Assignment 2 – Oisin Mc Laughlin – 22441106

Question 1 - Analysis

For this assignment I will require 8 classes:

- Customer
- Item
- ShoppingCart
- Order
- Address
- Payment
- Email
- TransactionTest

Customer:
For this class most of the code has been supplied in the lecture slides. There are four class fields; a string for firstName, surName, emailAddress and an int customerId. In the constructor all of these are initialised but customerId is initialised by calling a method makeCustomerId(). This method will generate a random number between 1 and 99 using the java random library and return it. This class will also have accessors for each of the fields.

Item:
Once again most of the code for this class has been provided from the lecture slides, it has 3 fields; a string name, int price and int id. The class also contains an accessor and a mutator for price. It also contains a toString method which returns a string with the details of the item; id, name and price. The only thing I will change in this class will be changing price from an int to a double.

ShoppingCart:
This class will contain five fields; int cartId, an array cartItems, float totalPrice, boolean isLocked a customer object. It will also use the java librarys for lists and arraylists. Each of these will be initialised in the constructor and isLocked will be set to false. There will be two methods for adding and removing items from the cart. The method addItem has a parameter which is an object of item and will check if the cart is locked by checking if isLocked is false, if the cart isn't locked it will use the add function from the library and add the item object to the cartItems array. It will also use the item accessor getPrice and add the price to totalPrice. If the cart is locked it will produce a message saying that the cart is locked. The removeItem array will perform the same way except using

the remove function from the library and removing the price found from getPrice from totalPrice. This class will also have a few accessors to get totalPrice, cartId, customer and cartItems. There will also be two methods to loop through the cart called printItems and clear. The printItems method will loop through and print each item by using the get function from the library. The clear method will loop through and remove each of the items in the cart, it will also set totalPrice to 0. There will be one more method in the class called close which will just set isLocked to true.

Order:
The order class will contain the class fields; arraylist orderItems and an object for customer, address, email and payment. It will also have a long orderNum, string orderStatus, string orderDetails and a float orderTotal. The constructor will initialise each of these and set orderNum to a method makeOrderNum() which will once again generate a number between 1 and 99 the same way as the customerId. orderStatus will be initialised as a string status saying incomplete or something along those lines. orderDetails will call a method that will display each item and orderTotal will call a method calcTotalprice which will loop through and add each of the item prices to a variable and return it. There will be a method called transferItems which will have a shoppingcart object and will loop through the cart and add each of the items to orderItems then calling the shoppingcart clear method. I also will need a mutator for the orderStatus and an accessor for orderNum, orderDetails and orderTotal.

Address:
The address fields will have the strings street, city, zip and country. There will be an accessor and mutator for each of these and a toString method to display each of them.

Payment:
This payment fields will have a object customer, string cardType, long cardNum, string date, object address and string bankName, after initialising each of these in the constructor, there will be an accessor for the cardType and a method to validiate the payment by checking if the cardType is either “visa” or “mastercard” and returning true if either of them match. There will also be a toString method to display each of the information about the card.

Email:
In the class field there will be an object from each of the other classes as it will be printing a message if the payment was successful by checking if the

cardType was validated (returning true). The email will contain things such as items, orderNumber, name, address etc. If the payment was not validated it will return a different message saying it was unsuccessful.

TestTransaction:
This class will have two scenarios from the lecture slides, one will be where the payment was successful and the other where it was not successful.

