouse.getProductWaitList(productID);
       System.out.println("Product: \n" + warehouse.findProduct(productID).toString());
       if(!it.hasNext())
          System.out.println("Product has no wait list currently");
       else{
          System.out.println("Wait list:\n"+
                        "_________________");
          while(it.hasNext())
            System.out.println(((WaitListItem)it.next()).toString());
       }//end else
       }//end try
       catch (Exception e){ System.out.println("ERROR: showWaitList() in
ClientMenuState " + e);}
 }//end showWaitList



/*****************************************************************************
**
   Editing the Shopping cart
   Add and remove in the shopping cart.

******************************************************************************
*/
   private static void ShopingCart(){
      Iterator it = warehouse.getProducts();
      while(it.hasNext() )
          System.out.println(it.next().toString());
   }//end displayAllClients

 //editing and adding the products in the shopping cart
   private static void displayShoppingCart() {

     Scanner sc = new Scanner(System.in);
      ArrayList<ItemList> cart = new ArrayList<ItemList>();

      ItemList item;
      int itemID;
      String itemName;
      double itemPrice;
      String itemDescription;
      int itemQuantity;
      double itemTax;
      int ch;
```

```
        String choice;

        ProductList shoppingCart = new ProductList();

        while (true) {
            System.out.println("Menu:");
            System.out.println("0) Exit " + "\n"
                    + "1) Add item in shopping cart" + "\n"
                    + "2) Remove item from shpping cart");
            ch = sc.nextInt();

            switch (ch) {
            case 0:
                System.out.println("\n" + "Good bye!");
                System.exit(0);

            case 1:
                System.out.println("Enter item ID: ");
                itemID = sc.nextInt();

                System.out.println("Enter item name: ");
                itemName = sc.next();

                System.out.println("Enter item price: ");
                itemPrice = sc.nextDouble();

                System.out.println("Enter short description of item: ");
                itemDescription = sc.next();

                System.out.println("Enter quantity: ");
                itemQuantity = sc.nextInt();

                System.out.println("Enter tax rate:");
                itemTax = sc.nextDouble();

                shoppingCart.add(itemID, itemName, itemPrice, itemDescription, itemQuantity,
itemTax);

                break;

            case 2:
                System.out.println("Enter name of the item that you would like to remove: ");
                choice = sc.next();
                shoppingCart.remove(choice);

                break;
```

```
          }

      }
  }

public static void processInput(Warehouse warehouse){
      Scanner input = new Scanner(System.in);
      String inputStr = "";
      System.out.println(MAINMENU);
      while(!inputStr.equals("exit") && !inputStr.equals("g")){
          inputStr = input.next();

          switch(inputStr.toUpperCase()){
              case "EXIT":
                  System.out.println("Exiting warehouse operations\n");
        break;
      case "A":
              case "DISPLAYCLIENTDETAILS":
              case "SHOWCLIENTDETAIL  ":
                  displayClientsDetails();
        break;
      case "B":
      case "DISPLAYPRODUCTSLIST":
      case "SHOWLISTOFPRODUCTS":
        displayProductList();
        break;
      case "C":
      case "DISPLAYCLIENTTRANSACTIONS":
      case "SHOWCLIENTTRANSACTIONS":
                  System.out.println("WARNING: Show client transactions unavailable");
        break;
      case "D":
      case "DISPLAYSHOPPINGCART":
      case "EDITCLIENTSSHOPPINGCARTANDCHANGETHEQUANTITIESOFPTODUCTSONIT":
        displayShoppingCart();
        break;
      case "E":
                  System.out.println("WARNING: Add client shopping cart unavailable");
                  break;
      case "F":
      case "Add":
      case "DISPLAYTHECLIENTWAITLIST":
        showWaitList();
        break;
      case "G":
```

```
                    System.out.println("Logging out of client\n");
          break;
        default:
          System.out.print("ERROR: Invalid option\n" + MAINMENU);
          break;
      }//end switch
    }//end while
  }//end processInput
}//end ClientMenuState class
```

```
/*******************************************************************
ClerkMenuState.java
Responsible individual: Mark Christenson

