

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	2
Opening State	3
Client Menu State	6
Clerk Menu State	13
Manager Menu State	22
 Code Distribution	 28

Completed Code

```

1
2
3 OpeningState.java
4 Provides Interface to chose the users role
5
6 - typeing exit can be used to quit any state
7 - higher authorities may log into lower permissions
8   + after exiting the user will return to the orignal role
9
10 Responsible individual: ALL
11
12 Options:
13   1. Client
14   2. Clerk
15   3. Manager
16   q. quit
17
18 import Source_Code.*;
19 import java.util.*;
20 import java.io.*;
21 import java.lang.*;
22
23 public class OpeningState {
24     final static String MAINMENU = ""+
25         "SELECT STATE      \n\t"+
26         "1. Client Menu State\n\t"+
27         "2. Clerk Menu State\n\t"+
28         "3. Manager Menu State\n\t" +
29         "q. Quit the program\n\n";
30     final static String FILENAME = "WareData";
31
32     public static void main(String args[]){
33         Warehouse warehouse = openWarehouse(FILENAME);
34         Scanner s = new Scanner(System.in);
35         boolean notDone = true;
36         while(notDone){
37             System.out.println(MAINMENU);
38             String choice = s.nextLine();
39             switch(choice){
40                 case "1":
41                     ClientMenuState.processInput(warehouse);
42                     break;

```

```

43     case "2":
44         ClerkMenuState.processInput(warehouse);
45         break;
46     case "3":
47         ManagerMenuState.performMenu(warehouse);
48         break;
49     case "exit":
50     case "q":
51         notDone = false;
52         break;
53     default:
54         System.out.println("ERROR: invalid option, exiting program");
55     } //end switch
56 } //end while
57 saveChanges(FILENAME, warehouse);
58 } //end main
59
60
/*****
*
61  saveChanges
62  Saves any changes made to the warehouse.
63
*****/
/
64  public static void saveChanges(String file, Warehouse warehouse){
65      if(warehouse.saveData(file))
66          System.out.println("Saved successfully");
67      else
68          System.out.println("Save failed. Error occurred");
69  } //end saveChanges
70
71  /*****
72  openWarehouse
73  Given a filename, attempts to open the warehouse file
74  If not found, then it creates a new warehouse
75  Returns a warehouse object.
76  *****/
77  private static Warehouse openWarehouse(String file){
78      Warehouse w;
79      try{
80          w = Warehouse.retrieveData(file);
81          if(w == null){
82              System.out.println("Empty file. Creating new warehouse.");
83              w = Warehouse.instance();
84          } else

```

```
85         System.out.println("Warehouse successfully read from file.");
86     } catch(IOException ioe){
87         w = Warehouse.instance();
88         System.out.println("Warehouse file not found. Creating new warehouse");
89     }
90
91     return w;
92 } //end openWarehouse
93 }
```

```

1  /*****
2  ClientMenuState.java
3  Responsible individual: Sabin Basnet
4
5  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:
6  (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.
7  (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.
8  (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.
9  (d) Edit client's shopping cart. Change quantities of products in the shopping cart.
Facade provides the iterator.
10 (e) Add to client's shopping cart. Actor provides the product id and quantity; invoke
method on Façade.
11 (f) Display client waitlist.
12 (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.)
13 *****/
14
15 import java.util.ArrayList;
16 import java.util.Iterator;
17 import java.util.Scanner;
18
19 import Source_Code.*;
20
21 public class ClientMenuState {
22     private static Warehouse warehouse;
23     private static ClerkMenuState clerkMenuState;
24     final static String MAINMENU = ""+
25         "CLIENT MENU OPTIONS                \n\t"+
26         "a. Show Client detail                (DISPLAYCLIENTDETAILS)\n\t"+
27         "b. Show list of products                (DISPLAYPRODUCTSLIST)\n\t"+
28         "c. Show client transactions            (DISPLAYCLIENTTRANSACTIONS)\n\t"+
29         "d. Edit client's shopping cart and change the quantities of product on it
(DISPLAYSHOPPINGCART)\n\t"+
30         "e. Add client shopping cart                (ADDSHOPPINGCART)\n\t"+
31         "f. Display the client waitlist          (DISPLAYCLIENTWAITLIST)\n\t"+
32         "g. Logout                                \n\n";
33
34

