OOP assignment 2
Maxwell Maia

21236277

Description

The start of a scenario.

The test class creates a Customer object. The customer information is input into the parameters of the Customer object constructor. The purpose of the Customer class is to set and retrieve information about the customer.

Now create a ShoppingCart object. The shopping cart has a customer.

When the ShoppingCart object is being created it initializes its own variables and also creates an array list to hold multiple items. The items class is used here.

The Item class is used to set and retrieve attributes about a singular item. Creating an array list of the type Item allows the program to store multiple items. Because it is an array list it will change its size dynamically which means the array list will compact itself when items in the middle of the array list are removed. This is useful when we are adding and removing items.

Now in the test class item objects are created using the item class.
//Item item2 = new Item("Watering can", 2.99, 31);

These objects are passed one by one into the ShoppingCart object via a method of the ShoppingCart class. This method then places them into the item array list using the .add() function inherited from java.util.ArrayList. As this happens the price of each item is retrieved using a getter of the Item class. The price is added to the total attribute of the shopping cart. There is a Boolean that prevents anything from being added to the array list if the cart is locked. This Boolean is also used to prevent the removal of items when the cart is locked.

In the test class, Address objects are made. Address information is passed into the parameters of the Address constructor. The purpose of the Address class is to set and retrieve information about an address.

In the test class, a function of the ShoppingCart, close() is called. This locks the shopping cart by setting the Boolean “locked” mentioned above to true.

Now that the cart is finalized, and order can be created in the test class. An Order object is created. The Order object has a ShoppingCart, a Customer and two Address objects. This means that all of the information needed to create an order is available to the Order class. Side note: The Order class does not need to access all of the information in the Payment class. Only the billing address is needed in the Order class and since the billing address was created in the test class, the Address is passed into the Order class inside the test class. Because of the nature of object oriented programming it is easy enough to pass the Payment object into the Order object as well, which would make ALL payment information available to the Order object. If the functionality requirements for this program were to evolve in the future, this can be easily done.

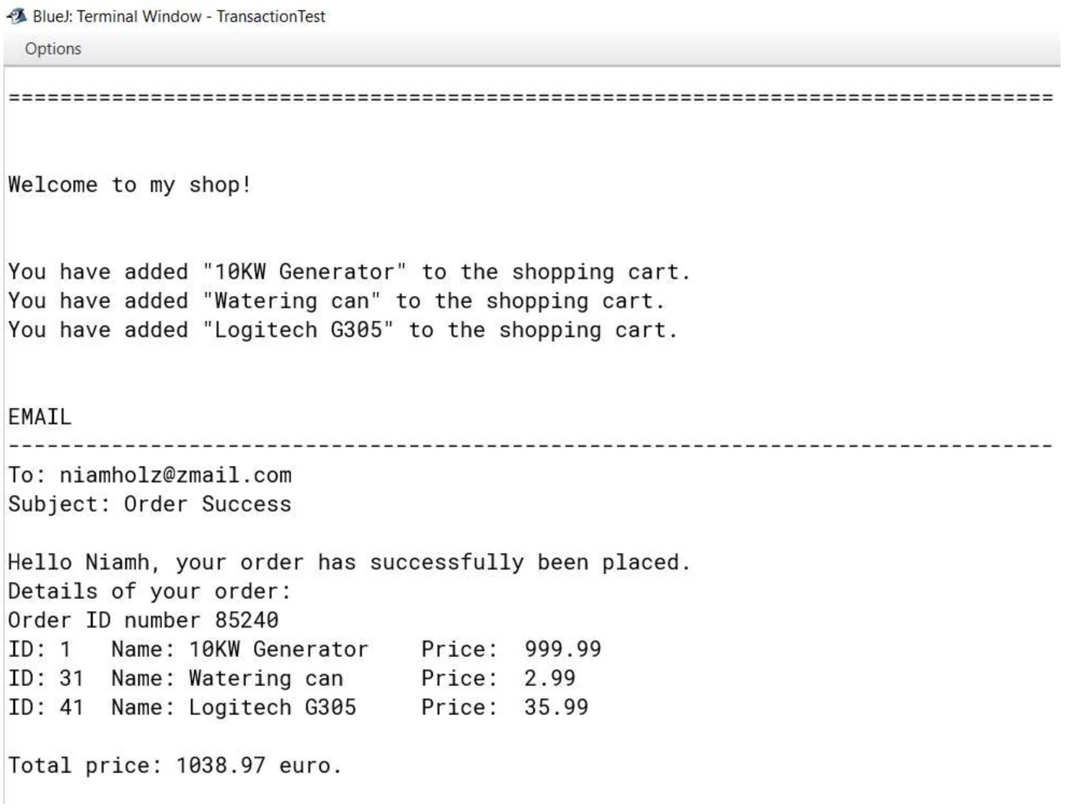
In the test class a Payment object is created using the Payment class. The purpose of the Payment class is to validate the payment details and return a Boolean that will determine whether a payment is valid or not.