 To run this User Interface element, call the function processInput()

Manages clerk options
 (a) Add A Client (ADDCLIENT)
     Gets details of new client; calls method on FaÃ§ade.
 (b) Show list of products (DISPLAYALLPRODUCTS) with quantities and sale prices.
     The state invokes a method on Facade to get an iterator, and then extracts the needed information.
 (c) Show list of clients (DISPLAYALLCLIENTS)
     The state invokes a method on Facade to get an iterator, and then extracts the needed information.
 (d) Show list of clients with outstanding balance (DISPLAYINVOICES)
     The state invokes a method on Facade to get an iterator, and then extracts the needed information.
 (e) Become a client
     The actor will be asked to input a ClientID; if valid, this ID will be stored in Context, and the system transitions to the  ClientMenuState.
 (f) Display the waitlist for a product (SHOWWAITLIST)
     The state asks the actor for productid; calls method on FaÃ§ade to get an iterator.
 (g) Receive a shipment (ADDSHIPMENT)
     The state asks the actor for productid and quantity; calls method on FaÃ§ade to get an iterator.
     Displays each waitlisted order and performs operation requested by actor (skip or fill).
 (h) Record a payment from a client.
     State asks the actor for ID and amount; calls method on FaÃ§ade to credit the amount to the clientâ€™s account.
 (i) Logout.
     System transitions to the previous  state, which has to be remembered in the context.
     (If previous state was the OpeningState, it goes there; otherwise it goes to ManagerMenuState.)
 *******************************************************************/
import Source_Code.*;
import java.util.*;
import java.io.*;
import java.lang.*;

public class ClerkMenuState{
   private static Warehouse warehouse;
   private static ClerkMenuState ClerkMenuState;
   final static String FILENAME = "WareData";
final static String MAINMENU =  ""+
    "CLERK MENU OPTIONS                                    \n\t"+
     "a. Add A Client                              (ADDCLIENT)\n\t"+
     "b. Show list of products                (DISPLAYALLPRODUCTS)\n\t"+
     "c. Show list of clients                (DISPLAYALLCLIENTS)\n\t"+
     "d. Show list of clients with outstanding balance (DISPLAYINVOICES)\n\t"+
     "e. Become a client                                    \n\t"+
     "f. Display the waitlist for a product            (SHOWWAITLIST)\n\t"+
     "g. Receive a shipment                          (ADDSHIPMENT)\n\t"+
     "h. Record a payment from a client.                        \n\t"+
     "i. Logout                                        \n\n";



/************************* Generic prompt methods *****************************/
  /*** For prompts that are used many times in many applications *****************/

  /*****************************************************************
  getClientId
  Prompts user for client id, retrieves it and returns it
  *****************************************************************/
  public static int getClientId(){
     System.out.print("Please enter a client id: ");
     Scanner s = new Scanner(System.in);
     return s.nextInt();
  }//end getClientId()

  /*****************************************************************
  getProductId
  Prompts user for product id, retrieves it and returns it
  *****************************************************************/
  public static int getProductId(){
     System.out.print("Please enter a product id: ");
     Scanner s = new Scanner(System.in);
     return s.nextInt();
  }//end getProductId()


/************************ File reading methods ********************************/


/*********************************************************************
*
  openWarehouse
```

```
    Opens the given Warehouse, or creates if it doesn't exist
    Returns the Warehouse object

*****************************************************************************
*/
/*  public static void openWarehouse(){
            Warehouse w = Warehouse.retrieveData(FILENAME);
            if(w == null){
                System.out.println("Warehouse not found in file. Creating new Warehouse.");
                warehouse = Warehouse.instance();
            } else{
                System.out.println("Warehouse successfully read from file.");
                warehouse = w;
            }//end else
    }//end openWarehouse
*/

/****************************************************************************
*
    saveChanges
    Saves any changes made to the warehouse.

