Fall 2023
Department of Computer Science
Bahria University, Karachi Campus

OBJECT-ORIENTED PROGRAMMING THEORY

ASSIGNMENT 3

NAME: ALIZA MALIK

ENROLLMENT NO.: 02-134231-029

FIELD: BSCS

MISS SAMEENA JAVED

DATE: 18-12-2023

Bahria University
Discovering Knowledge

Fall 2023
Department of Computer Science
Bahria University, Karachi Campus

ASSIGNMENT No. 3
Complex Computing Problem

Course Title: Object-Oriented Programming
Course Code: CSC-210
Class: BS (CS)-2 A/B　　Submission deadline: 18/12/23
Course Instructor: Dr. Sameena Javaid　　Marks: 10

Instructions

1. Students will perform this assignment individually.
2. Assignment should be done only on A4 size paper and will also be uploaded on LMS
3. Deadline will not be extended for any reason.
4. Copied assignments will be marked zero.
5. Name, class, section, and enrollment number on the sheets must be mentioned correctly
6. Make a single PDF file as both soft and hard copies are mandatory. Submit softcopy on LMS and hardcopy to CR (CR will submit hardcopies to faculty on the due date)

Solution must be designed by applying the following attributes/characteristics

Attribute	Problem Solving Description
Depth of analysis required	Has no obvious solution, and requires conceptual thinking regarding Object Oriented Programming concepts and innovative analysis to formulate suitable abstract models
Depth of knowledge required	A solution requires the use of in-depth computing or domain knowledge of Object-Oriented Programming (OOP) and an analytical approach that is based on well-founded principles of OOP.
Level of problem	Are outside problems encompassed by standards and standard practice for professional computing having above-average skills in Object-Oriented Programming?

Evaluate the given scenario and provide the requirements given after the case:

Case Study / Scenario

The proposed system is an E-Commerce Order Management platform designed to facilitate seamless transactions between customers and the online store. It encompasses various components and functionalities to ensure efficient order processing, inventory management, and customer interaction.

At its core, the system comprises three primary components: Orders, Products, and Customers. The Orders component handles the order placement process by allowing customers to add items, specify details, calculate total prices, cancel orders, and generate invoices. Concurrently, the Products component manages inventory by updating stock quantities whenever orders are placed. The system provides distinct interfaces catering to different user roles. Customers access an interface enabling them to place orders and access their order history. Staff members and administrators, on the other hand, utilize an administrative interface to manage orders, monitor inventory levels, and update customer information. Robust error handling mechanisms are embedded within the system to address potential issues such as out-of-stock products, invalid customer information, and system failures. A key feature involves the use of a robust data storage system (such as a database) to securely persist order, customer, and product information. Security measures are integrated to ensure data protection and system integrity. Authentication and authorization mechanisms are implemented to safeguard customer data and regulate access to sensitive information. The system is designed to handle high volumes of orders efficiently. It incorporates performance optimizations and scalable architecture to accommodate increasing demands without compromising performance. Logging functionalities are put in place to track system events comprehensively. These logs facilitate the generation of reports on order history, customer data, and inventory levels. Additionally, a comprehensive testing suite is developed to validate the system's functionality, ensuring its reliability and robustness. Comprehensive documentation, including class diagrams, flowcharts, and user manuals, is maintained to aid in understanding and maintaining the system. This documentation serves as a valuable resource for both developers and users, facilitating system comprehension and maintenance.

Overall, the system aims to create a seamless and secure environment for managing e-commerce operations. Its emphasis on error handling, scalability, security, and technical documentation ensures a reliable platform for efficient order management and customer satisfaction.

Assumptions:

1. Operations and Methods:
    - Place Order: A method within the Order class that allows a customer to place an order by adding items to the order and specifying the customer details.
    - Calculate Total Price: A method within the Order class that calculates the total price of the order, considering the prices and quantities of the ordered items.
    - Update Stock: A method within the Product class that updates the stock quantity when an order is placed.
    - Cancel Order: A method within the Order class that allows customers to cancel an order, which will update the stock and remove the order from the system.
    - Generate Invoice: A method within the Order class that generates an invoice with details of the order for the customer.
2. User Interfaces:
    - The system may have user interfaces for customers to place orders and view their order history.
    - An administrative interface for staff to manage orders, view inventory, and update customer information.
3. Error Handling:
    - Implement robust error handling to manage issues like out-of-stock products, invalid customer information, or system failures.
4. Data Persistence:
    - Implement a data storage component (such as a database or file system) to persist order, customer, and product information.
5. Security:
    - Ensure that the system is secure, with proper authentication and authorization mechanisms to protect customer data and system integrity.
6. Scalability and Performance:
    - Design the system to handle a large number of orders efficiently, taking into account performance optimizations and scalability options.
7. Logging and Reporting:
    - Implement a logging system to track system events and generate reports on order history, customer data, and inventory levels.
8. Testing:
    - Develop a comprehensive testing suite to validate the system's functionality and robustness.
9. Documentation:
    - Maintain thorough documentation for the system, including class diagrams, flowcharts, and user manuals.

Deliverables / Questions:

Question 01 **(CLO4, PLO4, C5) (5 Marks)**

a. **Determine** the class relationships. Make UML with proper relations among classes.
b. **Recommend** 2 more classes to enhance the vertical scope of the project. With complete justification.

Question 02 **(CLO3, PLO4, C4) (5 Marks)**

a. **Make inferences** using JAVA code of all classes of your modified UML with proper class relationships.
b. Analyze your project by Application Class. Making 2 objects of each class used. Provide Outputs as well.

Note:

Kindly provide a zip folder containing following requirements

1. Java Code (.java file)
2. Proper justification / technical documentation is mandatory.
3. Outputs with 2 cases at least (.docx / .pdf file)
4. UML diagram (.docx / .pdf / any other third party software generated image file)

****

SOURCE CODE:

CLASS MAIN:

```
package complexcomputingtheor3;

import java.util.Scanner;


public class ComplexComputingTheor3 {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      Item Iobj1=new Item(231,"Books",300.0,5);

      Item Iobj2=new Item(322,"Coins",200.0,4);

      Item Iobj3=new Item(972,"Bags",500.0,3);

      String move="";

      System.out.println("Are you from administrative(Staff/Administrator)? (yes/no/exit): ");

      String admin = sc.nextLine();

      Staff staff = new Staff("aliza","malik");

      Staff staff2 = new Staff("aliza","malik");

      if (admin.equalsIgnoreCase("yes")) {

      System.out.print("Enter username: ");

      String enteredUsername = sc.nextLine();

      System.out.print("Enter password: ");

      String enteredPassword = sc.nextLine();

         if(staff.login(enteredUsername, enteredPassword)){

            staff.monitorInventory();

            System.out.println("-----------------  STOCK DETAILS ---------------");
```

```
Iobj1.displayItemDetails();

Iobj2.displayItemDetails();

Iobj3.displayItemDetails();

int again;

do{

System.out.print("Enter item id to be updated =  ");

int updatedid = sc.nextInt();

if(updatedid==231){

staff.updateStockByAdmin(Iobj1);

Iobj1.displayItemDetails();

}

else if(updatedid==322){

staff.updateStockByAdmin(Iobj2);

Iobj2.displayItemDetails();

}

else if(updatedid==972){

staff.updateStockByAdmin(Iobj3);

Iobj3.displayItemDetails();

}

else{

System.out.println("Invalid item ");

}

System.out.println("Would you update any other stock?  yes=1/no=2 ");

again = sc.nextInt();
```

```
        }while(again==1);

        }

System.out.println("Would you like to move to Customer Side? (yes/no/exit): ");

move = sc.next();

    }

if (admin.equalsIgnoreCase("no") || move.equalsIgnoreCase("yes"))   {

Order orders = new Order();

Order order2 = new Order();

Customer cus1 = new Customer();

Customer cus2 = new Customer();

cus1.inputinfo();

System.out.println("-----------------  ITEM DETAILS ---------------");

Iobj1.displayItemDetails();

Iobj2.displayItemDetails();

Iobj3.displayItemDetails();

System.out.println();

cus1.displayCustomerDetails();

System.out.println();

System.out.println("Do you want to place order?  yes=1/no=2");

int choice = sc.nextInt();

while(choice==1){

System.out.println("Write Item ID of product you want to place in cart : ");

int itemid = sc.nextInt();

if(itemid==231){
```

```
orders.addItem(Iobj1);

}

else if(itemid==322){

orders.addItem(Iobj2);

}

else if(itemid==972){

orders.addItem(Iobj3);

}

else{

System.out.println("Invalid item ");

}

System.out.println("Do you want to place other order?  yes=1/no=2");

choice = sc.nextInt();

}

orders.orderTotalPrice();

System.out.println("\n-----------------------------------------------");

System.out.println("---------------- ORDER HISTORY -----------------");

cus1.displayOrderHistory(orders, "order_history.txt");

System.out.println("\n--------- READ ORDER HISTORY FROM FILE ---------");

System.out.println("Reading order history from file : ");

cus1.readOrderHistory("order_history.txt");

System.out.println("-----------------------------------------------");

System.out.println("Do you want to cancel order? yes=1/no=2");

int cancel = sc.nextInt();
```

```
while(cancel==1){

System.out.println("Write Item ID of product you want to cancel : ");

int itemid = sc.nextInt();

if(itemid==231){

orders.cancelOrder(Iobj1);

}

else if(itemid==322){

orders.cancelOrder(Iobj2);

}

else if(itemid==972){

orders.cancelOrder(Iobj3);

}

else{

System.out.println("Invalid item ");

}

System.out.println("Do you want to cancel other order?  yes=1/no=2");

cancel = sc.nextInt();

}

orders.orderTotalPrice();

System.out.println("\n----------------------------------------------");

System.out.println("\n--------- AFTER CANCELLING AN ORDER -----------");

orders.generateInvoice();

orders.orderTotalPrice();

System.out.println("\n\n----------------------------------------------");
```

```
        System.out.println("-------------------- STOCK ---------------------");
        orders.updatingStock();
        System.out.println("\n\nDo you want to confirm order? yes=1/no=2");
        int a = sc.nextInt();
        if(a==1){
          Payments pay1 = new Payments(101,"Done !");
          pay1.processPaymentsDetails(cus1,orders);
        }
        else{
          System.out.println("Thank You for visiting");
          }
        }
      }
    }
```