The Payment object validates the card by checking that the card type is “Visa” or “MasterCard”. The card number also needs to be 16 digits long. If either of these checks fail then the “valid” Boolean is set to false.

In the test class, an Email object is created. The purpose of the Email class is to compile information from the Order and Customer classes to send an email. The email class has an Order object and the email class has a customer object. This relationship allows the email class to access details within the Order and Customer classes. The Email class has 2 methods, one prints an email describing a successful order and the other prints an email describing an unsuccessful order.

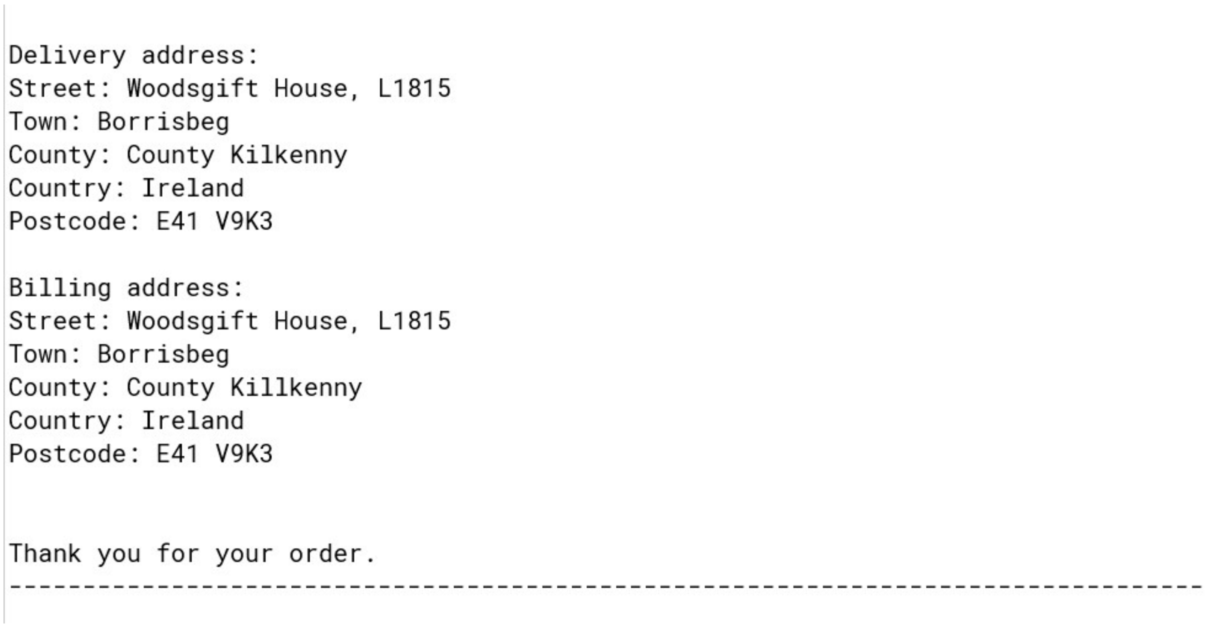
In the test class, the “valid” Boolean of the Payment object is returned. If the payment is valid, send a success email by calling the method in the Email class that prints a successful email. Else, (the payment was not valid) send an email of regret by calling the method in the Email class that prints an email of regret.