*****************************************************************************
/
/*  public static void saveChanges(){
        if(warehouse.saveData(FILENAME) )
                System.out.println("Saved successfully");
        else
            System.out.println("Save failed. Error occured");
    }//end saveChanges
 */

/****************************************************************************
*
     instance()
     Called to create an instance of the ClerkMenuState

****************************************************************************/
//  public static void logOut(){saveChanges();}


/****************************************************************************
*
     instance()
     Called to create an instance of the ClerkMenuState

```

```
******************************************************************************/
  public static ClerkMenuState instance() {
    if(ClerkMenuState == null)
      return ClerkMenuState = new ClerkMenuState();
    else
      return ClerkMenuState;
  }//end instance()


/******************************************************************************
*
addClient
Code to prompt user for necessary information to add a new client to the Warehouse

******************************************************************************
*/
private static void addClient(){
  Scanner input = new Scanner(System.in);
  System.out.print("Enter a name for the client: ");
  String name = input.nextLine();
  System.out.print("Enter a phone number for the client: ");
  String phone = input.nextLine();
  System.out.print("Enter an address for the client: ");
  String address = input.nextLine();
  warehouse.addClient(name, phone, address);
  System.out.println("Client added successfully");
}//end addClient


/******************************************************************************
**
  displayAllProducts
  Displays all Product objects in the system.

******************************************************************************
*/
  private static void displayAllProducts(){
    Iterator it = warehouse.getProducts();
    while(it.hasNext() )
      System.out.println(it.next().toString());
  }//end displayAllProducts


/******************************************************************************
**
  displayAllClients
  Displays all client objects in the system.

*****************************************************************************
*/
  private static void displayAllClients(){
      Iterator it = warehouse.getClients();
      while(it.hasNext() )
          System.out.println(it.next().toString());
  }//end displayAllClients


/**************************************************************************
displayInvoices()
Prompts the user for a client id.
Asks the user if they'd like to display detailed data (This includes the items
                        that were charged for)
Displays the date and relevant data for each invoice in the client's history

**************************************************************************/
private static void displayInvoices(){
  int clientId = getClientId();
  Iterator invoiceIt;
  boolean choice;
  if(!warehouse.verifyClient(clientId)){
    System.out.println("Error, invalid client id. Aborting operation");
    return;
  }//end if
  System.out.print("Would you like to display detailed transactions? (Y|N) ");
  choice = new Scanner(System.in).next().equals("Y"); //true if Y
  invoiceIt = warehouse.getInvoiceIt(clientId);
  if(choice)
    while(invoiceIt.hasNext() )
      System.out.println(((Invoice)(invoiceIt.next())).detailedString() );
  else
    while(invoiceIt.hasNext() )
      System.out.println(((Invoice)(invoiceIt.next())).toString());

      System.out.println("WARNING: displayInvoices in ClerkMenuState testing
incomplete");
}//end displayInvoices()


/****************************************************************************
*
showWaitList
Gets the product id, then displays its wait list
```

```
******************************************************************************
/
  private static void showWaitList(){
    int productID = getProductId();
    Iterator it;
    if(!warehouse.verifyProduct(productID)){
      System.out.println("Error, invalid product id. Aborting operation");
      return;
    }//end if
    it = warehouse.getProductWaitList(productID);
    System.out.println("Product: \n" + warehouse.findProduct(productID).toString());
    if(!it.hasNext())
      System.out.println("Product has no wait list currently");
    else{
      System.out.println("Wait list:\n"+
                  "__________________");
      while(it.hasNext())
        System.out.println(((WaitListItem)it.next()).toString());
    }//end else
  }//end showWaitList


/***************************************************************************
   addShipment()
   Will prompt operator for a supplier ID for the shipment being taken in.
   Verifies that id
   Will repeatedly prompt operator for a product id
      Verifies that id
      Prompts for a quantity
      Increases that product's quantity
   Repeats until user enters a sentinel key to exit

****************************************************************************/
   private static void addShipment(){
      int supplierId, productId, quantity, quantCount;
      Scanner scanner;
      boolean moreProducts = true;
      Iterator waitList;
      WaitListItem currItem;
      char choice;
      //Get supplier id:
      System.out.print("Please enter a supplier id: ");
      scanner = new Scanner(System.in);
      supplierId = scanner.nextInt();
      //Verify supplier id:
```