```

```

/*****
*
35
36 /*****
37 getProductId
38 Prompts user for product id, retrieves it and returns it
39 *****/
40 public static int getProductId(){
41     System.out.print("Please enter a product id: ");
42     Scanner s = new Scanner(System.in);
43     return s.nextInt();
44 }//end getProductId()
45
46
/*****
**
47 displayClientDetails
48 Displays all Product objects in the system.
49
*****
*/
50 private static void displayClientsDetails(){
51     try {
52         int clientId = clerkMenuState.getClientId();
53         Iterator it = warehouse.getClients();
54         while(it.hasNext() )
55             System.out.println(it.next().toString());
56     }
57     catch (Exception e){ System.out.println("ERROR: displayClientDetails() in
ClientMenuState " + e);}
58 }//end displayAllProducts
59
60
/*****
**
61 displayProductList
62 Displays all client objects in the system.
63
*****
*/
64 private static void displayProductList(){
65     try {
66         Iterator it = warehouse.getProducts();
67         while(it.hasNext() )
68             System.out.println(it.next().toString());
69     }

```



```

109         return;
110     } //end if
111     it = warehouse.getProductWaitList(productID);
112     System.out.println("Product: \n" + warehouse.findProduct(productID).toString());
113     if(!it.hasNext())
114         System.out.println("Product has no wait list currently");
115     else{
116         System.out.println("Wait list:\n"+
117             "_____");
118         while(it.hasNext())
119             System.out.println(((WaitListItem)it.next()).toString());
120     } //end else
121 } //end try
122 catch (Exception e){ System.out.println("ERROR: showWaitList() in
ClientMenuState " + e);}
123 } //end showWaitList
124
125
126
/*****
**
127  Editing the Shopping cart
128  Add and remove in the shopping cart.
129
*****
*/
130 private static void ShopingCart(){
131     Iterator it = warehouse.getProducts();
132     while(it.hasNext() )
133         System.out.println(it.next().toString());
134 } //end displayAllClients
135
136 //editing and adding the products in the shopping cart
137 private static void displayShoppingCart() {
138
139     Scanner sc = new Scanner(System.in);
140     ArrayList<ItemList> cart = new ArrayList<ItemList>();
141
142     ItemList item;
143     int itemID;
144     String itemName;
145     double itemPrice;
146     String itemDescription;
147     int itemQuantity;
148     double itemTax;
149     int ch;

```

```

150     String choice;
151
152     ProductList shoppingCart = new ProductList();
153
154     while (true) {
155         System.out.println("Menu:");
156         System.out.println("0) Exit " + "\n"
157             + "1) Add item in shopping cart" + "\n"
158             + "2) Remove item from shipping cart");
159         ch = sc.nextInt();
160
161         switch (ch) {
162             case 0:
163                 System.out.println("\n" + "Good bye!");
164                 System.exit(0);
165
166             case 1:
167                 System.out.println("Enter item ID: ");
168                 itemID = sc.nextInt();
169
170                 System.out.println("Enter item name: ");
171                 itemName = sc.next();
172
173                 System.out.println("Enter item price: ");
174                 itemPrice = sc.nextDouble();
175
176                 System.out.println("Enter short description of item: ");
177                 itemDescription = sc.next();
178
179                 System.out.println("Enter quantity: ");
180                 itemQuantity = sc.nextInt();
181
182                 System.out.println("Enter tax rate:");
183                 itemTax = sc.nextDouble();
184
185                 shoppingCart.add(itemID, itemName, itemPrice, itemDescription, itemQuantity,
itemTax);
186
187                 break;
188
189             case 2:
190                 System.out.println("Enter name of the item that you would like to remove: ");
191                 choice = sc.next();
192                 shoppingCart.remove(choice);
193
194                 break;