Scenario one


BlueJ: Terminal Window - TransactionTest

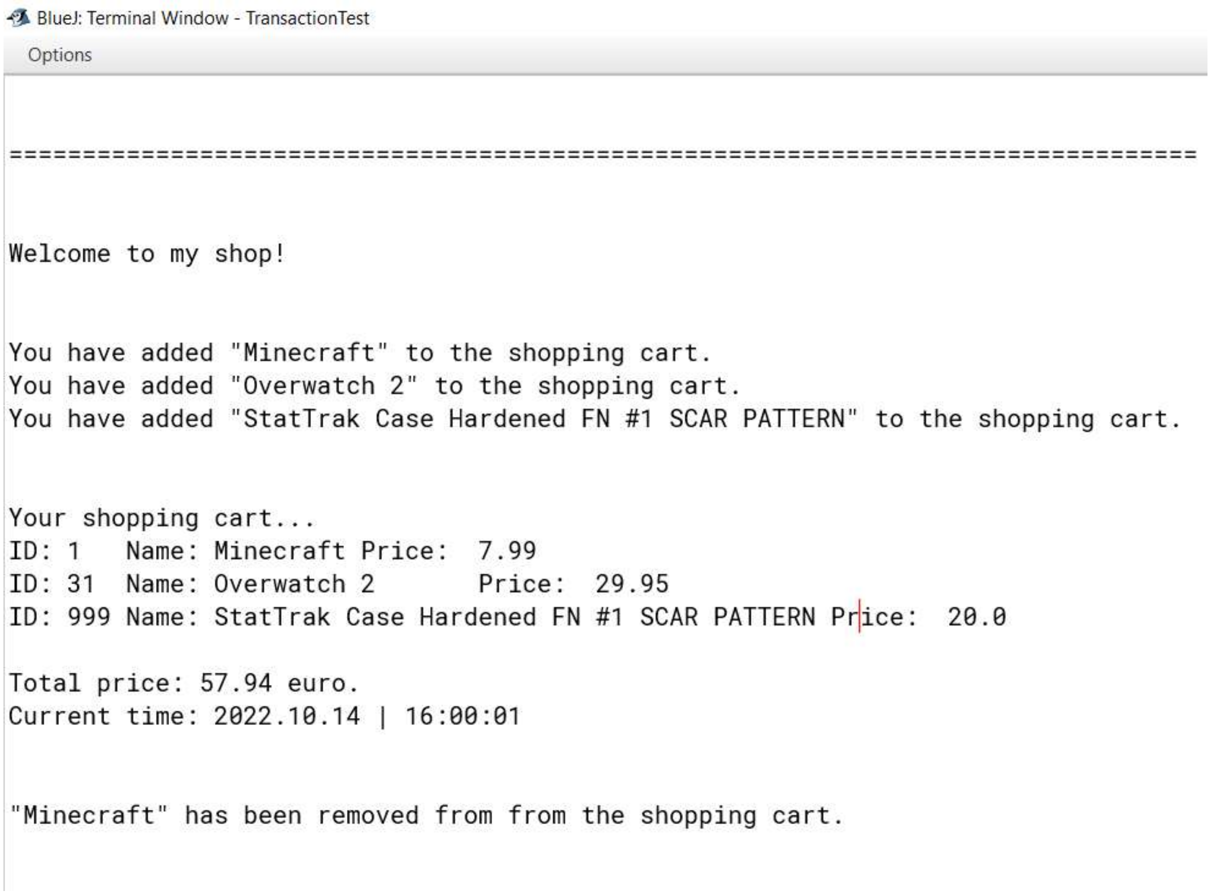
Options

```
=================================================================================


Welcome to my shop!


You have added "10KW Generator" to the shopping cart.
You have added "Watering can" to the shopping cart.
You have added "Logitech G305" to the shopping cart.


EMAIL
---------------------------------------------------------------------------------
To: niamholz@zmail.com
Subject: Order Success

Hello Niamh, your order has successfully been placed.
Details of your order:
Order ID number 85240
ID: 1   Name: 10KW Generator    Price:  999.99
ID: 31  Name: Watering can      Price:  2.99
ID: 41  Name: Logitech G305     Price:  35.99

Total price: 1038.97 euro.
```