```
    if(!warehouse.verifySupplier(supplierId)){
      System.out.println("Error, invalid supplier id. Aborting operation");
      return;
    }//end if
    while(moreProducts){
      //Get product id:
      System.out.print("Enter the received product's id (0 to quit): ");
      scanner = new Scanner(System.in); //flush input buffer
      productId = scanner.nextInt();
     if(productId == 0) //Sentinel value to return with
      return;
      //Verify product id:
      if(!warehouse.verifyProduct(productId)){
        System.out.println("Error, invalid product id. Aborting operation");
        return;
      }//end if
      //If valid, get product quantity
      System.out.print("Enter a quantity for the product: ");
      scanner = new Scanner(System.in); //flush buffer
      quantity = scanner.nextInt();
      //Increment product's quantity
      waitList = warehouse.addShippedItem(productId, quantity);
      //Receive waitList, prompt for any quantities that can be fulfilled.
      quantCount = 0; //Reset quant count for new item
      while(waitList.hasNext() ){
        currItem = (WaitListItem)waitList.next();
        if((currItem.getQuantity() + quantCount)<= warehouse.getStock(productId)){
          System.out.println("Order " + currItem.getOrder().getId() +
            " can be fulfilled with new stock.\n" +
       " Current stock: " + warehouse.getStock(productId) +
       " Order quantity needed: " + currItem.getQuantity() +
            "\nFulfill? (Y|N) ");
          scanner = new Scanner(System.in);
          choice = scanner.next().charAt(0);
          if(choice == 'Y'){
       System.out.println("Fulfilling order");
            warehouse.fulfillWaitListItem(productId, currItem);
            quantCount += currItem.getQuantity();
          }
        }//end if
      }//end while
      warehouse.doneAddingfulfillItems();
    }//end while(moreProducts)
  }//end addShipment


```

```
/******************************************************************************
callClient
calls the ClientMenuState
*******************************************************************************/
  private static void callClient(){
     ClientMenuState.processInput(warehouse);
     System.out.println(MAINMENU);
  }

public static void processInput(Warehouse w){
     warehouse = w;
     Scanner input = new Scanner(System.in);
     String inputStr = "";
     System.out.println(MAINMENU);
     while(!inputStr.equals("exit") && !inputStr.equals("i") && !inputStr.equals("logout")){
         inputStr = input.next();

         switch(inputStr.toUpperCase()){
             case "EXIT":
                 System.out.println("Exiting Clerk Operations\n");
       break;
     case "A":
             case "ADDCLIENT":
             case "ADDACLIENT":
                 addClient();
       break;
     case "B":
     case "DISPLAYALLPRODUCTS":
     case "SHOWLISTOFPRODUCTS":
       displayAllProducts();
       break;
     case "C":
     case "DISPLAYALLCLIENTS":
     case "SHOWLISTOFCLIENTS":
       displayAllClients();
       break;
     case "D":
     case "DISPLAYINVOICES":
     case "SHOWLISTOFCLIENTSWITHOUTSTANDINGBALANCE":
       displayInvoices();
       break;
```

```
        case "E":
                callClient();
          break;
        case "F":
        case "SHOWWAITLIST":
        case "DISPLAYTHEWAITLISTFORAPRODUCT":
          showWaitList();
          break;
        case "G":
              case "ADDSHIPMENT":
              case "RECEIVEASHIPMENT":
                addShipment();
          break;
        case "H":
                System.out.println("WARNING: Record a payment from a client unavailable");
          break;
        case "I":
          break;
        default:
          System.out.print("ERROR: Invalid option\n" + MAINMENU);
          break;
      }//end switch
    }//end while
  }//end processInput
}//end ClerkMenuState class
```

```
/***********************************************************************
**
ManagerMenuState.java
Contains all options for the managers that are utilizing the UI
Written by: Brent Clapp     Date: 04/01/2020

