Assignment 2

21325456 – Cathal Lawlor

My approach to this was having two array lists, one in shopping cart and other in order.
These were the two ‘heavy lifters’ in terms of code.

Items objects make up the cart list on the arraylist in shoppingCart. The user is if they’d like to confirm the cart which is then sent to the order class. The confirmed card is assigned to the order’s array list. This kicks off the order class which processes the order and prompts the payment processing/validation and the billing & delivery address formation. Finally, the email is sent out, informing the user of the order details or that the processing has failed due to validation on the payment.

Unfortunately, I never got the date working in time for the submission in the payment class.

TransactionTest starts off by creating customer and shopping objects containing info for the customer object.
It proceeds by adding in the items we’d like in our shopping cart after creating them. In scenario two it removes one of these items. We then ‘confirm’ the class by prompting the customer to answer yes or no.

This confirmation seals off the cart and then we proceed by assigning the cart details over to the new order object. This order sets delivery address and billing address to the order. It then asks the payment to check the payment info given. After all this the email is created and it is ‘sent’ containing the details of the order. For scenario one it all passed smoothly but in scenario two it fails and the user is notified.

Note: I didn’t realise there was customer and item class available so I made my own. I have attached them at the end of this pdf and in the zip.

Output scenario 1:

```
Welcome to the greatest shop known to mankind!

Headphones added to the shopping cart
Mountain Bike added to the shopping cart
Harrier Jet Wreckage added to the shopping cart
Id: 1  -  Headphones    Cost: 120.0
Id: 2  -  Mountain Bike Cost: 899.0
Id: 3  -  Harrier Jet Wreckage  Cost: 24600.0

Above items added to your cart, thank you.
 Confirm your order, Yes || No?
y
Cart confirmed
- - - - - - - - - - - - - -
Mail to:Cathal

 Davis c.Lettuce@nuigalway.ie,

Order contents and details enclosed below

   Order no: 1532751389306   Items:
        Id: 1  -  Headphones    Cost: 120.0
        Id: 2  -  Mountain Bike Cost: 899.0
        Id: 3  -  Harrier Jet Wreckage  Cost: 24600.0
   Total: 25619.0

   Order will be delivered to: 22 Pound Hill,
                               Gortnaclohy,
                               Co. Cork,
                               Ireland,
                               P81 Y499.

   Order will be billed to: 16 Fairview Manor, Ballyederowen,
                               Burnfoot,
                               Co. Donegal,
                               Ireland,
                               F93 Y523.

Thank you again for shopping here
Have a lovely day,
Shop-102 Staff

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

Output scenario 2

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

 Ok, perhaps not the best shop but it's up there!

Black pudding added to the shopping cart
Porridge added to the shopping cart
Milk added to the shopping cart

Removed the item: Porridge from your cart

Id: 4  -  Black pudding Cost: 2.0
Id: 6  -  Milk  Cost: 1.5

Above items added to your cart, thank you.
 Confirm your order, Yes || No?
y
Cart confirmed
- - - - - - - - - - - - - - -
Mail to:Liam

 Schukat Liam.Sch@mouseMail.ie,

Unfortunetly your order has not been processed.
We're sorry, the validation on your payment has failed. Please check and retry.


Many thanks
Shop-102 Staff
```