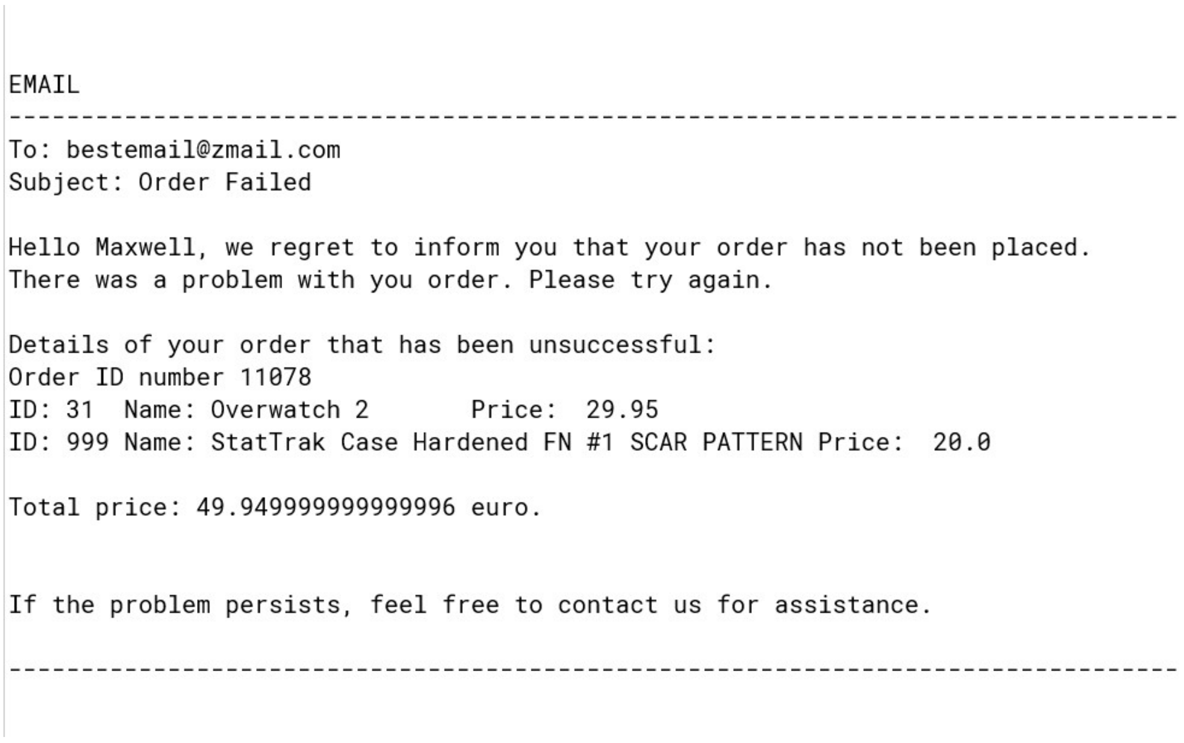


```
Delivery address:
Street: Woodsgift House, L1815
Town: Borrisbeg
County: County Kilkenny
Country: Ireland
Postcode: E41 V9K3

Billing address:
Street: Woodsgift House, L1815
Town: Borrisbeg
County: County Killkenny
Country: Ireland
Postcode: E41 V9K3


Thank you for your order.
---------------------------------------------------------------------------------
```

Scenario two



```
==================================================================================


Welcome to my shop!


You have added "Minecraft" to the shopping cart.
You have added "Overwatch 2" to the shopping cart.
You have added "StatTrak Case Hardened FN #1 SCAR PATTERN" to the shopping cart.


Your shopping cart...
ID: 1   Name: Minecraft Price:  7.99
ID: 31  Name: Overwatch 2       Price:  29.95
ID: 999 Name: StatTrak Case Hardened FN #1 SCAR PATTERN Price:  20.0

Total price: 57.94 euro.
Current time: 2022.10.14 | 16:00:01


"Minecraft" has been removed from from the shopping cart.
```



```
EMAIL
--------------------------------------------------------------------------------
To: bestemail@zmail.com
Subject: Order Failed

Hello Maxwell, we regret to inform you that your order has not been placed.
There was a problem with you order. Please try again.

Details of your order that has been unsuccessful:
Order ID number 11078
ID: 31  Name: Overwatch 2       Price:  29.95
ID: 999 Name: StatTrak Case Hardened FN #1 SCAR PATTERN Price:  20.0

Total price: 49.949999999999996 euro.


If the problem persists, feel free to contact us for assistance.

--------------------------------------------------------------------------------
```