***********************************************************************
**/
import Source_Code.*;
import java.util.*;
import java.io.*;
import java.lang.*;

public class ManagerMenuState{
    private final static String MENUOPTIONS = "     MANAGER MENU    \n" +
                                "___________________________________________\n" +
                                "Enter a to add a product\n" +
                                "Enter s to add a supplier\n" +
                                "Enter v to view all suppliers\n" +
                                "Enter l to view suppliers of a product\n" +
                                "Enter p to view all products of a supplier\n" +
                                "Enter n to add a supplier for a product\n" +
                                "Enter m to modify the purchase price of a product from a supplier\n"
+
                                "Enter c to login as a salesclerk\n" +
                                "Enter q to logout\n";
    private static Warehouse warehouse;
    private static ManagerMenuState ManagerMenuState;

    /*****************************************************************
    performMenu
    Continuously displays the menu for the user and receives their choice.
    Sends that input to processUserChoice()
    *****************************************************************/
    public static void performMenu(Warehouse w){
        boolean getNextOption = true;
        Scanner s = new Scanner(System.in);
        warehouse = w;
        char choice;
        while(getNextOption){
            //Print the menu
            System.out.println(MENUOPTIONS);
            //Get the user's selection
            choice = s.nextLine().charAt(0);
            //Process that chosen option
```

```
         getNextOption = processUserChoice(choice);
      }//end getNextOption
   }//end performMenu

   private static boolean processUserChoice(char choice){
      boolean iterateAgain = true;
      switch(Character.toLowerCase(choice)){
         case 'a':
            addProduct(); break;
         case 's':
            addSupplier(); break;
         case 'v':
            viewAllSuppliers(); break;
         case 'l':
            viewSuppliersOfProduct(); break;
         case 'p':
            viewProductsOfSupplier(); break;
         case 'n':
            addSupplierForProduct(); break;
         case 'm':
            modifyPurchasePrice(); break;
         case 'c':
            loginAsClerk(); break;
         case 'q':
            System.out.println("Logging out");
            iterateAgain = false; break;
         default:
            System.out.println("Invalid input; Try again");
      }//end switch
      return iterateAgain;
   }//end processUserChoice


/***************************************************************************
   addProduct
   Provides necessary prompts for the user to add a product to the Warehouse
   Allows user to repeat as many times as needed

***************************************************************************/
   private static void addProduct(){
     boolean adding = true;
     Scanner input;
     while(adding){
       input = new Scanner(System.in);
       System.out.print("Enter a description for the product: ");
       String description = input.nextLine();
       System.out.print("Enter an sale price for the product: ");
       String salePrice = input.nextLine();
       System.out.print("Enter a stock for the product: ");
       int stock = input.nextInt();
       warehouse.addProduct(description, Double.valueOf(salePrice), stock);
       System.out.println("Product added successfully");
        //Check to see if they want to add another product
        System.out.print("Add another product? (Y|N) ");
        input = new Scanner(System.in);
        adding = (input.next().charAt(0) == 'Y');
     }//end while
   }//end addProduct


/**************************************************************************
*****
    addSupplier
    Code to prompt user for necessary information to add a new supplier to the warehouse

**************************************************************************
***/
    private static void addSupplier(){
       String description;
       Scanner s = new Scanner(System.in);
       System.out.println("Enter a description for the supplier:");
       description = s.nextLine();
       warehouse.addSupplier(description);
       System.out.println("Supplier added successfully");
    }//end addSupplier

    /*********************************************************************
    viewAllSuppliers
    Displays all suppliers registered in the warehouse
    *********************************************************************/
    private static void viewAllSuppliers(){
       Iterator it = warehouse.getSuppliers();
       System.out.println("Suppliers:");
       System.out.println("_____________________");
       while(it.hasNext())
          System.out.println(it.next().toString());
    }//end viewAllSuppliers

    /*********************************************************************
    viewSuppliersOfProduct
    User passes in the id of the product.
    Interface displays all suppliers that supply that product, along with
```