```

```

195     }
196
197 }
198 }
199
200 public static void processInput(Warehouse warehouse){
201     Scanner input = new Scanner(System.in);
202     String inputStr = "";
203     System.out.println(MAINMENU);
204     while(!inputStr.equals("exit") && !inputStr.equals("g")){
205         inputStr = input.next();
206
207         switch(inputStr.toUpperCase()){
208             case "EXIT":
209                 System.out.println("Exiting warehouse operations\n");
210                 break;
211             case "A":
212                 case "DISPLAYCLIENTDETAILS":
213                 case "SHOWCLIENTDETAIL ":
214                     displayClientsDetails();
215                 break;
216             case "B":
217                 case "DISPLAYPRODUCTSLIST":
218                 case "SHOWLISTOFPRODUCTS":
219                     displayProductList();
220                 break;
221             case "C":
222                 case "DISPLAYCLIENTTRANSACTIONS":
223                 case "SHOWCLIENTTRANSACTIONS":
224                     System.out.println("WARNING: Show client transactions unavailable");
225                 break;
226             case "D":
227                 case "DISPLAYSHOPPINGCART":
228                 case
"EDITCLIENTSSHOPPINGCARTANDCHANGETHQUANTITIESOFPTODUCTSONIT":
229                     displayShoppingCart();
230                 break;
231             case "E":
232                 System.out.println("WARNING: Add client shopping cart unavailable");
233                 break;
234             case "F":
235                 case "Add":
236                 case "DISPLAYTHECLIENTWAITLIST":
237                     showWaitList();
238                 break;
239             case "G":

```

```
240         System.out.println("Logging out of client\n");
241     break;
242 default:
243     System.out.print("ERROR: Invalid option\n" + MAINMENU);
244     break;
245 }//end switch
246 }//end while
247 }//end processInput
248 }//end ClientMenuState class
```

```

1  /*****
2  ClerkMenuState.java
3  Responsible individual: Mark Christenson
4
5  To run this User Interface element, call the function processInput()
6
7  Manages clerk options
8  (a) Add A Client (ADDCLIENT)
9      Gets details of new client; calls method on Façade.
10 (b) Show list of products (DISPLAYALLPRODUCTS) with quantities and sale prices.
11     The state invokes a method on Facade to get an iterator, and then extracts the
needed information.
12 (c) Show list of clients (DISPLAYALLCLIENTS)
13     The state invokes a method on Facade to get an iterator, and then extracts the
needed information.
14 (d) Show list of clients with outstanding balance (DISPLAYINVOICES)
15     The state invokes a method on Facade to get an iterator, and then extracts the
needed information.
16 (e) Become a client
17     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.
18 (f) Display the waitlist for a product (SHOWWAITLIST)
19     The state asks the actor for productid; calls method on Façade to get an iterator.
20 (g) Receive a shipment (ADDSHIPMENT)
21     The state asks the actor for productid and quantity; calls method on Façade to get
an iterator.
22     Displays each waitlisted order and performs operation requested by actor (skip or
fill).
23 (h) Record a payment from a client.
24     State asks the actor for ID and amount; calls method on Façade to credit the
amount to the client's account.
25 (i) Logout.
26     System transitions to the previous state, which has to be remembered in the
context.
27     (If previous state was the OpeningState, it goes there; otherwise it goes to
ManagerMenuState.)
28 *****/
29 import Source_Code.*;
30 import java.util.*;
31 import java.io.*;
32 import java.lang.*;
33
34 public class ClerkMenuState {
35     private static Warehouse warehouse;
36     private static ClerkMenuState ClerkMenuState;
37     final static String FILENAME = "WareData";

```



```

80  Opens the given Warehouse, or creates if it doesn't exist
81  Returns the Warehouse object
82
83  /**
84      Warehouse w = Warehouse.retrieveData(FILENAME);
85      if(w == null){
86          System.out.println("Warehouse not found in file. Creating new Warehouse.");
87          warehouse = Warehouse.instance();
88      } else{
89          System.out.println("Warehouse successfully read from file.");
90          warehouse = w;
91      } //end else
92  } //end openWarehouse
93  */
94
95  saveChanges
96  Saves any changes made to the warehouse.
97
98  /**
99      if(warehouse.saveData(FILENAME) )
100          System.out.println("Saved successfully");
101      else
102          System.out.println("Save failed. Error occurred");
103  } //end saveChanges
104  */
105
106  instance()
107  Called to create an instance of the ClerkMenuState
108
109  // public static void logOut(){saveChanges();}
110
111
112  instance()
113  Called to create an instance of the ClerkMenuState
114