CLASS CUSTOMER:

```
package complexcomputingtheor3;


import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
public class Customer {
   private int CustomerID;
   private String CustomerName;
   private String CustomerEmail;
   public Customer(){
      this.CustomerID=CustomerID;
      this.CustomerName=CustomerName;
      this.CustomerEmail=CustomerEmail;
   }
   public void inputinfo(){
      Scanner sc = new Scanner(System.in);
      System.out.println("---- CUSTOMER SIDE ----");
      System.out.print("ID = ");
      CustomerID = sc.nextInt();
      System.out.print("Name = ");
```

```
        CustomerName = sc.next();

        System.out.print("Email = ");

        CustomerEmail= sc.next();

    }

    public void displayOrderHistory(Order order, String myFile) {

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(myFile))) {

            for (Item item : order.getOrderList()) {

                writer.write("Order Name = " + item.getproductName() + "\nOrder ID = " +
item.getproductID()+ "\nOrder Price = " + item.getproductPrice()+"\n");

                writer.newLine();

            }

            System.out.println("Order history has been written to the file : " + myFile);

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    public String readOrderHistory(String myFile) {

        try (BufferedReader reader = new BufferedReader(new FileReader(myFile))) {

            String line;

            while ((line = reader.readLine()) != null) {

                System.out.println(line);

            }

        } catch (IOException e) {

            e.printStackTrace();

        }
        return "";
    }
    public int getCustomerID(){
        return CustomerID;
    }
    public String getCustomerName(){
        return CustomerName;
    }
    public String getCustomerEmail(){
        return CustomerEmail;
    }
    public void displayCustomerDetails(){
        System.out.println("-------------- CUSTOMER DETAILS --------------");
        System.out.println("Customer ID = "+CustomerID);
        System.out.println("Customer Name = "+CustomerName);
        System.out.println("Customer Email = "+CustomerEmail);
    }
}
```

CLASS ORDER:

```
package complexcomputingtheor3;

import java.util.ArrayList;


public class Order {

   private ArrayList<Item> itemList = new ArrayList<>();

   public void addItem(Item e) {

      try {

         if (e.getitemQuantity() > 0) {

            itemList.add(e);

            System.out.println("Item added to the order = " + e.getproductName());

         } else {

            throw new OutOfStockException("Item " + e.getproductName() + " is out of
stock.");

         }

      } catch (OutOfStockException ex) {

         System.out.println("Error = " + ex.getMessage());

      }

   }

   public void cancelOrder(Item o) {

      itemList.remove(o);

      System.out.println("Item removed from the order = " + o.getproductName());

   }

   public String orderTotalPrice() {

      int total = 0;
```

```
    for (Item item : itemList) {
        total += item.getproductPrice();
    }
    System.out.print("TOTAL ORDER PRICE = " + total);
    return "";
}
public ArrayList<Item> getOrderList() {
    return itemList;
}
public String generateInvoice() {
    System.out.println("---------- ORDERED ITEM DETAILS ---------------");

    for (int i = 0; i < itemList.size(); i++) {
        Item item = itemList.get(i);
        System.out.println(item.displayItemDetails());
    }
    return "";
}
private static class OutOfStockException extends Exception {
    public OutOfStockException(String message) {
        super(message);
    }
}
public void updatingStock() {
        try {
            for (Item item : itemList) {
                if (item.getitemQuantity() >=1) {
                    item.decreaseStock();
                    System.out.println("Stock updated for item = " + item.getproductName());
                } else {
                    throw new OutOfStockException("Item " + item.getproductName() + " is out of
stock.");
                }
            }
        } catch (OutOfStockException ex) {
            System.out.println("Error = " + ex.getMessage());
        }
    }
}
```

CLASS ITEM:

```
package complexcomputingtheor3;

public class Item extends Product{

   private int itemQuantity;



   public Item(int productID, String productName,double productPrice,int itemQuantity){

      super(productID, productName, productPrice);

      this.itemQuantity=itemQuantity;

   }

   public int getitemQuantity(){

      return itemQuantity;

   }

    public void setitemQuantity(int itemQuantity) {

      this.itemQuantity = itemQuantity;

   }

   public double calculatePrice(){

      return getproductPrice()*itemQuantity;

   }

   public void decreaseStock(){

      itemQuantity--;

   }

   public String displayItemDetails(){

      System.out.println("Item ID = "+getproductID());

      System.out.println("Item Name = "+getproductName());
```

```
        System.out.println("Item Quantities available = "+itemQuantity);

        System.out.println("Their Price = "+calculatePrice());

        System.out.println("---------------------------------------------");

        return "";

    }

}
```

CLASS PRODUCT:

```
package complexcomputingtheor3;

public class Product {

   private int productID;

   private String productName;

   private double productPrice;


   public Product(int productID, String productName,double productPrice){

      this.productID=productID;

      this.productName=productName;

      this.productPrice=productPrice;

   }

   public double getproductPrice(){

      return productPrice;

   }

   public String getproductName(){

      return productName;

   }

   public int getproductID(){

      return productID;

   }

   public void displayProductDetails(){

      System.out.println("----- PRODUCT DETAILS -----");

      System.out.println("Product ID = "+productID);
```

```
        System.out.println("Product Name = "+productName);

        System.out.println("Product Price = "+productPrice);

    }

}
```

CLASS PAYMENT:

```
package complexcomputingtheor3;

public class Payments {

   private int id;

   private String status;



   public Payments(int id, String status){

      this.id=id;

      this.status=status;

   }

   public int getPaymentsID(){

      return id;

   }

   public String getPaymentsStatuc(){

      return status;

   }

   public void processPaymentsDetails(Customer customer, Order order){


System.out.println("\n***********************************************\n*******
************ BILL ***********************");

      System.out.println("Payment ID = "+id);

      System.out.println("Payment Status = "+status);

      System.out.println("\nCustomer Name = "+customer.getCustomerName());

      System.out.println("Customer ID = "+customer.getCustomerID());

      System.out.println("Customer Email = "+customer.getCustomerEmail()+"\n");
```

```
        System.out.print(customer.readOrderHistory("order_history.txt"));

        System.out.println("-----------------------------------------------");

        System.out.println(order.orderTotalPrice());

        System.out.println("-----------------------------------------------");

    }

}
```

CLASS ADMINISTRATIVE:

```
package complexcomputingtheor3;

public interface Adminstrative {

   void manageOrder();

   void monitorInventory();

   void updateCustomerInfo();

}
```