TransactionTest

```
/**
 * Test class for all of the rest of the classes.
 *
 * @Cathal Lawlor - 21325456
 * @V1.1
 */
public class TransactionTest
{

   public static void main(String[] args)
   {
      TransactionTest test = new TransactionTest();
      test.transaction1(); // calls the method holding our first test scenario
      test.transaction2(); // calls the method holding our secon test scenario
   }

   public void transaction1() {
      System.out.println("\nWelcome to the greatest shop known to mankind!\n"); // Opening Print Statement
      Customer customer1 = new Customer("Cathal", "Davis", "c.Lettuce@nuigalway.ie"); // Creating Customer object
      ShoppingCart cart1 = new ShoppingCart(); // Creating ShoppingCart object

      // instantiating item objects
      Item item1 = new Item("Headphones", 120);
      Item item2 = new Item("Mountain Bike", 899);
      Item item3 = new Item("Harrier Jet Wreckage", 24600);

      // Adding in the items to the shoppingcart
      cart1.addItem(item1);
      cart1.addItem(item2);
      cart1.addItem(item3);

      cart1.confirmCart(); //confirming cart - user is asked whether they want to confirm
```

```
    cart1.closeCart(); //boolean variable is controlled by this method. Cart will be sealed off when cartOpen is set to false

    // Passing in addresses
    Address deliveryAddress = new Address("22 Pound Hill", "Gortnaclohy", "Co. Cork", "Ireland", "P81 Y499");
    Address billingAddress = new Address("16 Fairview Manor, Ballyederowen", "Burnfoot", "Co. Donegal", "Ireland", "F93 Y523");

    Payment firstPayment = new Payment("Mastercard", 2334113943320401L, "03/24", "AIB"); // Payment object instantiated - Valid information

    Order firstOrder = new Order(cart1, firstPayment); // Order Object instantiated

    // Setting in the addresses in delivery and billing address
    firstOrder.setDelAddress(deliveryAddress);
    firstPayment.setBillAddress(billingAddress);

    // Creating email object and sending email
    Email email = new Email(customer1);
    email.emailSend(firstOrder);
  }

  public void transaction2() {
    System.out.println("\n Ok, perhaps not the best shop but it's up there!\n");
    Customer customer2 = new Customer("Liam", "Schukat", "Liam.Sch@mouseMail.ie");
    ShoppingCart cart2 = new ShoppingCart();

    Item item1 = new Item("Black pudding", 2); //Lovely for a good fry
    Item item2 = new Item("Porridge", 3);
    Item item3 = new Item("Milk", 1.5);

    cart2.addItem(item1);
    cart2.addItem(item2);
    cart2.addItem(item3);
```

```
        cart2.removeItem(1); //removing the 1st item in the arrayList


        cart2.confirmCart();
        cart2.closeCart();


        Address deliveryAddress = new Address("6 Glenard Avenue", "Salthill", "Co. Galway", "Ireland", "H91 PT4V");
        Address billingAddress = new Address("5 Manor Way", "Shanagolden", "Co. Limerick", "Ireland", "V94 CCH6");
        Payment firstPayment = new Payment("Vita", 8654644668811230L, "10/18", "Bank Of Ireland"); // Payment object instantiated - Not valid information - Vita card and wrong starting number for cardNum
        Order orderTwo = new Order(cart2, firstPayment);
        orderTwo.setDelAddress(deliveryAddress);
        firstPayment.setBillAddress(billingAddress);


        Email email = new Email(customer2);
        email.emailSend(orderTwo);
    }
}
```

ShoppingCart

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

// For date handling but found it too tricky to get working
//import java.text.DateFormat;
//import java.text.SimpleDateFormat;
//import java.util.Calendar;
//import java.util.Date;

/**
 * The 'class in charge' at the start. It holds an array for the items added to it and then to be sent to order.
 *
 * @Cathal
 * @V1.3
 */
public class ShoppingCart
{
   // Instance variables
   private ArrayList<Item> sList;
   private Item item;
   private long cartId;
   private int total;
   private boolean cartOpen;

   public ShoppingCart()
   {
      //Initialised to a new empty arraylist (for holding Item objects)
      sList = new ArrayList<Item>();
      cartId = getCartId();
      cartOpen = true;
   }
```

```
public void addItem(Item i)
{
   //Adding in an item object that's read in to the arraylist

   if (cartOpen) { //only happens if cart is open
      sList.add(i);
      System.out.println(i.getName(i) + " added to the shopping cart");
   } else { //failure statement
      System.out.println("Failure to add item; shopping card is closed");
   }
}

public Item getItem(int index)
{
   //Reads in an index, returns item at passed in index
   if(sList.get(index)!=null)
   {
      return sList.get(index);
   }
   else
   {
      System.out.println("The shopping cart is empty!");
      return null;
   }
}

public void closeCart() {
   cartOpen = false; //closes off the cart when called
}

public void confirmCart() {
   listItems();
   System.out.println("\nAbove items added to your cart, thank you.\n Confirm your order, Yes || No?");
```

```
    Scanner in = new Scanner(System.in);

    /* Using scanner we take the user input to determine if they want to proceed & confirm
     * current cart. Takes yes/no and y/n as inputs */

    char input = in.next().charAt(0);

    //if else statements to let user know of machine result
    if (input == 'y') {
       System.out.println("Cart confirmed");
    }
    else if (input == 'n') {
       System.out.println("Cart cancelled, closing down shop");
       System.exit(0);
    }
    else {
       System.out.println("Out of bounds input, closing down shop");
       System.exit(0);
    }

  }




  public boolean removeItem(int index)
  {
    if (cartOpen) { //only works if the cart is open
       if(sList.get(index)!=null) // if there is an item we proceed
       {
          System.out.println("\nRemoved the item: " + getItem(index).getName(item) + " from your cart\n");
//notifying user of change
          sList.remove(index);
          return true;
```

