OPP Assignment 2

Code Outline:

This shopping/order program revolves around one main aspect, the shopping cart, which is a class on its own. The program creates a customers and associated shopping carts. Inside the test class, customer and cart objects are created. These represent sample customers for the program. The customer class creates variables that hold the user's information such as name and email address. A unique customer Id is generated in a method that generates a random number from 1 – 9999999. The shopping cart class features an array list that holds each item that the user selects in the test class. The date and time of the creation of the cart are noted and a cart (array list) and an associated ID are created. The cart class features many methods such as clear, remove, add, and remove that are responsible for removing items from the cart or adding items upon request.

Each item has a name and an ID, and the price is set within the test class. Each item chosen is then added to the cart. Items that are selected are printed to the console, and once finished, the items in the cart are printed.

Once the items are chosen, an order is made to the system. All the items from the cart are transferred to the order, the items are unpacked, and the total price is calculated. A unique order ID is generated (in an iterative way).

The users billing and delivery addresses are noted. The shipping address is associated with the order and the billing address is associated with the payment system.

Once the order is processed, the user's card details are needed. Once received, the system validates the card details. The system checks to see if the card name is correct(i.e., either visa or mastercard) and if it passes this test its valid. If the details are valid, true is passed to the printmail method which prints a payment successful email. If it is not valid, an email is sent to the user informing that payment has failed.

Output For Scenario 1:

```
 Items Selected:

        Item Id: 932472893      Samsung S20     Price: 3422

        Item Id: 6368283        Lenovo S332     Price: 2000

        Item Id: 2433243        Mineral Water   Price: 2

Items in Cart:
                Item Id: 932472893      Samsung S20     Price: 3422

                Item Id: 6368283        Lenovo S332     Price: 2000

                Item Id: 2433243        Mineral Water   Price: 2


The shopping cart has been closed.

        Payment Succeeded

Email: tomatomran666@email.ie
Name: Adam Gleeno
Delivery Address:
Street: 54 Road
City: Big Town
ZipCode: FHSDHU3
Country: Latvia

Billing Address:
Street: 47 Big Road
City: Town
ZipCode: FJF34
Country: Poland

Order Number: 1
Order Details:
Total Cost: 5424.00
Items Orders:
        -       Item Id: 932472893      Samsung S20     Price: 3422

        -       Item Id: 6368283        Lenovo S332     Price: 2000

        -       Item Id: 2433243        Mineral Water   Price: 2
```

Output For Scenario 2:

```
 Items Selected:

        Item Id: 932472893      Food    Price: 9

        Item Id: 6363   Water   Price: 200

        Item Id: 2433243        Pencil  Price: 20

Items in Cart:
                Item Id: 932472893      Food    Price: 9

                Item Id: 6363   Water   Price: 200

                Item Id: 2433243        Pencil  Price: 20


The shopping cart has been closed.
There was a problem verifying your payment information

        Payment Unsuccessful

Email: Yes66@email.ie
Name: Thomas Gleeno
Order Details:
Total Cost: 209.00
Items Orders:
        -       Item Id: 932472893      Food    Price: 9

        -       Item Id: 6363   Water   Price: 200
```

Code:

TransactionTest:

```
/**
 * Testing transaction scenarios class.
 * @author Tim Samoska
 * @version 14/10/22
 */
public class TransactionTest
{
   /**
    *The start of the program
    */
   public static void main(String[] args)
```

```
{
 TransactionTest test = new TransactionTest();
 //Tests for two scenarios
 test.firstTransaction();
 test.secondTransaction();
}

/**
 The first scenario to be tested
 */

public void firstTransaction(){
    //A new customer who wishes to shop is created
    Customer customer1 = new Customer("Adam", "Gleeno",
"tomatomran666@email.ie");

    //A shopping cart object is created for the customer. The customer
    //is assigned a shopping cart. A shopping cart has a customer.
    ShoppingCart shoppingCart = new ShoppingCart(customer1);

    //Items with cost to be added the cart are created
    //Item 1
     System.out.println("\n Items Selected: \n");
    Item item1 = new Item("Samsung S20", 932472893);
    item1.setPrice(3422);
    System.out.println(item1);
    //Item 2
    Item item2 = new Item("Lenovo S332", 6368283);
    item2.setPrice(2000);
```

```
System.out.println(item2);
//Item 3
Item item3 = new Item("Mineral Water",2433243);
item3.setPrice(2);
System.out.println(item3);
//Adding the desired items to the customers cart
shoppingCart.add(item1);
shoppingCart.add(item2);
shoppingCart.add(item3);
System.out.println(shoppingCart);
//Customer's Order is created
Order order = new Order(shoppingCart, customer1);
//Addresses of the Customer are set
//Billing Address
Address billing = new Address("47 Big Road", "Town","FJF34", "Poland");
//Shipping Address
Address shipping = new Address("54 Road", "Big Town","FHSDHU3", "Latvia");
//Shipping Address is needed for the order to be shipped to the customer
order.setShipping(shipping);
//Payment details of the customer
 Payment payment1 = new Payment(customer1, billing, "Visa", 23439, "05/10/2078");
 order.setPayment(payment1);
//Payment datails are validated
Email email = new Email(order);
if(payment1.isValid()){
    //If the card info is valid, Customer is emailed a success email and the total cost of
items
      email.printMail(true);
}
```

```
    else{
        System.out.println("There was a problem verifying your payment information");
        email.printMail(false);
    }
}

/**
 The second scenario to be tested
 */
public void secondTransaction(){
      //A new customer who wishes to shop is created
    Customer customer2 = new Customer("Thomas", "Gleeno", "Yes66@email.ie");

    //A shopping cart object is created for the customer. The customer
    //is assigned a shopping cart. A shopping cart has a customer.
    ShoppingCart shoppingCart = new ShoppingCart(customer2);

    //Items with cost to be added the cart are created
    //Item 1
    System.out.println("\n Items Selected: \n");
    Item item1 = new Item("Food", 932472893);
    item1.setPrice(9);
    System.out.println(item1);
    //Item 2
    Item item2 = new Item("Water", 6363);
    item2.setPrice(200);
    System.out.println(item2);
    //Item 3
    Item item3 = new Item("Pencil",2433243);
```

```
        item3.setPrice(20);
        System.out.println(item3);
        //Adding the desired items to the customers cart
        shoppingCart.add(item1);
        shoppingCart.add(item2);
        shoppingCart.add(item3);
        System.out.println(shoppingCart);

        shoppingCart.remove(item3);
        Order order = new Order(shoppingCart, customer2);

        //Billing Address
        Address billing = new Address("Greg Street", "YesTown","FJF34", "Ireland");
        //Shipping Address
        Address shipping = new Address("54 Road", "Big Town","FHSDHU3", "America");

        order.setShipping(shipping);

        Payment payment1 = new Payment(customer2, billing, "Risa", 23439, "05/10/2078");
         order.setPayment(payment1);

        Email email = new Email(order);
        if(payment1.isValid()){
            //If the card info is valid, Customer is emailed a success email and the total cost of
items
              email.printMail(true);
        }
        else{
           System.out.println("There was a problem verifying your payment information");
```

```
            email.printMail(false);
        }


    }
}
```