TransactionTest

```
/**
 * TransactionTest - Test the program.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class TransactionTest
{
    public TransactionTest()
    {
        // initialise instance variables
        
    }
    
    public static void main(String[] args)
    {
        TransactionTest test = new TransactionTest();
        
        //Run scenarios
        test.transaction1();
        test.transaction2();
    }
    
    public void transaction1()
    {
System.out.println("\n\n==============================================
===================================");

        System.out.println("\n\nWelcome to my shop!\n\n");
        Customer customer1 = new Customer("Niamh", "O'Leary",
"niamholz@zmail.com");
        ShoppingCart cart1 = new ShoppingCart(customer1);



        //Create 3 items
        Item item1 = new Item("10KW Generator", 999.99, 1);
        Item item2 = new Item("Watering can", 2.99, 31);
        Item item3 = new Item("Logitech G305", 35.99, 41);

        //Add 3 items to the cart
        cart1.addItem(item1);
        cart1.addItem(item2);
        cart1.addItem(item3);



        //Set addresses
        Address deliveryAddress = new Address("Woodsgift House, L1815",
"Borrisbeg", "County Kilkenny", "Ireland", "E41 V9K3");
        Address billingAddress = new Address("Woodsgift House, L1815", "Borrisbeg",
"County Killkenny", "Ireland", "E41 V9K3");

        //Confirm the cart and make an order
        cart1.close();

        Order order = new Order(cart1, customer1, deliveryAddress, billingAddress);



        //Add payment method
        Payment payment1 = new Payment(customer1, billingAddress, "MasterCard",
2658418599332067L, order.getTotal(), "03/12/2023", "AIB");


        /*
         * I included a payment amount in the call to payment below [order.getTotal()]
         * so that someone can upgrade this code in the future
         * to be able to check whether the card has enough funds to make the order.
        */


        //Validate the card details
        payment1.validateCard();



        Email email = new Email(order);
        if (payment1.isValid())
        {
            //Payment details are valid
            //EMAIL SUCCESS
            email.success();
        }
        else
        {
            //Payment details are invalid
            //EMAIL
            email.regret();
        }



        System.out.println("\n");
        System.out.println("\n");
    }
    public void transaction2()
    {

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


        System.out.println("\n\nWelcome to my shop!\n\n");


        Customer customer2 = new Customer("Maxwell", "Maia",
"bestemail@zmail.com");
        ShoppingCart cart2 = new ShoppingCart(customer2);


        //Create 3 items
        Item nr1 = new Item("Minecraft", 7.99, 1);
        Item nr2 = new Item("Overwatch 2", 30.00, 31);
        Item nr3 = new Item("StatTrak Case Hardened FN #1 SCAR PATTERN", 20.00,
999);


        //Add 3 items to the cart
        cart2.addItem(nr1);
        cart2.addItem(nr2);
        cart2.addItem(nr3);


        //Request display of the shopping cart items and total
        cart2.printItems();


        //Remove one item
        cart2.removeItem(nr1);


        //Set addresses
        Address deliveryAddress = new Address("Kill Lane", "Foxrock", "County
Galway", "Ireland", "H91 GP8D");

        Address billingAddress = new Address("25 Rose Inn St.", "Killkenny", "County
Killkenny", "Ireland", "R95 VK02");


        //Confirm the cart and make an order

        cart2.close();


        Order order = new Order(cart2, customer2, deliveryAddress, billingAddress);



        //Add payment method

        Payment payment2 = new Payment(customer2, billingAddress, "MasterCard",
5428L, order.getTotal(), "09/09/2023", "AIB");


        /* REPEATED COMMENT

         * I included a payment amount in the call to payment below [order.getTotal()]

         * so that someone can upgrade this code in the future

         * to be able to check whether the card has enough funds to make the order.

        */


        //Validate the card details

        payment2.validateCard();



        Email email = new Email(order);

        if (payment2.isValid())

        {

           //Payment details are valid

           //EMAIL SUCCESS

           email.success();
        }
        else
        {
            //Payment details are invalid
            //EMAIL
            email.regret();
        }



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



}
```