```
        }
        else
        {
            return false;
        }
    }
    return false;
}

 public long getCartId() { //for finding a random cart ID we use the random function
    return (long)(Math.random() * 999999999999L);
}

public int numItem() //number of items in the array
{
    return sList.size();
}

public double getTotal(){ //rudimentry enough but it works by summing everything going through the array
                    //tried implementing a running counter when adding / removing items but was too buggy
    total = 0;
    int size = numItem();
    for(int idx = 0; idx < size; idx++) {
        total += sList.get(idx).getCost();
    }

    return total;
}

public void emptyCart() { //clears off the cart if needed
    sList = null;
}

public void listItems() //method for listing off the items, and their fields
```

```
    {
        if(numItem()>0)
        {
            for(int i=0; i<numItem(); i++)
            {
                System.out.println(getItem(i).toString());
            }
        }
        else
        {
            System.out.println("This group is currently empty.");
        }
    }

}
```

Order

```
import java.util.ArrayList;
import java.util.Arrays;
/**
 * Order class, holds lots of attributes and then takes in shopping cart onto a new arraylist.
 * It then processes the addresses and prompts the payment class to process the card details
 *
 * @Cathal
 * @V1.3
 */
public class Order
{
   // Instance variables
   private Address deliveryAddress;
   private boolean deliveryAddressValid;
   private Payment payment;
   private ShoppingCart cart;
   private ArrayList<Item> confirmedItems;
   private long orderNumber;
   private double totalCost;

   public Order(ShoppingCart cart, Payment payment) {
      confirmedItems = new ArrayList<>(); // Initialising ArrayList for the order
      orderNumber = setOrderId(); // Creating order ID

      // Copying passed through objects
      this.cart = cart;
      this.payment = payment;

      this.totalCost = cart.getTotal(); // Getting total cost for order

      for (int i = 0; i < cart.numItem(); i++) {
         confirmedItems.add(cart.getItem(i)); // Adding items individually to order
```

```
        }
        cart.emptyCart(); // Emptying shopping cart of items
    }
    // Setter Functions
    public long setOrderId() {
        return (long)(Math.random() * 99999999999999L);
    }
    public void setDelAddress(Address deliveryAddress) {
        deliveryAddressValid = true; //confirming that the address is available to be used.
        this.deliveryAddress = deliveryAddress;
    }
    public void printItems() {
        for (int i = 0; i < confirmedItems.size(); i++) {
            System.out.println("\t" + confirmedItems.get(i)); //going through the confirmed items and printing
        }
    }
        public Payment getPayment() {
        return payment;
    }
    public double getOrderCost() {
        return totalCost;
    }
    public long getOrderId() {
        return orderNumber;    }
// method for the string out to aggregate the getter functions on delivery address
    public String getDelAddress() {
        if (deliveryAddressValid) { // Checks if deliveryAddress has been set
            String out = deliveryAddress.getStreet() + ",\n\t\t\t\t" + deliveryAddress.getTown() + ",\n\t\t\t\t" +
deliveryAddress.getCounty() + ",\n\t\t\t\t" + deliveryAddress.getCountry() + ",\n\t\t\t\t" +
deliveryAddress.getEircode() + ".";
            return out;
        } else {
            String out = "Address not found.";
            return out;
        }   }     }
```

Address

```
/**
 * Address class, basic in that it holds address info and just getter methods
 *
 * @Cathal
 * @V1.3
 */



public class Address
{
   // Instance variables
   private String street;
   private String town;
   private String county;
   private String country;
   private String eircode;

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

   // Getter Functions
      public String getEircode() {
      return eircode;
   }

   public String getStreet() {
      return street;
   }
```

```
    public String getTown() {
        return town;
    }

    public String getCounty() {
        return county;
    }

    public String getCountry() {
        return country;
    }

}
```

Payment

```
//import java.text.SimpleDateFormat;
//import java.util.Date;
import java.text.ParseException;

/**
 * Payment class, holds the card details and then validates the card details based on card number and card type
 *
 * @Cathal
 * @V1.3
 */

public class Payment
{
   // Instance variables

   private String cardDate;
   private String cardType;
   private long cardNumber;
   private String cardBankName;
   private Customer customer;
   private boolean billingAddressValid;
   private Address billingAddress;

   public Payment(String cardType, long cardNumber, String cardDate, String cardBankName) {
      // Copying passed through variables into local variables
      this.cardNumber = cardNumber;
      this.cardType = cardType;
      this.cardDate = cardDate;
      this.cardBankName = cardBankName;
   }

   // This function validates if payment details are correct
   public boolean isValid() {
```