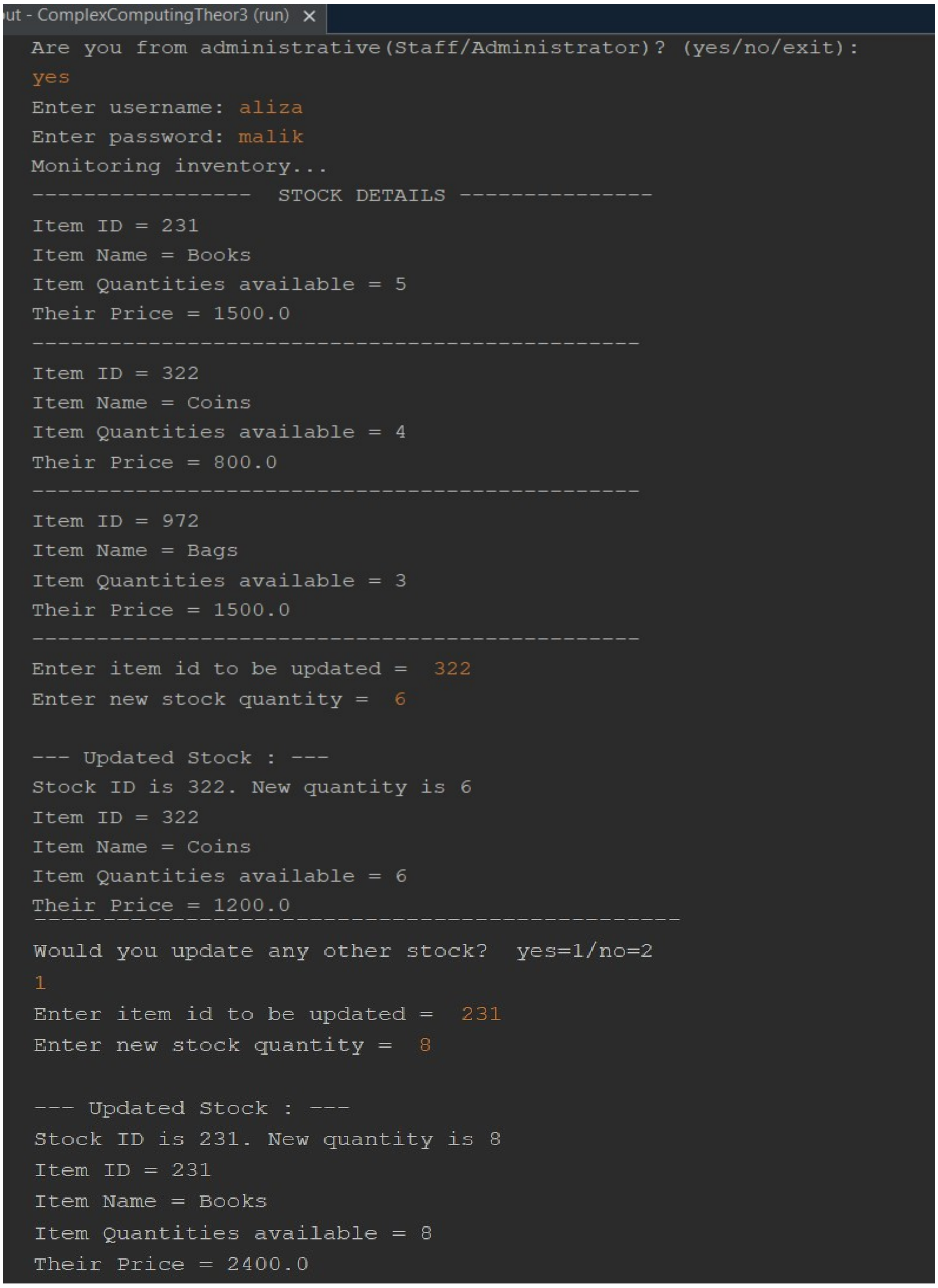
CLASS ADMINISTRATOR:

```
package complexcomputingtheor3;

import java.util.Scanner;

public class Administrator implements Adminstrative{

@Override

   public void manageOrder(){

    System.out.println("Managing orders...");

   }

   @Override

   public void monitorInventory(){

     System.out.println("Monitoring inventory...");

   }

   @Override

   public void updateCustomerInfo(){

    System.out.println("Updating customer information...");

   }

 }
```

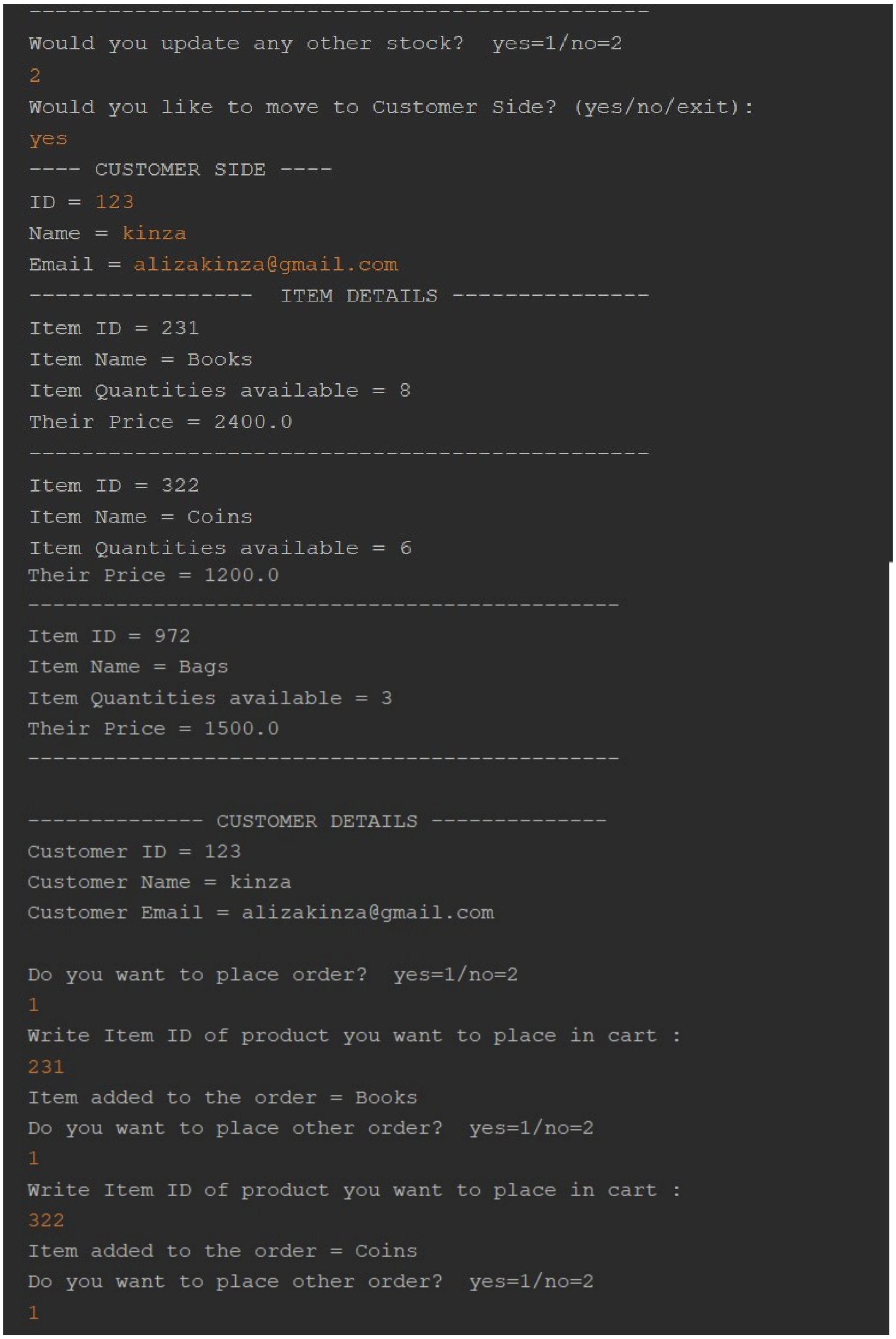
CLASS STAFF:

```
package complexcomputingtheor3;

import java.util.Scanner;


public class Staff implements Adminstrative{

    private String username;

    private String password;

     Scanner sc = new Scanner(System.in);

    public Staff(String username, String password) {

        this.username = username;

        this.password = password;

    }

    @Override

    public void manageOrder(){

     System.out.println("Managing orders...");

    }

    @Override

    public void monitorInventory(){

      System.out.println("Monitoring inventory...");

    }

    @Override

    public void updateCustomerInfo(){

     System.out.println("Updating customer information...");

    }
    public boolean authenticate(String enteredUsername, String enteredPassword) {

        return enteredUsername.equals(username) && enteredPassword.equals(password);

    }

    public void updateStockByAdmin(Item item) {

        System.out.print("Enter new stock quantity =  ");

        int newStock = sc.nextInt();

        item.setitemQuantity(newStock);

        System.out.println("\n--- Updated Stock : --- \nStock ID is " + item.getproductID() + ".
New quantity is " + newStock);

    }

}
```