ShoppingCart:

```
/**
 * Shopping cart for customers class.
 * @author Tim Samoska
 * @version 14/10/22
 */

//Java util package for arrays is imported
import java.util.ArrayList;

//Java time package is used to fetch current date and time
import java.time.LocalDateTime;
//Package used to format current date and time
import java.time.format.DateTimeFormatter;

public class ShoppingCart
{
    //Instance Variables for the customer's shopping cart

    //Unique Cart ID associated with the customer. In the beginning of shop,
    //there are no carts created yet, therefore its intialized as 0.
    private long cartID = 0;

    //Array of Items
    private ArrayList<Item> cartItems;
```

```
//A shopping cart has a customer.
private Customer customer;

//Date and Time the cart was created for the customer
private String time;

//Total cost of all items
private float totalPrice;

//Boolean variable to keep track of the status of the cart (False = Open/ True = Closed)
//Used for adding/removing items from the shopping cart.
private boolean cartClosedStatus = false;

/**
 * Constructor
 */
public ShoppingCart(Customer customer)
{
   this.customer = customer;
   //Shopping cart is created for the customer
   cartItems = new ArrayList<>();
   cartID = createCartID();

  //Time and Date the shopping cart was created unformatted
  LocalDateTime dateTime = LocalDateTime.now();
  //Desired format for date and time is generated using the datetimeformatter class imported
  DateTimeFormatter dateTimeFormat = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
  //Formatted string is returned
  time = dateTime.format(dateTimeFormat);
```

```
}

/**
 * Methods
 */

 private long createCartID(){
    //Cart ID: As users create carts, the carts variable is incremented to generated a unique
    //number for each cart to keep on track with orders made to the system.

  //Whenever a new cart is made, the ID is incremented.
  return cartID++;
}



//Add/Remove functionalities methods
public void add(Item item){
    //If shopping cart is closed, alert user
    if (cartClosedStatus){
        System.out.println("Sorry the shopping cart is closed.");
        return;
    }
    //Else add requested item to array list(cart)
    cartItems.add(item);
    //Total price of items in cart are updated
    totalPrice = totalPrice + item.getPrice();
    return;
}

 public void remove(Item item){
    //If shopping cart is closed, alert user
    if (cartClosedStatus){
```

```
        System.out.println("Sorry the shopping cart is closed.");
        return;
    }
    //Else remove requested item from array list(cart)

    //If the remove method from the arraylist class removes item successfully, the total
    //is updated by taking away the price of the item removed from the total.
    if(cartItems.remove(item)) {
        totalPrice = totalPrice - item.getPrice();
        return;
    }

    //Else do nothing.
    return;
}

public void clearCart(){
    //The clear method clears everything in the arraylist
    cartItems.clear();
    //Total price of cart is reset
    totalPrice=0;
    System.out.println("The shopping cart has been cleared.");
    return;
}

public void closeCart(){
    //Cart is closed
    cartClosedStatus = true;
    System.out.println("The shopping cart has been closed.");
}

 /**
```

```
 * Accessor Methods
 */

public float getTotal() {
   return totalPrice;
}

public long getCartID(){
   return cartID;
}

public Customer getCustomer(){
   return customer;
}

//Accessor method used to get arraylist
public ArrayList<Item> getCartItems() {
   return cartItems;
}

/**
 * Printing items to string;
 */

@Override
public String toString(){
   String out = "Items in Cart:\n";
   for(Item item: cartItems){
      out+= "\t"+item+"\n";
   }
   return out;
}
```

```
}
```

Order:

```
/**
 * Orders class.
 * @author Tim Samoska
 * @version 14/10/22
 */

import java.util.ArrayList;

public class Order
{
    private Customer customer;
    private ArrayList<Item> orders;
    private float total;
    private long orderNumberID = 0;
    private Payment payment;

    //Used to store customer's shipping address
    private Address shipping;

    public Order(ShoppingCart shoppingCart, Customer customer)
    {
        this.customer = customer;
        //Total price for items got using an accessor method
        total = shoppingCart.getTotal();
        //New orders list created
```

```
        orders = new ArrayList<>();
        //Transferring items from shopping cart into Orders list
        //Shopping cart items are cloned into orders array. Each item in the cart is accessed using an
        //accessor method in the cart that allows to access the cart item. Each item is assigned into
        //Orders array
        for (Item item : shoppingCart.getCartItems()) {
            orders.add(item);
        }
        //Once items in cart have been transferred to orders, cart is closed.
         shoppingCart.closeCart();

         //Generating Unique Order number
         orderNumberID = createOrderID();
    }

    /**
    Methods
     */
     private long createOrderID(){
         //Used to keep track of total orders made in the shop
     //Whenever a new order is made, the ID is incremented.
     orderNumberID = orderNumberID + 1;
     return orderNumberID;
    }

    public void setShipping(Address shipping) {
        this.shipping = shipping;
    }
```

```
 public long getOrderID() {
    return orderNumberID;
}

public Address getShipping() {
    return shipping;
}

 public void setPayment(Payment payment) {
    this.payment = payment;
}

public Payment getPayment() {
    return payment;
}

 public Customer getCustomer() {
    return customer;
}

@Override
public String toString() {
    String out = "Total Cost: " + String.format("%.2f", total)+"\nItems Orders: \n";
    for (Item item : orders) {
        out += "\t- " + item + "\n";
    }
    return out;
}
```