```
        String cardNumStr = Long.toString(cardNumber); // Converting to string so we can then check position [0]
        if (String.valueOf(cardNumber).length() == 16) { // Check if card number is 16 digits long
            // I've made a check for the first digit on cardNumber. Master card has a specific first number with 2 for example. If it doesn't match then it is rejected.
            // Check if cardType is an accepted payment type. Checks for the payment type name and if it's a valid type e.g. not Vita or MateUrCard
            if ((cardType == "Mastercard" ) || (cardType == "Visa")) {
                return true;
            }
        }
        return false;
    }

    public void setBillAddress(Address billingAddress) {
        billingAddressValid = true;
        this.billingAddress = billingAddress;
    }

    // method for the string out to aggregate the getter functions on billing address - same as
    public String getBillAddress() {
        if (billingAddressValid) { // Check for if billing address has been set
            String out = billingAddress.getStreet() + ",\n\t\t\t\t" + billingAddress.getTown() + ",\n\t\t\t\t" + billingAddress.getCounty() + ",\n\t\t\t\t" + billingAddress.getCountry() + ",\n\t\t\t\t" + billingAddress.getEircode() + ".";
            return out;
        } else {
            String out = "Address not found";
            return out;
        }
    }
}
```

Email

```
/**
 * Email class for sending out email containing details of payment and address. Mainly print statements.
* @Cathal
 * @V1.3
 */
public class Email
{
   // Instance variables
   private Customer customer;
   private Order order;

   public Email(Customer customer) {
      this.customer = customer;    }

   public void emailSend(Order order) {
      /* Function to handle the passed in order object. It takes in all of the customer data, checks
       * if the payment is valid, generates a confirmation letter. */
      System.out.println("- - - - - - - - - - - - - -");
      System.out.println("Mail to:" + customer.getEmailAddress());
      System.out.println("\n " + customer.getFirstName() + " " + customer.getSurname() + ",\n");
      if(order.getPayment().isValid()){
         System.out.println("Order contents and details enclosed below");
         System.out.println("\n   Order no: " + order.getOrderId() + "   Items: ");
         order.printItems();
         System.out.println("   Total: " + order.getOrderCost());
         System.out.println("\n   Order will be delivered to: " + order.getDelAddress()); // Error statement included if no address has been found
         System.out.println("\n   Order will be billed to: " + order.getPayment().getBillAddress()); // Error statement included if no address has been found
         System.out.println("\nThank you again for shopping here\nHave a lovely day,\nShop-102 Staff\n");
         System.out.println("- - - - - - - - - - - - - -");
      }
      else{
         System.out.println("Unfortunately
```

Customer

```
/**
 * Customer class, like item it holds lots of attributes and getter and setter methods
 *
 * @Cathal
 * @V1.3
 */
public class Customer
{
   // Instance variables
   private final long customerID;
   private String fName;
   private String surname;
   private String emailAddress;

   public Customer(String emailAddress, String firstName, String surname) {
      this.surname = surname;
      this.fName = firstName;
      this.emailAddress = emailAddress;
      customerID = genCustomerID();
   }

   //generating random ID for customer
    private long genCustomerID() {
      return (long) (Math.random() * 99999999999999L);
   }


   //Getter and setter methods

   public String getSurname() {

      return surname;
```

```
    }

    public String getEmailAddress() {
        return emailAddress;
    }

    public long getID() {
        return customerID;
    }

    public String getFirstName() {
        return fName;
    }




    public void setEmailAddress(String emailAddress) {
        this.emailAddress = emailAddress;
    }

        public void setFirstName(String first) {
        fName = first;
    }

    public void setSurname(String sName) {
        surname = sName;
    }

}
```

Item

```
/**
 * Item class. Holds item attributes with get and set methods
 *
 * @Cathal
 * @V1.3
 */
public class Item
{
   // Instance variables
   private double cost;
   private String name;
   private int itemID;
   private static int prevId = 0;

      public Item(String name) { // taking in a new item with no cost amount
      this.name = name;
      itemID = generateID();
   }

      public Item(String name, double cost) { //taking in a new item with cost
      this.name = name;
      this.cost = cost;
      itemID = generateID();
   }

      private int generateID() { //sequential ID generator for items
      int id = prevId + 1;
      prevId = id;

      return id;
   }
```

```
    @Override
    public String toString() {
        String out = "Id: " + itemID + "  -  " + name + "\tCost: " + cost; //sending out the order's list


        return out;
    }




    //Get and set methods
    public void setCost(double cost) {
        this.cost = cost;
    }

    public String getName(Item item) {
        return name;
    }

    public double getCost()
    {
        return cost;
    }

}
```