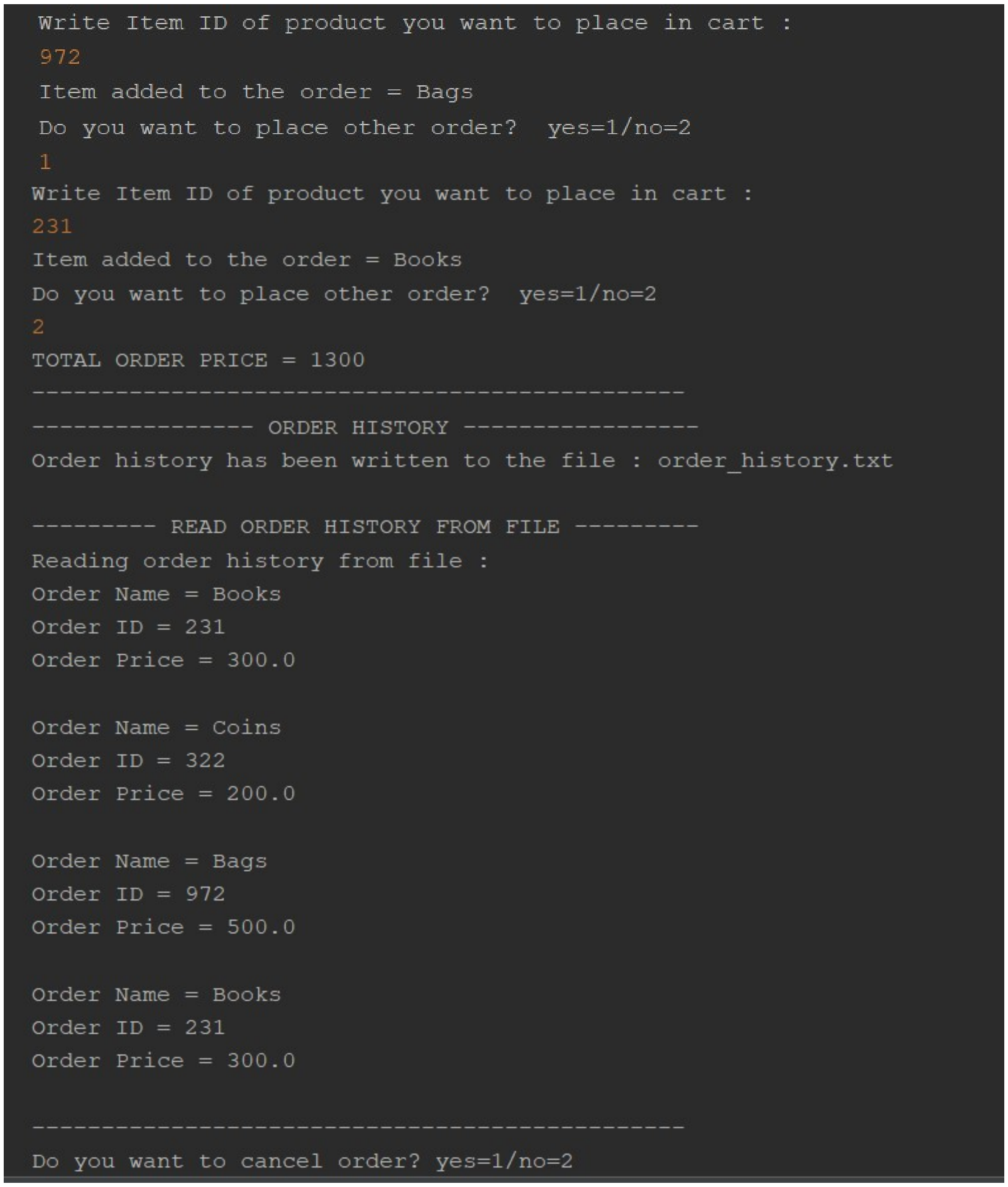
DETAILED OUTPUT FOR STAFF AND CUSTOMER :


ut - ComplexComputingTheor3 (run)

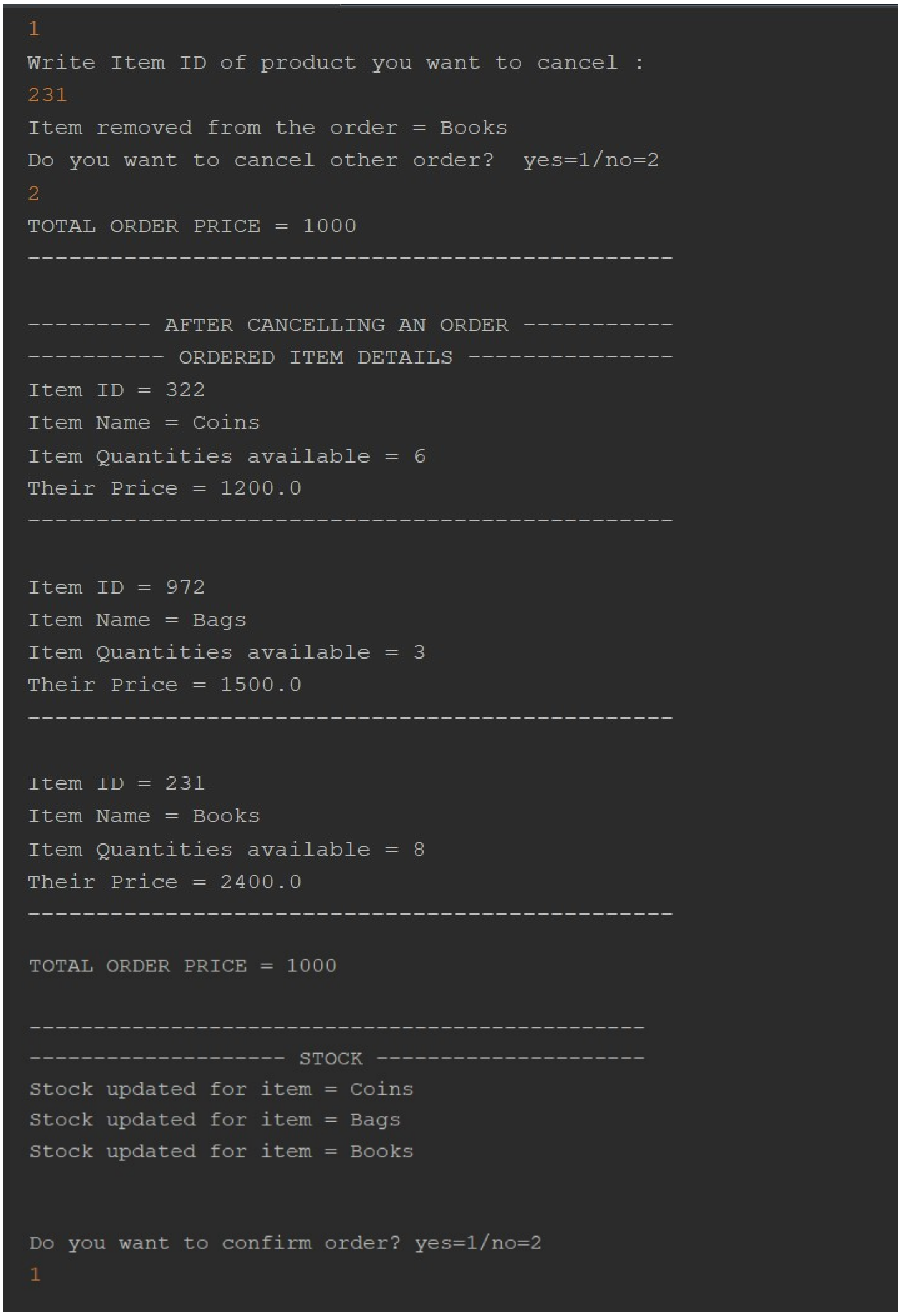
```
Are you from administrative(Staff/Administrator)? (yes/no/exit):
yes
Enter username: aliza
Enter password: malik
Monitoring inventory...
-----------------  STOCK DETAILS ---------------
Item ID = 231
Item Name = Books
Item Quantities available = 5
Their Price = 1500.0
-----------------------------------------------
Item ID = 322
Item Name = Coins
Item Quantities available = 4
Their Price = 800.0
-----------------------------------------------
Item ID = 972
Item Name = Bags
Item Quantities available = 3
Their Price = 1500.0
-----------------------------------------------
Enter item id to be updated =  322
Enter new stock quantity =  6

--- Updated Stock : ---
Stock ID is 322. New quantity is 6
Item ID = 322
Item Name = Coins
Item Quantities available = 6
Their Price = 1200.0
-----------------------------------------------
Would you update any other stock?  yes=1/no=2
1
Enter item id to be updated =  231
Enter new stock quantity =  8

--- Updated Stock : ---
Stock ID is 231. New quantity is 8
Item ID = 231
Item Name = Books
Item Quantities available = 8
Their Price = 2400.0
```