```



```

*****/
115     public static ClerkMenuState instance() {
116         if(ClerkMenuState == null)
117             return ClerkMenuState = new ClerkMenuState();
118         else
119             return ClerkMenuState;
120     }//end instance()
121
122
123     addClient
124     Code to prompt user for necessary information to add a new client to the Warehouse
125
126     private static void addClient(){
127         Scanner input = new Scanner(System.in);
128         System.out.print("Enter a name for the client: ");
129         String name = input.nextLine();
130         System.out.print("Enter a phone number for the client: ");
131         String phone = input.nextLine();
132         System.out.print("Enter an address for the client: ");
133         String address = input.nextLine();
134         warehouse.addClient(name, phone, address);
135         System.out.println("Client added successfully");
136     }//end addClient
137
138
139     displayAllProducts
140     Displays all Product objects in the system.
141
142     private static void displayAllProducts(){
143         Iterator it = warehouse.getProducts();
144         while(it.hasNext() )
145             System.out.println(it.next().toString());
146     }//end displayAllProducts
147
148
149     displayAllClients

```

```

150     Displays all client objects in the system.
151
152     */
153     private static void displayAllClients(){
154         Iterator it = warehouse.getClients();
155         while(it.hasNext() )
156             System.out.println(it.next().toString());
157     }//end displayAllClients
158
159     /**
160     displayInvoices()
161     Prompts the user for a client id.
162     Asks the user if they'd like to display detailed data (This includes the items
163         that were charged for)
164     Displays the date and relevant data for each invoice in the client's history
165     */
166     private static void displayInvoices(){
167         int clientId = getClientId();
168         Iterator invoiceIt;
169         boolean choice;
170         if(!warehouse.verifyClient(clientId)){
171             System.out.println("Error, invalid client id. Aborting operation");
172             return;
173         }//end if
174         System.out.print("Would you like to display detailed transactions? (Y|N) ");
175         choice = new Scanner(System.in).next().equals("Y"); //true if Y
176         invoiceIt = warehouse.getInvoiceIt(clientId);
177         if(choice)
178             while(invoiceIt.hasNext() )
179                 System.out.println(((Invoice)(invoiceIt.next())).detailedString() );
180         else
181             while(invoiceIt.hasNext() )
182                 System.out.println(((Invoice)(invoiceIt.next())).toString());
183         System.out.println("WARNING: displayInvoices in ClerkMenuState testing
incomplete");
184     }//end displayInvoices()
185
186     /**
187     showWaitList
188     Gets the product id, then displays its wait list

```

```

189
*****
/
190 private static void showWaitList(){
191     int productID = getProductID();
192     Iterator it;
193     if(!warehouse.verifyProduct(productID)){
194         System.out.println("Error, invalid product id. Aborting operation");
195         return;
196     }//end if
197     it = warehouse.getProductWaitList(productID);
198     System.out.println("Product: \n" + warehouse.findProduct(productID).toString());
199     if(!it.hasNext())
200         System.out.println("Product has no wait list currently");
201     else{
202         System.out.println("Wait list:\n"+
203             "_____");
204         while(it.hasNext())
205             System.out.println(((WaitListItem)it.next()).toString());
206     }//end else
207 }//end showWaitList
208
209
/*****
210 addShipment()
211 Will prompt operator for a supplier ID for the shipment being taken in.
212 Verifies that id
213 Will repeatedly prompt operator for a product id
214 Verifies that id
215 Prompts for a quantity
216 Increases that product's quantity
217 Repeats until user enters a sentinel key to exit
218
*****/
219 private static void addShipment(){
220     int supplierId, productId, quantity, quantCount;
221     Scanner scanner;
222     boolean moreProducts = true;
223     Iterator waitList;
224     WaitListItem currItem;
225     char choice;
226     //Get supplier id:
227     System.out.print("Please enter a supplier id: ");
228     scanner = new Scanner(System.in);
229     supplierId = scanner.nextInt();
230     //Verify supplier id:

```

```

231     if(!warehouse.verifySupplier(supplierId)){
232         System.out.println("Error, invalid supplier id. Aborting operation");
233         return;
234     }//end if
235     while(moreProducts){
236         //Get product id:
237         System.out.print("Enter the received product's id (0 to quit): ");
238         scanner = new Scanner(System.in); //flush input buffer
239         productId = scanner.nextInt();
240         if(productId == 0) //Sentinel value to return with
241             return;
242         //Verify product id:
243         if(!warehouse.verifyProduct(productId)){
244             System.out.println("Error, invalid product id. Aborting operation");
245             return;
246         }//end if
247         //If valid, get product quantity
248         System.out.print("Enter a quantity for the product: ");
249         scanner = new Scanner(System.in); //flush buffer
250         quantity = scanner.nextInt();
251         //Increment product's quantity
252         waitList = warehouse.addShippedItem(productId, quantity);
253         //Receive waitList, prompt for any quantities that can be fulfilled.
254         quantCount = 0; //Reset quant count for new item
255         while(waitList.hasNext() ){
256             currItem = (WaitListItem)waitList.next();
257             if((currItem.getQuantity() + quantCount)<= warehouse.getStock(productId)){
258                 System.out.println("Order " + currItem.getOrder().getId() +
259                     " can be fulfilled with new stock.\n" +
260                     " Current stock: " + warehouse.getStock(productId) +
261                     " Order quantity needed: " + currItem.getQuantity() +
262                     "\nFulfill? (Y|N) ");
263                 scanner = new Scanner(System.in);
264                 choice = scanner.next().charAt(0);
265                 if(choice == 'Y'){
266                     System.out.println("Fulfilling order");
267                     warehouse.fulfillWaitListItem(productId, currItem);
268                     quantCount += currItem.getQuantity();
269                 }
270             }//end if
271         }//end while
272         warehouse.doneAddingfulfillItems();
273     }//end while(moreProducts)
274 }//end addShipment
275
276

```

```

277
/*****
*
278  callClient
279  calls the ClientMenuState
280
*****/
/
281  private static void callClient(){
282      ClientMenuState.processInput(warehouse);
283      System.out.println(MAINMENU);
284  }
285
286  public static void processInput(Warehouse w){
287      warehouse = w;
288      Scanner input = new Scanner(System.in);
289      String inputStr = "";
290      System.out.println(MAINMENU);
291      while(!inputStr.equals("exit") && !inputStr.equals("i")
&& !inputStr.equals("logout")){
292          inputStr = input.next();
293
294          switch(inputStr.toUpperCase()){
295              case "EXIT":
296                  System.out.println("Exiting Clerk Operations\n");
297                  break;
298              case "A":
299                  case "ADDCLIENT":
300                  case "ADDACLIENT":
301                      addClient();
302                  break;
303              case "B":
304                  case "DISPLAYALLPRODUCTS":
305                  case "SHOWLISTOFPRODUCTS":
306                      displayAllProducts();
307                  break;
308              case "C":
309                  case "DISPLAYALLCLIENTS":
310                  case "SHOWLISTOFCLIENTS":
311                      displayAllClients();
312                  break;
313              case "D":
314                  case "DISPLAYINVOICES":
315                  case "SHOWLISTOFCLIENTSWITHOUTSTANDINGBALANCE":
316                      displayInvoices();
317                  break;

```

```

318     case "E":
319         callClient();
320         break;
321     case "F":
322     case "SHOWWAITLIST":
323     case "DISPLAYTHEWAITLISTFORAPRODUCT":
324         showWaitList();
325         break;
326     case "G":
327         case "ADDSHIPMENT":
328         case "RECEIVEASHIPMENT":
329             addShipment();
330         break;
331     case "H":
332         System.out.println("WARNING: Record a payment from a client
unavailable");
333         break;
334     case "I":
335         break;
336     default:
337         System.out.print("ERROR: Invalid option\n" + MAINMENU);
338         break;
339     } //end switch
340 } //end while
341 } //end processInput
342 } //end ClerkMenuState class