Question 2

TransactionTest:

```
/**
 * As illustrated in the lectures, this is a test class. It should have two methods - one for each
scenario. They are executed from the main method, which is contained by the TransactionTest
class
 *
 * @author (Oisin Mc Laughlin)
 * @version (v1.0)
 */
public class TransactionTest
{

   public TransactionTest()
   {
      //no instance variables
   }

   /**
    * main method - program execution starts here
    */
   public static void main(String[] args)
   {
      TransactionTest test = new TransactionTest();
      test.transaction1();
      test.transaction2();
   }

   public void transaction1() {
```

```
        System.out.println("\n\n\nScenario 1\n");
        //Create Customer Object
        Customer customer = new Customer("Oisin", "Mc Laughlin",
"o.mclaughlin2@universityofgalway.ie");

        //Create a Shopping Cart for Customer.
        ShoppingCart cart = new ShoppingCart(customer);

        //Add 3 items with known cost to cart
        Item item1 = new Item("Milk", 2.09, 340);
        Item item2 = new Item("Bread", 1.90, 231);
        Item item3 = new Item("Pancakes", 4.10, 653);

        /*
        item1.setPrice(2.09F);
        item2.setPrice(1.90F);
        item3.setPrice(4.10F);
        */

        cart.addItem(item1);
        cart.addItem(item2);
        cart.addItem(item3);

        //Finalise the cart and create an order
        cart.close();

        //Add a delivery address for the order
        Address billing = new Address("Ard Foyle", "Moville", "F93YRD0",
"Ireland");
        Address delivery = new Address("Dock Street", "Galway", "H91KH32",
"Ireland");

        //Add a payment type
        Payment payment = new Payment(customer, "Visa", 4219987690235476L,
"12/10/23", billing, "Bank Of Ireland");

        //Creating Order object.
        Order order = new Order(cart, customer, delivery);

        //Validate the payment
```

```
        //If successful, email the customer with a success email and the cost of the
purchased items
        //Order order, Address delivery, Address billing
        Email email = new Email(order, delivery, billing, payment, customer);
        email.sendEmail();
    }

    public void transaction2() {
        System.out.println("\n\nScenario 2\n");
        //Create Customer Object
        Customer customer = new Customer("Ciaran", "Gray",
"c.gray3@universityofgalway.ie");

        //Create a Shopping Cart for Customer.
        ShoppingCart cart = new ShoppingCart(customer);

        //The user adds three items.
        Item item1 = new Item("Spuds", 4.40, 672);
        Item item2 = new Item("Butter", 3.20, 389);
        Item item3 = new Item("Jam", 5.80, 391);

        cart.addItem(item1);
        cart.addItem(item2);
        cart.addItem(item3);

        //Requests a display of the shopping cart items and total.
        cart.printItems();
        System.out.println("Cart Total: €" + cart.getTotal() + "\n");

        //Removes one item.
        cart.removeItem(item1);

        //Confirms the cart and makes an order.
        cart.printItems();
        System.out.println("Cart Total: €" + cart.getTotal() + "\n");

        //Setting billing and delivery address.
        Address billing = new Address("BogSide", "Derry", "BT4800R", "Ireland");
        Address delivery = new Address("Dock Street", "Galway", "H91KH32",
"Ireland");
```

```
        //The user submits a payment; however, the payment is not valid.
        Payment payment = new Payment(customer, "Disa", 4219987690235476L,
"11/10/23", billing, "Bank Of Ireland");

        //Creating Order object.
        Order order = new Order(cart, customer, delivery);

        //The user is sent a regret email notifying them that the order was
unsuccessful.
        Email email = new Email(order, delivery, billing, payment, customer);
        email.sendEmail();
    }
}
```

ShoppingCart:

```
//Array packages
import java.util.List;
import java.util.ArrayList;
/**
 * It should be clear how to implement all the methods needed here. Some are
'getter' methods,
others are add/remove methods for adding/removing items from the
ShoppingCart's ArrayList.
The one method that might not be clear is the one called 'close()'. When the
close() method is
called, items cannot be added or removed from the ShoppingCart. If you try to
call the
ShoppingCart's add(Item item) method after the close method is called, it will
print out an error
: "Sorry the shopping cart is closed".
 *
 * @author (Oisin Mc Laughlin - 22441106)
 * @version (v1.0)
 */
public class ShoppingCart
{
    //Class fields
    private int cartId;
    //private String time;
```

```
private ArrayList<Item> cartItems;
private float totalPrice;
private boolean isLocked;
private Customer customer;

//Class constructor that initialises fields and creates a customer object
public ShoppingCart(Customer customer)
{
   this.cartId = cartId;
   //this.time = time;
   this.cartItems = new ArrayList<>();
   this.totalPrice = totalPrice;
   this.isLocked = false;
   this.customer = customer;
}

//Add item to cart method with item object as parameter
public void addItem(Item item) {
   //Checks if cart is locked, if unlocked is false
   if (isLocked == false) {
      //Add the parameter item to cartItems array
      cartItems.add(item);
      //Use get price accessor from item and add value to totalPrice
      totalPrice += item.getPrice();
      //Print message
      System.out.println("Added item to cart:\n" + item + "\n");
   }
   //If cart is locked (set to true), print this
   else {
      System.out.println("Sorry the shopping cart is closed");
   }
}

//Remove item to cart method with item object as parameter
public void removeItem(Item item) {
   //Checks if cart is locked, if unlocked is false
   if (isLocked == false) {
      //Remove the parameter item from the cartItems array
      cartItems.remove(item);
      //Use price accessor from item and minus item price from totalPrice
```

```
            totalPrice -= item.getPrice();
            //Print message
            System.out.println("Removed item from cart:\n" + item + "\n");
        }
        //If cart is locked (set to true), print this
        else {
            System.out.println("Sorry the shopping cart is closed");
        }
    }

    //Accessors to get totalPrice, cartID, customer from object and cartItems
array
    public float getTotal() {
        return totalPrice;
    }

    public int getCartId() {
        return cartId;
    }

    public Customer getCustomer() {
        return customer;
    }

    public ArrayList<Item> getCartItems() {
        return cartItems;
    }

    //Method to print items in cart
    public void printItems() {
        System.out.println("Items in cart:");

        //Loop through the cart and print out each item at index i until i is less
than size
        for (int i = 0; i < cartItems.size(); i++) {
            System.out.println(cartItems.get(i));
        }
    }

    //Method to close cart, sets isLocked to true
```

```
    public void close() {
        isLocked = true;
    }

    //Clear cart method
    public void clear() {
        //Loops through size of cartItems array, removes each element at index i
        for (int i = 0; i < cartItems.size(); i++) {
            cartItems.remove(i);
        }
        //Sets totalPrice to 0 again
        totalPrice = 0.0F;
        //System.out.println("Cart cleared");
    }
}
```