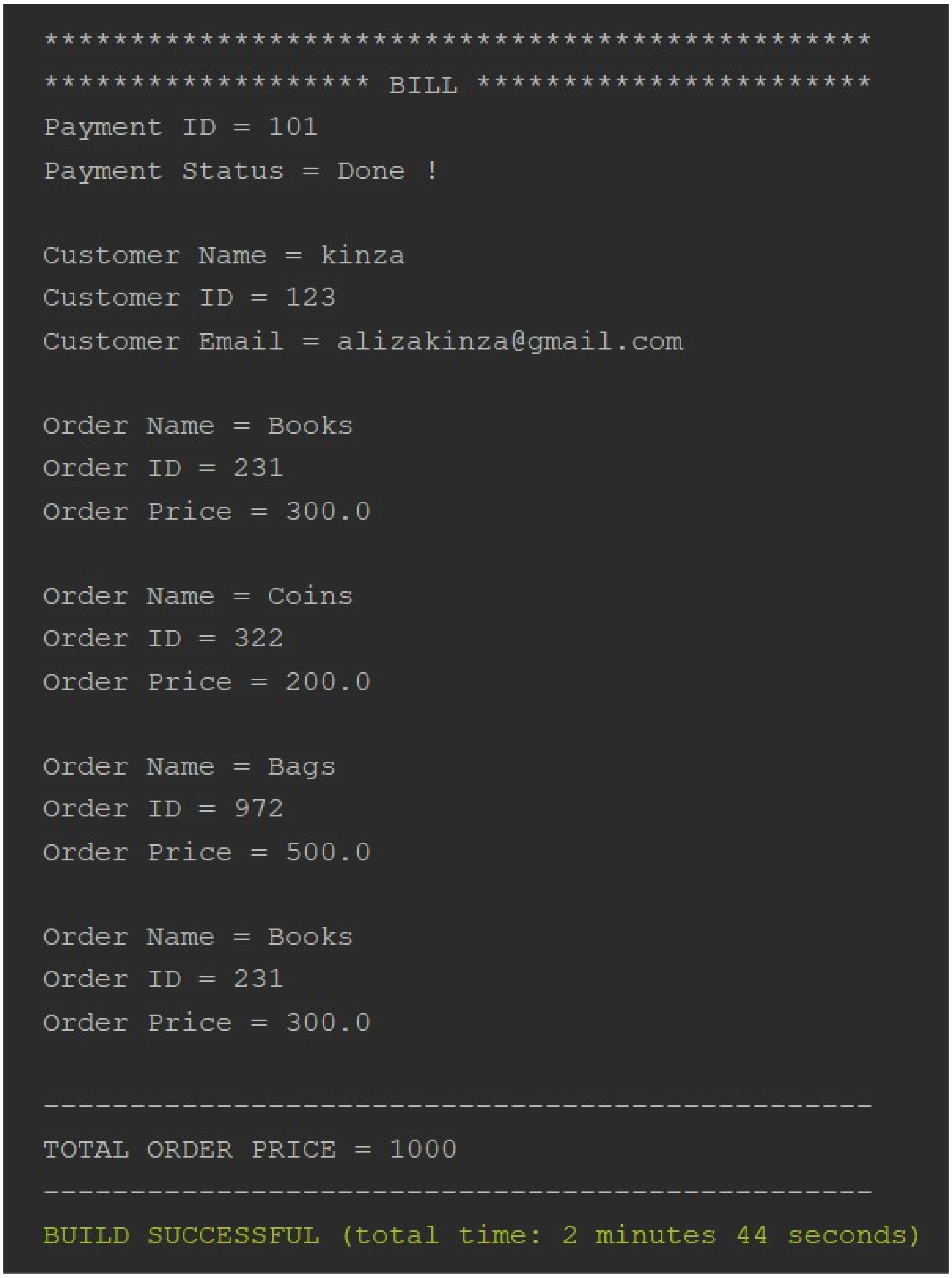
```
-----------------------------------------------
Would you update any other stock?  yes=1/no=2
2
Would you like to move to Customer Side? (yes/no/exit):
yes
---- CUSTOMER SIDE ----
ID = 123
Name = kinza
Email = alizakinza@gmail.com
-----------------  ITEM DETAILS ---------------
Item ID = 231
Item Name = Books
Item Quantities available = 8
Their Price = 2400.0
-----------------------------------------------
Item ID = 322
Item Name = Coins
Item Quantities available = 6
Their Price = 1200.0
-----------------------------------------------
Item ID = 972
Item Name = Bags
Item Quantities available = 3
Their Price = 1500.0
-----------------------------------------------


-------------- CUSTOMER DETAILS --------------
Customer ID = 123
Customer Name = kinza
Customer Email = alizakinza@gmail.com

Do you want to place order?  yes=1/no=2
1
Write Item ID of product you want to place in cart :
231
Item added to the order = Books
Do you want to place other order?  yes=1/no=2
1
Write Item ID of product you want to place in cart :
322
Item added to the order = Coins
Do you want to place other order?  yes=1/no=2
1
```



```
Write Item ID of product you want to place in cart :
972
Item added to the order = Bags
Do you want to place other order?  yes=1/no=2
1
Write Item ID of product you want to place in cart :
231
Item added to the order = Books
Do you want to place other order?  yes=1/no=2
2
TOTAL ORDER PRICE = 1300
-----------------------------------------------
---------------- ORDER HISTORY -----------------
Order history has been written to the file : order_history.txt

--------- READ ORDER HISTORY FROM FILE ---------
Reading order history from file :
Order Name = Books
Order ID = 231
Order Price = 300.0

Order Name = Coins
Order ID = 322
Order Price = 200.0

Order Name = Bags
Order ID = 972
Order Price = 500.0

Order Name = Books
Order ID = 231
Order Price = 300.0

-----------------------------------------------
Do you want to cancel order? yes=1/no=2
```