}

Address:

```
/**
 * Addresses of each customer.
 * @author Tim Samoska
 * @version 14/10/22
 */
public class Address
{
   private String street;
   private String zipCode;
   private String city;
   private String country;

   /**
    * Constructor
    */
   public Address(String street, String city, String zipCode, String country)
   {
      //Setting Address for each customer
      this.street = street;
      this.city= city;
      this.zipCode = zipCode;
      this.country = country;
   }

   /**
     Mutator Methods
    */
   public void setStreet(String street) {
```

```
    this.street = street;
}

public void setZipCode(String zipCode) {
    this.zipCode = zipCode;
}

public void setCity(String city) {
    this.city = city;
}

public void setCountry(String country) {
    this.country = country;
}

/**
   Accessor Methods
 */
 public String getStreet() {
    return street;
}

public String getZipCode() {
    return zipCode;
}

public String getCity() {
    return city;
}

public String getCountry() {
    return country;
```

```
    }
}
```

Payment:

```
/**
 * Payment System.
 * @author Tim Samoska
 * @version 14/10/22
 */
public class Payment
{
    private Customer customer;
    private Address billing;
    private int cardNum;
    private String expiry;
    private String cardType;

    //Boolean variable used to validate card info
    private boolean valid;

    /**
     * Constructor
     */
    public Payment(Customer customer, Address billing, String cardType, int cardNum, String expiry)
    {
        //Setting customer payment details
       this.customer = customer;
       this.billing = billing;
```

```
    this.cardType = cardType;
    this.cardNum = cardNum;
    this.expiry = expiry;
    //Card infomation needs to be validated.
    valid = validCardType();

}

/**
 Methods
 */
public boolean validCardType()
{
    //If cardtype entered equals visa or mastercard, return true
    if(cardType.equalsIgnoreCase("visa") || cardType.equalsIgnoreCase("mastercard")){
        return true;
    }
    else
    {
        return false;
    }
}

public boolean isValid() {
    return valid;
}

 public Address getBilling() {
    return billing;
```

```
    }

    public Address getBillingAddress() {
        return billing;
    }
}
```

Email:

```
/**
 * Email class.
 * @author Tim Samoska
 * @version 14/10/22
 */
public class Email
{

    private Order order;
    private String mail;
    /**
     * Constructor
     */
    public Email(Order order)
    {
        this.order = order;
    }

    /**
     * Method used to create email and send it to the user
     */
```

```
    public void printMail(boolean valid)
    {
        //Printing positive message
        if(valid){
            mail = "\n"+"\tPayment Succeeded\n"+
            "\nEmail: "+order.getCustomer().getEmail()+
            "\nName: "+order.getCustomer().getFName()+" "+order.getCustomer().getSName()+
            "\nDelivery Address: \nStreet: "+order.getShipping().getStreet()+
            "\nCity: "+order.getShipping().getCity()+
            "\nZipCode: "+order.getShipping().getZipCode()+
            "\nCountry: "+order.getShipping().getCountry()+
            "\n"+
            "\nBilling Address: \nStreet: "+order.getPayment().getBilling().getStreet()+
            "\nCity: "+order.getPayment().getBilling().getCity()+
            "\nZipCode: "+order.getPayment().getBilling().getZipCode()+
            "\nCountry: "+order.getPayment().getBilling().getCountry()+
            "\n"+
            "\nOrder Number: "+ order.getOrderID()+
            "\nOrder Details:\n"+order;
        }
        else{
            mail = "\n"+"\tPayment Unsuccessful\n"+
            "\nEmail: "+order.getCustomer().getEmail()+
            "\nName: "+order.getCustomer().getFName()+" "+order.getCustomer().getSName()+
            "\nOrder Details:\n"+order;
        }
        System.out.println(mail);
    }
}
```