```

```

1
/*****
**
2  ManagerMenuState.java
3  Contains all options for the managers that are utilizing the UI
4  Written by: Brent Clapp   Date: 04/01/2020
5
*****/
6  import Source_Code.*;
7  import java.util.*;
8  import java.io.*;
9  import java.lang.*;
10
11  public class ManagerMenuState {
12      private final static String MENUOPTIONS = "    MANAGER MENU    \n" +
13          "    \n" +
14          "Enter a to add a product\n" +
15          "Enter s to add a supplier\n" +
16          "Enter v to view all suppliers\n" +
17          "Enter l to view suppliers of a product\n" +
18          "Enter p to view all products of a supplier\n" +
19          "Enter n to add a supplier for a product\n" +
20          "Enter m to modify the purchase price of a product from a supplier\n"
21          +
22          "Enter c to login as a salesclerk\n" +
23          "Enter q to logout\n";
24      private static Warehouse warehouse;
25      private static ManagerMenuState ManagerMenuState;
26
27      /*****
28      performMenu
29      Continuously displays the menu for the user and receives their choice.
30      Sends that input to processUserChoice()
31      *****/
32      public static void performMenu(Warehouse w){
33          boolean getNextOption = true;
34          Scanner s = new Scanner(System.in);
35          warehouse = w;
36          char choice;
37          while(getNextOption){
38              //Print the menu
39              System.out.println(MENUOPTIONS);
40              //Get the user's selection
41              choice = s.nextLine().charAt(0);
42              //Process that chosen option

```

```

42     getNextOption = processUserChoice(choice);
43 } //end getNextOption
44 } //end performMenu
45
46 private static boolean processUserChoice(char choice){
47     boolean iterateAgain = true;
48     switch(Character.toLowerCase(choice)){
49         case 'a':
50             addProduct(); break;
51         case 's':
52             addSupplier(); break;
53         case 'v':
54             viewAllSuppliers(); break;
55         case 'l':
56             viewSuppliersOfProduct(); break;
57         case 'p':
58             viewProductsOfSupplier(); break;
59         case 'n':
60             addSupplierForProduct(); break;
61         case 'm':
62             modifyPurchasePrice(); break;
63         case 'c':
64             loginAsClerk(); break;
65         case 'q':
66             System.out.println("Logging out");
67             iterateAgain = false; break;
68         default:
69             System.out.println("Invalid input; Try again");
70     } //end switch
71     return iterateAgain;
72 } //end processUserChoice
73
74
75 /*****
76  addProduct
77  Provides necessary prompts for the user to add a product to the Warehouse
78  Allows user to repeat as many times as needed
79  *****/
80
81 private static void addProduct(){
82     boolean adding = true;
83     Scanner input;
84     while(adding){
85         input = new Scanner(System.in);
86         System.out.print("Enter a description for the product: ");
87         String description = input.nextLine();

```



```

86     System.out.print("Enter an sale price for the product: ");
87     String salePrice = input.nextLine();
88     System.out.print("Enter a stock for the product: ");
89     int stock = input.nextInt();
90     warehouse.addProduct(description, Double.valueOf(salePrice), stock);
91     System.out.println("Product added successfully");
92     //Check to see if they want to add another product
93     System.out.print("Add another product? (Y|N) ");
94     input = new Scanner(System.in);
95     adding = (input.next().charAt(0) == 'Y');
96 }//end while
97 }//end addProduct
98
99
100  addSupplier
101  Code to prompt user for necessary information to add a new supplier to the warehouse
102
103  private static void addSupplier(){
104      String description;
105      Scanner s = new Scanner(System.in);
106      System.out.println("Enter a description for the supplier:");
107      description = s.nextLine();
108      warehouse.addSupplier(description);
109      System.out.println("Supplier added successfully");
110  }//end addSupplier
111
112  viewAllSuppliers
113  Displays all suppliers registered in the warehouse
114  *****/
115  private static void viewAllSuppliers(){
116      Iterator it = warehouse.getSuppliers();
117      System.out.println("Suppliers:");
118      System.out.println("_____");
119      while(it.hasNext())
120          System.out.println(it.next().toString());
121  }//end viewAllSuppliers
122
123  viewSuppliersOfProduct
124  User passes in the id of the product.
125  Interface displays all suppliers that supply that product, along with