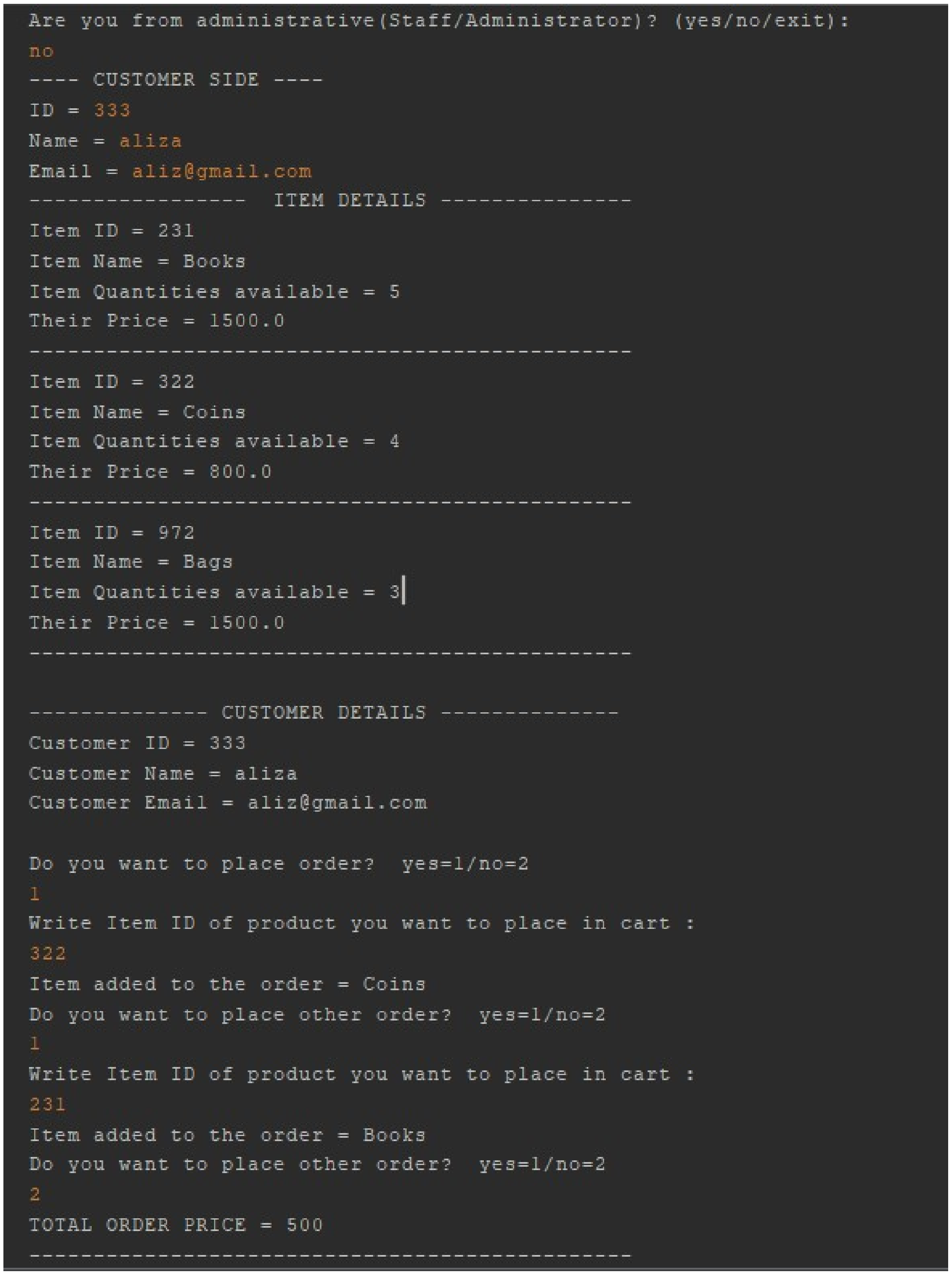


```
1
Write Item ID of product you want to cancel :
231
Item removed from the order = Books
Do you want to cancel other order?  yes=1/no=2
2
TOTAL ORDER PRICE = 1000
-----------------------------------------------


--------- AFTER CANCELLING AN ORDER -----------
---------- ORDERED ITEM DETAILS ---------------
Item ID = 322
Item Name = Coins
Item Quantities available = 6
Their Price = 1200.0
-----------------------------------------------


Item ID = 972
Item Name = Bags
Item Quantities available = 3
Their Price = 1500.0
-----------------------------------------------


Item ID = 231
Item Name = Books
Item Quantities available = 8
Their Price = 2400.0
-----------------------------------------------

TOTAL ORDER PRICE = 1000

---------------------------------------------
-------------------- STOCK -------------------
Stock updated for item = Coins
Stock updated for item = Bags
Stock updated for item = Books


Do you want to confirm order? yes=1/no=2
1
```

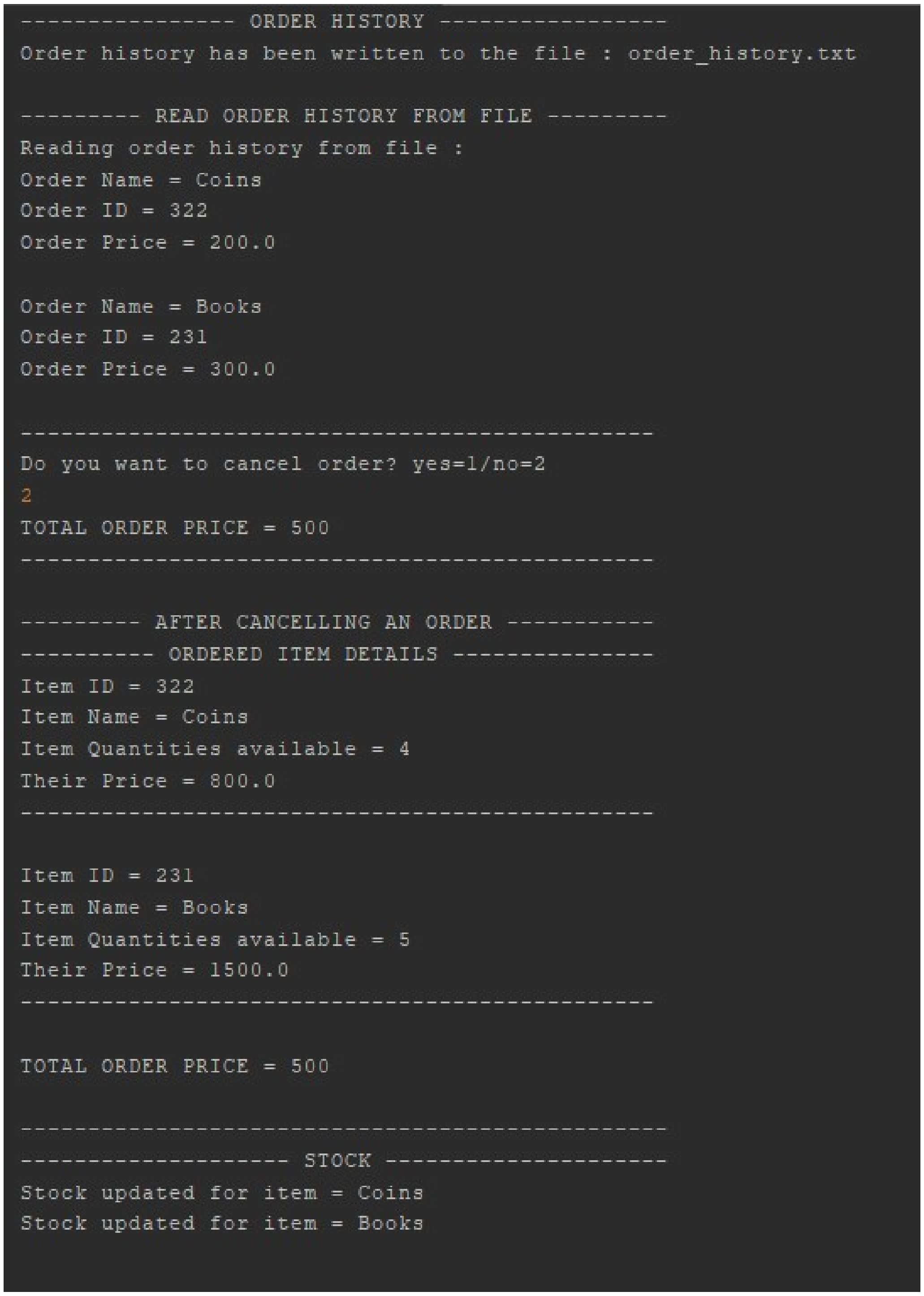


```
***********************************************
****************** BILL **********************
Payment ID = 101
Payment Status = Done !

Customer Name = kinza
Customer ID = 123
Customer Email = alizakinza@gmail.com

Order Name = Books
Order ID = 231
Order Price = 300.0

Order Name = Coins
Order ID = 322
Order Price = 200.0

Order Name = Bags
Order ID = 972
Order Price = 500.0

Order Name = Books
Order ID = 231
Order Price = 300.0

-----------------------------------------------
TOTAL ORDER PRICE = 1000
-----------------------------------------------
BUILD SUCCESSFUL (total time: 2 minutes 44 seconds)
```