ShoppingCart

```
import java.util.ArrayList;
import java.util.Date;
import java.text.SimpleDateFormat;
import java.util.Scanner;

/**
 * ShoppingCart - Store and manipulate Items in an Array list.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class ShoppingCart
```

```
{
    private final long cartId;

    private String time;

    private ArrayList<Item> items;

    private double total;


    private Customer customer;


    private boolean locked; //boolean to show whether the cart is locked or not

    //this will be checked to add or remove items

    //this will be updated by close()


    public ShoppingCart(Customer customer)

    {

        this.cartId = generateCartId();

        this.time = new SimpleDateFormat("yyyy.MM.dd | HH:mm:ss").format(new
java.util.Date());


        this.items = new ArrayList<Item>();


        this.total = 0;


        this.customer = customer;


        this.locked = false;

    }



    //Add items to array list (add to shopping cart)

    public void addItem(Item i)

    {
        if(!locked) //Only add an item if cart is not locked
        {
            items.add(i); //Add the item to the array list.
            System.out.println("You have added \"" + i.getName() + "\" to the shopping
cart.");
            total += i.getPrice(); //Update total price.
        }
        else
        {
            System.out.println("Sorry, item not added. The cart is locked.");
        }
    }

    //Return an item using its index
    public Item getItem(int index)
    {
        if(items.get(index) != null)
        {
            return items.get(index);
        }
        else
        {
            System.out.println("This item does not exist! Error code:
ShoppingCartGetItem");
            return null;
        }
    }

    //Removes an item from the array list
    public void removeItem(Item item)
    {
        if(!locked) //Only remove an item if cart is not locked
        {
            if(items.contains(item)) //check that the item is present
            {
                items.remove(item); //Remove item
                total = total - item.getPrice(); //update the total price
                System.out.println("\n\"" + item.getName() + "\" has been removed from
from the shopping cart.\n");
            }
            else
            {
                System.out.println("\nError: Item not found in shopping cart. Please try
again.");
            }
        }
        else
        {
            System.out.println("\nYour cart is locked. Item has not been removed from
shopping cart.\n");
        }
    }

    //Return size of array list
    public int numItems()
    {
        return items.size();
    }

    public double getTotal()
    {
        return total;
```

```
}

public long getCartId()
{
   return cartId;
}

//Print all item details of all items and the current date and time
public void printItems()
{
   System.out.println("\n\nYour shopping cart...");
   if(numItems() > 0)
   {
      System.out.println(writeItems());
      System.out.println("Current time: "+ time + "\n");
   }
   else
   {
      System.out.println("The shopping cart is empty.");
   }
}

//Returns a string of all item details of all items.
public String writeItems()
{
   String out = "";
   for(int i = 0; i < numItems(); i++)
      {

         out += getItem(i).toString() + "\n";
```

```
            }
            out += "\nTotal price: " + total + " euro.";
            return out;
    }

    //Lock the cart
    public void close()
    {
        locked = true;
    }

    //Empty the cart
    public void clear()
    {
        items = null;
        total = 0;
    }

    public long generateCartId()
    {
        return (long)(Math.random() * 9999L);
    }
}
```

Order

```
import java.util.ArrayList;
import java.util.Scanner;
```

```
/**
 * Write a description of class Order here.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class Order
{
   private long orderId;

   private Address deliveryAddress;
   private Address billingAddress;

   private ShoppingCart cart;
   private Customer customer;

   private ArrayList<Item> orderItems;

   private double total;



   public Order(ShoppingCart cart, Customer customer, Address delivery, Address
billing)
   {
      orderId = generateOrderId();

      this.cart = cart;
      this.customer = customer;
      deliveryAddress = delivery;
      billingAddress = billing;
```

```
    orderItems = new ArrayList<>(); //To store items in order.

    //Transfer items one by one to Order object
    for(int i = 0; i < cart.numItems(); i++)
    {
        orderItems.add(cart.getItem(i));
    }

    total = cart.getTotal(); //get the total

    cart.clear(); //Empty the shopping cart.
}

public long generateOrderId()
{
    return (long)(Math.random() * 99999L);
}

public long getOrderId()
{
    return orderId;
}

public double getTotal()
{
    return total;
}


//Return a String with the item details
```

```
 // of the items that the customer is attempting to order.
 public String writeItems()
 {
     String out = "";
     for(int i = 0; i < orderItems.size(); i++)
        {

            out += orderItems.get(i).toString() + "\n";

        }
        out += "\nTotal price: " + total + " euro.";
        return out;
 }



public Customer getCustomer()
{
    return customer;
}

//Return a string of all address details via the Address toString()
public String getDeliveryAddressString()
{
    return deliveryAddress.toString();
}

//Return a string of all address details via the Address toString()
public String getBillingAddressString()
{
    return billingAddress.toString();
    }
}
```

Address

```
/**
 * Address - set and retrieve information about an Address.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class Address
{
    private String street;
    private String town;
    private String county;
    private String country;
    private String postcode;



    public Address(String street, String town, String county, String country, String
postcode)
    {
        this.street = street;
        this.town = town;
        this.county = county;
        this.country = country;
        this.postcode = postcode;
    }
    public String getStreet()
    {
        return street;
    }

    @Override
    public String toString()
    {
        String out = "Street: " + street + "\nTown: " + town + "\nCounty: " + county +
"\nCountry: " + country + "\nPostcode: " + postcode;

        return out;
    }
}
```