```
their sale price for the product.
*********************************************************************/
private static void viewSuppliersOfProduct(){
   Scanner s = new Scanner(System.in);
   int productId;
   String output = "";
   System.out.print("Please enter the product id to search for: ");
   productId = s.nextInt();
   if(!warehouse.verifyProduct(productId)){
      System.out.println("No product found with given id. Aborting\n");
      return;
   }//end if
   Iterator it = warehouse.getSuppliers();
   Supplier currSupplier;
   while(it.hasNext()){
      currSupplier = (Supplier)it.next();
      output += currSupplier.searchProduct(productId) + '\n';
   }//end while
   System.out.println(output);
}//end viewSuppliersOfProduct

private static void viewProductsOfSupplier(){
   Scanner s = new Scanner(System.in);
   int supplierId;
   System.out.print("Enter a supplier id: ");
   supplierId = s.nextInt();
   Supplier supp = warehouse.findSupplier(supplierId);
   if(supp == null){
      System.out.println("Error, no supplier found with given id. Aborting.\n");
      return;
   }//end if
   System.out.println("\n" + supp.toString() + "\n" + "____________" + "\n");
   Iterator it = supp.getProducts();
   while(it.hasNext())
      System.out.println(((SupplierItem)it.next()).toString());
}//end viewProductsOfSupplier

/*********************************************************************
addSupplierForProduct
Prompts the user for a supplier id, product id and price. If the supplier
does not already stock the given product, the product will be added to the list
of products that the supplier provides, and will be assigned the given price..
*********************************************************************/
private static void addSupplierForProduct(){
   Scanner s = new Scanner(System.in);
   int supplierId, productId;
```

```
      double price;
      Supplier supplier;
      Product product;
      System.out.print("Enter the id of the supplier who will stock this product: ");
      supplierId = s.nextInt();
      supplier = warehouse.findSupplier(supplierId);
      if(supplier == null){
         System.out.println("Error, no supplier found with given id. Aborting");
         return;
      }//end if
      System.out.print("Enter the id of the product to be added: ");
      productId = s.nextInt();
      product = warehouse.findProduct(productId);
      if(product == null){
         System.out.println("Error, no product found with given id. Aborting");
         return;
      }//end if
      if(supplier.hasProduct(productId)){
         System.out.println("Error, Product already supplied by supplier. Aborting");
         return;
      }//end if
      s = new Scanner(System.in);
      System.out.print("Enter the purchase price for this product: ");
      price = Double.valueOf(s.nextLine());
      supplier.addProduct(product, price);
   }//end addSupplierForProduct

   /****************************************************************
   modifiyPurchasePrice()
   Prompts the user for a supplier id, product id and price. If the supplier
   stocks the given product, this method will reassign the price of the product
   to the new one specified
   ****************************************************************/
   private static void modifyPurchasePrice(){
      Scanner s = new Scanner(System.in);
      int supplierId, productId;
      double price;
      Supplier supplier;
      Product product;
      System.out.print("Enter the id of the supplier who stocks this product: ");
      supplierId = s.nextInt();
      supplier = warehouse.findSupplier(supplierId);
      if(supplier == null){
         System.out.println("Error, no supplier found with given id. Aborting");
         return;
      }//end if
      System.out.print("Enter the id of the product: ");
      productId = s.nextInt();
      product = warehouse.findProduct(productId);
      if(product == null){
         System.out.println("Error, no product found with given id. Aborting");
         return;
      }//end if
      if(!supplier.hasProduct(productId)){
         System.out.println("Error, Product not supplied by supplier. Aborting");
         return;
      }//end if
      s = new Scanner(System.in); //flush input buffer
      System.out.print("Enter the purchase price for this product: ");
      price = Double.valueOf(s.nextLine());
      supplier.setPurchasePrice(product, price);
   }//end modifyPurchasePrice

   private static void loginAsClerk(){
      ClerkMenuState.processInput(warehouse);
   }//end loginAsClerk


/************************************************************************
   instance()
   Used to implement class as singleton. Returns an instance of the ManagerMenuState

*************************************************************************/
   public static ManagerMenuState instance() {
      if(ManagerMenuState == null)
         return ManagerMenuState = new ManagerMenuState();
      else
         return ManagerMenuState;
   }//end instance()
}//end clas
```

Code Distribution

Opening State – ALL
Client Menu State – Sabeen Basnet
Clerk Menu State – Mark Christenson
Manager Menu State – Brent Clapp