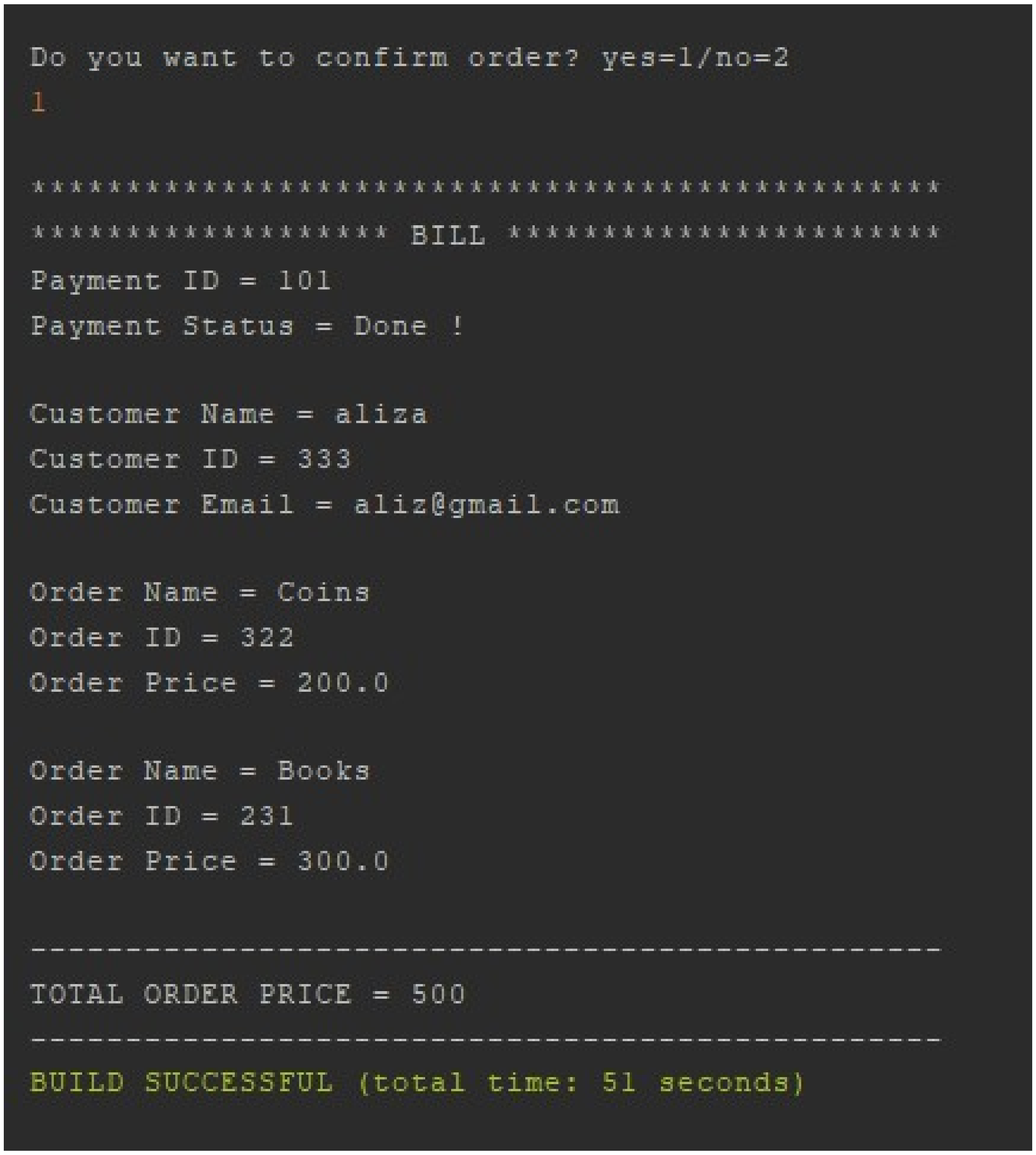
BRIEF OUTPUT FOR CASE 2:



```
Are you from administrative(Staff/Administrator)? (yes/no/exit):
no
---- CUSTOMER SIDE ----
ID = 333
Name = aliza
Email = aliz@gmail.com
-----------------  ITEM DETAILS ---------------
Item ID = 231
Item Name = Books
Item Quantities available = 5
Their Price = 1500.0
-----------------------------------------------
Item ID = 322
Item Name = Coins
Item Quantities available = 4
Their Price = 800.0
-----------------------------------------------
Item ID = 972
Item Name = Bags
Item Quantities available = 3
Their Price = 1500.0
-----------------------------------------------

-------------- CUSTOMER DETAILS --------------
Customer ID = 333
Customer Name = aliza
Customer Email = aliz@gmail.com

Do you want to place order?  yes=1/no=2
1
Write Item ID of product you want to place in cart :
322
Item added to the order = Coins
Do you want to place other order?  yes=1/no=2
1
Write Item ID of product you want to place in cart :
231
Item added to the order = Books
Do you want to place other order?  yes=1/no=2
2
TOTAL ORDER PRICE = 500
-----------------------------------------------
```

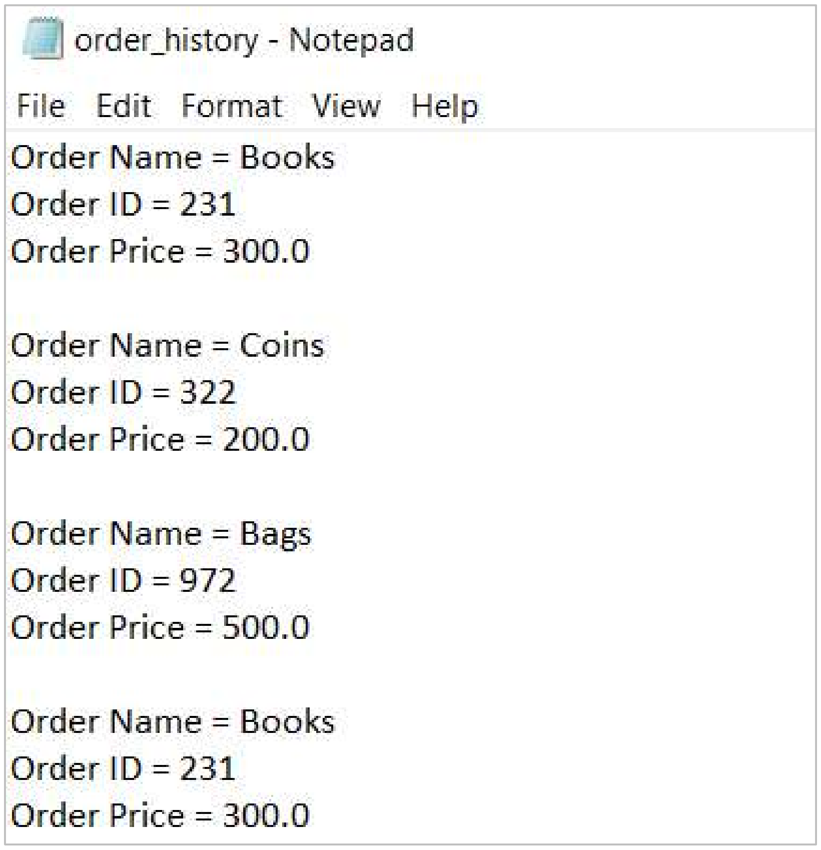


```
---------------- ORDER HISTORY -----------------
Order history has been written to the file : order_history.txt

--------- READ ORDER HISTORY FROM FILE ---------
Reading order history from file :
Order Name = Coins
Order ID = 322
Order Price = 200.0

Order Name = Books
Order ID = 231
Order Price = 300.0

-----------------------------------------------
Do you want to cancel order? yes=1/no=2
2
TOTAL ORDER PRICE = 500
-----------------------------------------------

--------- AFTER CANCELLING AN ORDER -----------
---------- ORDERED ITEM DETAILS ---------------
Item ID = 322
Item Name = Coins
Item Quantities available = 4
Their Price = 800.0
-----------------------------------------------

Item ID = 231
Item Name = Books
Item Quantities available = 5
Their Price = 1500.0
-----------------------------------------------

TOTAL ORDER PRICE = 500

-----------------------------------------------
-------------------- STOCK ---------------------
Stock updated for item = Coins
Stock updated for item = Books
```

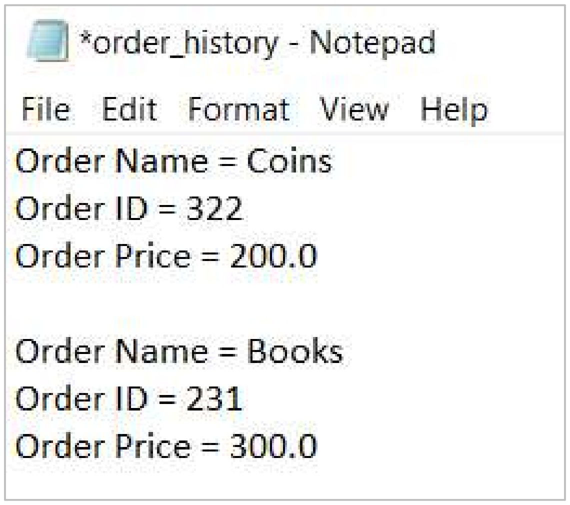


```
Do you want to confirm order? yes=1/no=2
1

************************************************
******************* BILL ***********************
Payment ID = 101
Payment Status = Done !

Customer Name = aliza
Customer ID = 333
Customer Email = aliz@gmail.com

Order Name = Coins
Order ID = 322
Order Price = 200.0

Order Name = Books
Order ID = 231
Order Price = 300.0

------------------------------------------------
TOTAL ORDER PRICE = 500
------------------------------------------------
BUILD SUCCESSFUL (total time: 51 seconds)
```

FILE HANDLING:

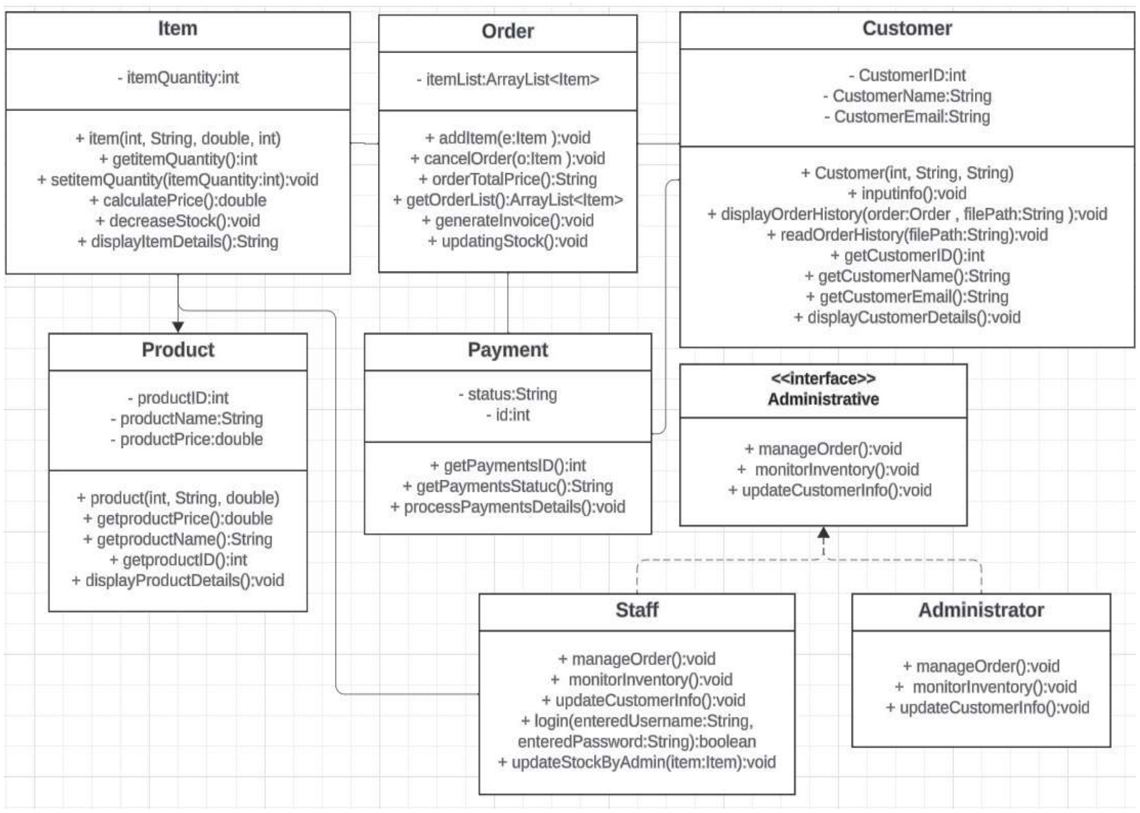


```
Order Name = Books
Order ID = 231
Order Price = 300.0

Order Name = Coins
Order ID = 322
Order Price = 200.0

Order Name = Bags
Order ID = 972
Order Price = 500.0

Order Name = Books
Order ID = 231
Order Price = 300.0
```

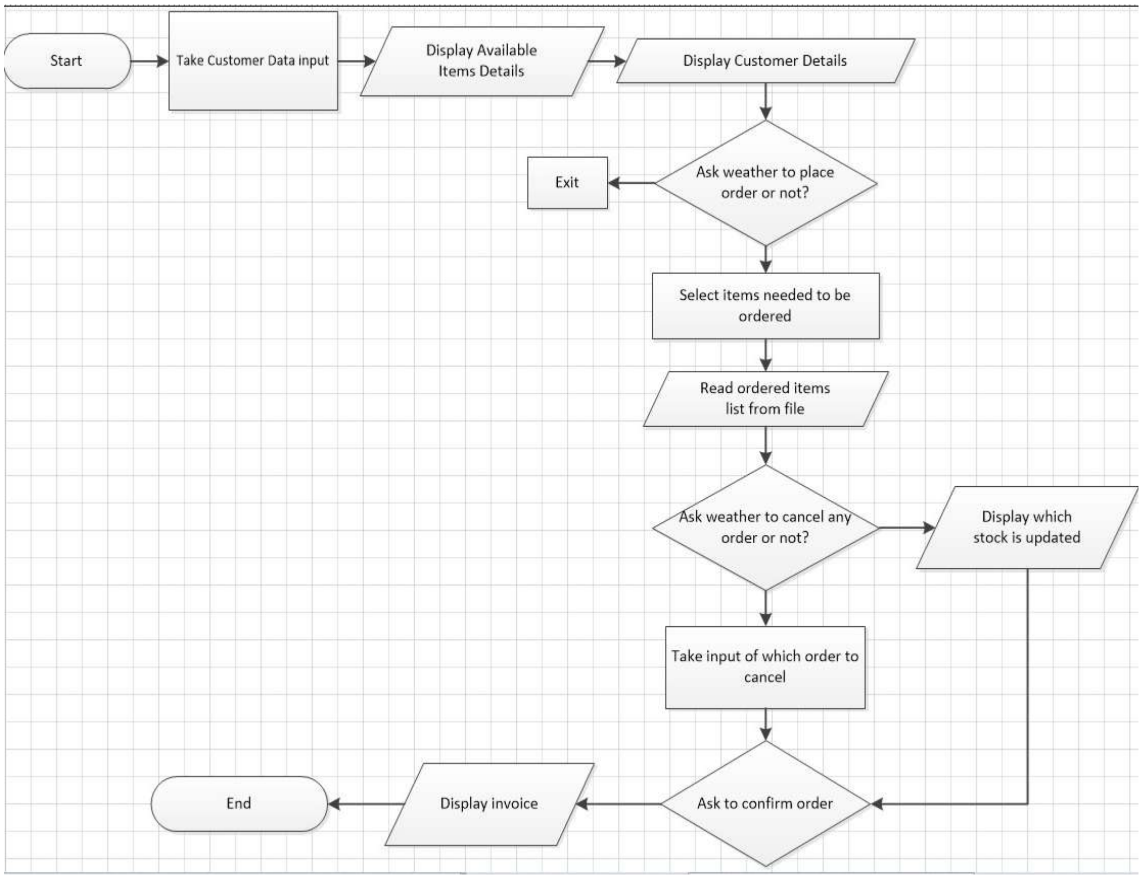


```
Order Name = Coins
Order ID = 322
Order Price = 200.0

Order Name = Books
Order ID = 231
Order Price = 300.0
```

UML DIAGRAM:


Item
- itemQuantity:int
+ item(int, String, double, int)
+ getitemQuantity():int
+ setitemQuantity(itemQuantity:int):void
+ calculatePrice():double
+ decreaseStock():void
+ displayItemDetails():String
Order
- itemList:ArrayList<Item>
+ addItem(e:Item):void
+ cancelOrder(o:Item):void
+ orderTotalPrice():String
+ getOrderList():ArrayList<Item>
+ generateInvoice():void
+ updatingStock():void
Customer
- CustomerID:int
- CustomerName:String
- CustomerEmail:String
+ Customer(int, String, String)
+ inputinfo():void
+ displayOrderHistory(order:Order , filePath:String):void
+ readOrderHistory(filePath:String):void
+ getCustomerID():int
+ getCustomerName():String
+ getCustomerEmail():String
+ displayCustomerDetails():void
Product
- productID:int
- productName:String
- productPrice:double
+ product(int, String, double)
+ getproductPrice():double
+ getproductName():String
+ getproductID():int
+ displayProductDetails():void
Payment
- status:String
- id:int
+ getPaymentsID():int
+ getPaymentsStatuc():String
+ processPaymentsDetails():void
<<interface>>
Administrative
+ manageOrder():void
+ monitorInventory():void
+ updateCustomerInfo():void
Staff
+ manageOrder():void
+ monitorInventory():void
+ updateCustomerInfo():void
+ login(enteredUsername:String, enteredPassword:String):boolean
+ updateStockByAdmin(item:Item):void
Administrator
+ manageOrder():void
+ monitorInventory():void
+ updateCustomerInfo():void

FLOWCHART:


Start
Take Customer Data input
Display Available Items Details
Display Customer Details
Ask weather to place order or not?
Exit
Select items needed to be ordered
Read ordered items list from file
Ask weather to cancel any order or not?
Display which stock is updated
Take input of which order to cancel
Ask to confirm order
Display invoice
End