```

```

128 their sale price for the product.
129 *****/
130 private static void viewSuppliersOfProduct(){
131     Scanner s = new Scanner(System.in);
132     int productId;
133     String output = "";
134     System.out.print("Please enter the product id to search for: ");
135     productId = s.nextInt();
136     if(!warehouse.verifyProduct(productId)){
137         System.out.println("No product found with given id. Aborting\n");
138         return;
139     }//end if
140     Iterator it = warehouse.getSuppliers();
141     Supplier currSupplier;
142     while(it.hasNext()){
143         currSupplier = (Supplier)it.next();
144         output += currSupplier.searchProduct(productId) + '\n';
145     }//end while
146     System.out.println(output);
147 }//end viewSuppliersOfProduct
148
149 private static void viewProductsOfSupplier(){
150     Scanner s = new Scanner(System.in);
151     int supplierId;
152     System.out.print("Enter a supplier id: ");
153     supplierId = s.nextInt();
154     Supplier supp = warehouse.findSupplier(supplierId);
155     if(supp == null){
156         System.out.println("Error, no supplier found with given id. Aborting.\n");
157         return;
158     }//end if
159     System.out.println("\n" + supp.toString() + "\n" + "_____ " + "\n");
160     Iterator it = supp.getProducts();
161     while(it.hasNext())
162         System.out.println(((SupplierItem)it.next()).toString());
163 }//end viewProductsOfSupplier
164
165 /***/
166 addSupplierForProduct
167 Prompts the user for a supplier id, product id and price. If the supplier
168 does not already stock the given product, the product will be added to the list
169 of products that the supplier provides, and will be assigned the given price..
170 *****/
171 private static void addSupplierForProduct(){
172     Scanner s = new Scanner(System.in);
173     int supplierId, productId;

```

```

174     double price;
175     Supplier supplier;
176     Product product;
177     System.out.print("Enter the id of the supplier who will stock this product: ");
178     supplierId = s.nextInt();
179     supplier = warehouse.findSupplier(supplierId);
180     if(supplier == null){
181         System.out.println("Error, no supplier found with given id. Aborting");
182         return;
183     }//end if
184     System.out.print("Enter the id of the product to be added: ");
185     productId = s.nextInt();
186     product = warehouse.findProduct(productId);
187     if(product == null){
188         System.out.println("Error, no product found with given id. Aborting");
189         return;
190     }//end if
191     if(supplier.hasProduct(productId)){
192         System.out.println("Error, Product already supplied by supplier. Aborting");
193         return;
194     }//end if
195     s = new Scanner(System.in);
196     System.out.print("Enter the purchase price for this product: ");
197     price = Double.valueOf(s.nextLine());
198     supplier.addProduct(product, price);
199 }//end addSupplierForProduct
200
201 /*****
202 modifyPurchasePrice()
203 Prompts the user for a supplier id, product id and price. If the supplier
204 stocks the given product, this method will reassign the price of the product
205 to the new one specified
206 *****/
207 private static void modifyPurchasePrice(){
208     Scanner s = new Scanner(System.in);
209     int supplierId, productId;
210     double price;
211     Supplier supplier;
212     Product product;
213     System.out.print("Enter the id of the supplier who stocks this product: ");
214     supplierId = s.nextInt();
215     supplier = warehouse.findSupplier(supplierId);
216     if(supplier == null){
217         System.out.println("Error, no supplier found with given id. Aborting");
218         return;
219     }//end if

```

```

220     System.out.print("Enter the id of the product: ");
221     productId = s.nextInt();
222     product = warehouse.findProduct(productId);
223     if(product == null){
224         System.out.println("Error, no product found with given id. Aborting");
225         return;
226     }//end if
227     if(!supplier.hasProduct(productId)){
228         System.out.println("Error, Product not supplied by supplier. Aborting");
229         return;
230     }//end if
231     s = new Scanner(System.in); //flush input buffer
232     System.out.print("Enter the purchase price for this product: ");
233     price = Double.valueOf(s.nextLine());
234     supplier.setPurchasePrice(product, price);
235 }//end modifyPurchasePrice
236
237 private static void loginAsClerk(){
238     ClerkMenuState.processInput(warehouse);
239 }//end loginAsClerk
240
241
242 instance()
243 Used to implement class as singleton. Returns an instance of the ManagerMenuState
244
245 *****/
246 public static ManagerMenuState instance() {
247     if(ManagerMenuState == null)
248         return ManagerMenuState = new ManagerMenuState();
249     else
250         return ManagerMenuState;
251 }//end instance()
252 }//end clas

```

Code Distribution

Opening State – ALL

Client Menu State – Sabeen Basnet

Clerk Menu State – Mark Christenson

Manager Menu State – Brent Clapp