Payment

```
/**
 * Payment - validate the payment details and return the valid boolean.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class Payment
{
    private Customer customer;

    private String cardType;
    private long cardNumber;
    private String expDate;

    private Address billingAddress;
    private String bankName;

    private boolean valid;

    private double total;



    public Payment(Customer customer, Address billingAddress, String cardType,
long cardNumber, double total, String expDate, String bankName)
    {
        this.customer = customer;
        this.billingAddress = billingAddress;
        this.cardType = cardType;
        this.cardNumber = cardNumber;
        this.total = total;
        this.expDate = expDate;
        this.bankName = bankName;
    }

    //Validate payment details and update the valid boolean.
    public void validateCard()
    {

        int cardNumLength = String.valueOf(cardNumber).length();

        //The card type has to be either Visa or MasterCard.
        //And the card number must be 16 digits long.
        if((cardType.equals("MasterCard") || cardType.equals("Visa")) &&
cardNumLength == 16)
```

```
        {
            valid = true;
        }
        else
        {
            valid = false;
        }
    }

    //Return the boolean that describes whether the payment details are valid or not
    public boolean isValid()
    {
        return valid;
    }
}
```

Email

```
/**
 * Email - Compiles information to send an email (print to terminal).
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class Email
{
    private Order order;
    private Customer customer;
    public Email(Order order)
    {
        this.order = order;
        customer = order.getCustomer();
    }



    //Send an email describing success
    public void success()
    {
        String out = "";
        out += "\n\nEMAIL\n";
        out += "--------------------------------------------------------------------------------";
        out += "\nTo: "+customer.getEmailAddress()+"\nSubject: Order Success\n\nHello "+customer.getFirstName()+", your order has successfully been placed.";
        out += "\nDetails of your order:\n";
        out += "Order ID number " + order.getOrderId() + "\n";
        out += order.writeItems();
        out += "\n\n\n";
        out += "Delivery address:\n" + order.getDeliveryAddressString();
        out += "\n\nBilling address:\n" + order.getBillingAddressString();

        out += "\n\n\nThank you for your order.\n";

        out += "--------------------------------------------------------------------------------";



        System.out.println(out);
    }
    //Send an email describing order failure
    public void regret()
    {
        String out = "";
        out += "";
        out += "\n\nEMAIL\n";
        out += "------------------------------------------------------------------------------";
        out += "\nTo: "+customer.getEmailAddress()+"\nSubject: Order Failed\n\nHello
"+customer.getFirstName();;
        out += ", we regret to inform you that your order has not been placed. \nThere
was a problem with you order. Please try again.\n";
        out += "\nDetails of your order that has been unsuccessful:\n";
        out += "Order ID number " + order.getOrderId() + "\n";
        out += order.writeItems();

        out += "\n\n\nIf the problem persists, feel free to contact us for assistance.\n\n";



        out += "------------------------------------------------------------------------------";

        System.out.println(out);
    }
}
```

ADDITIONAL CLASSES

Customer

```
/**
 * Customer - set and retrieve information about the Customer.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class Customer
{
    private String firstName;
    private String surName;
    private String emailAddress;
    private final long customerId;



    public Customer(String firstName, String surName, String emailAddress)
    {
        this.firstName = firstName;
        this.surName = surName;
        this.emailAddress = emailAddress;
        this.customerId = generateCustomerId(); //Creates a customer ID
    }

    public long getId()
    {
        return customerId;
    }

    public String getFirstName()
    {
        return firstName;
    }

    public String getSurName()
    {
        return surName;
    }

    public String getEmailAddress()
    {
        return emailAddress;
    }

    public long generateCustomerId()
    {
        return (long)(Math.random() * 9999L);
    }
}
```

Item

```
/**
 * Item - set and retrieve information about an item.
 *
 * @author Maxwell Maia
 * @version 1.0
 */
public class Item
{
    private String name;
    private double price;
    private long itemId;


    public Item(String itemName, double price, long id)
    {
        this.name = itemName;
        this.price = price;
        this.itemId = id;
    }

    public void setPrice(double price)
    {
        this.price = price;
    }

    public double getPrice()
    {
        return price;
    }

    public String getName()
    {
        return name;
    }

    @Override
    public String toString()
    {

        String out = "ID: " + itemId + "\tName: " + name + "\tPrice:\t" + price;

        return out;
    }
}
```