Order:

```
//Array and Random packages
import java.util.ArrayList;
import java.util.ArrayList;
import java.util.Random;
/**
 * The role of the Order class is to take a ShoppingCart object and transfer its items one by one
into itself. It should also take the information about the Customer. Once this is done, the
ShoppingCart should be empty. i.e. its Arraylist should be empty.
Order is an important class in this programme. Look over the lecture notes to see the list of
methods that we provisionally assigned to it. It will have relationships with several other
objects such as ShoppingCart, Payment, Address and Email
 *
 * @author (Oisin Mc Laughlin - 22441106)
 * @version (v1.0)
 */
public class Order
{
    //Class fields
    private ArrayList<Item> orderItems;
```

```
    private Customer customer;
    private Address shippingAddress;
    private Email customerEmail;
    private Payment payment;
    private long orderNum;
    private String orderStatus;
    private String orderDetails;
    private float orderTotal;

    public Order(ShoppingCart shoppingcart, Customer customer, Address
shippingaddress)
    {
        //Initialises fields as well as creates customer object,
        //calling makeOrderNum method, transferItems method with shoppingcart
object as paramter,
        //setting orderstatus to "Incomplete", and setting orderDetails to the items
as a string,
        //setting orderTotal as the result of the calcTotalPrice() method.
        this.orderItems = new ArrayList<Item>();
        this.customer = customer;
        this.shippingAddress = shippingAddress;
        this.orderNum = makeOrderNum();
        this.orderStatus = "Incomplete";
        transferItems(shoppingcart);
        this.orderDetails = orderItems.toString();
        this.orderTotal = calcTotalPrice();
    }

    //Transfer items method
    public void transferItems(ShoppingCart shoppingcart) {
        //ArrayList<Item> orderItems = shoppingcart.cartItems;

        //Iterates through each of the items in shopping cart
        for (Item item : shoppingcart.getCartItems()) {
            //Adds each item to the orderItems array
            orderItems.add(item);
        }
        //Calls the shoppingcart clear method
        shoppingcart.clear();
    }
```

```
    //Random number generator to create customer id between 1 and 99.
    public long makeOrderNum() {
        int min = 1;
        int max = 99;

        // Create an instance of the Random class
        Random random = new Random();

        // Generate a random number within the specified range
        long randomOrderNum = min + random.nextInt(max - min + 1);

        return randomOrderNum;
    }

    //Mutator to set orderStatus to the parameter
    public void setOrderStatus(String status) {
        this.orderStatus = status;
    }

    //Method to calculate the total price in order
    public float calcTotalPrice() {
        float totalP = 0.0F;

        //Loops through each of the orderItems and gets the price by calling the
item getPrice() method
        for (Item item : orderItems) {
            //Adds the item price to totalP and returns it
            totalP += item.getPrice();
        }
        return totalP;
    }

    //Accessors to get orderNum, orderDetails and orderTotal
    public double getOrderNum() {
        return orderNum;
    }
    public String getOrderDetails() {
        return orderDetails;
    }
```

```
    public float getOrderTotal() {
        return orderTotal;
    }
}
```

Address:

```
/**
 * The role of the Address class is to hold the address fields of the Customer's
address: e.g. street,
city, zip, country. A customer may have two Address objects associated with
them - a billing
address and a delivery address. You will need methods to set and get the
information in each
Address object.
 *
 * @author (Oisin Mc Laughlin - 22441106)
 * @version (v1.0)
 */
public class Address
{
    //Address fields
    private String street;
    private String city;
    private String zip;
    private String country;

    //Constructor to initialise each of the fields
    public Address(String street, String city, String zip, String country)
    {
        this.street = street;
        this.city = city;
        this.zip = zip;
        this.country = country;
    }

    //Accesor and mutator for each field
    public String getStreet() {
        return street;
    }
```

```
    public void setStreet(String street) {
        this.street = street;
    }

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

    public String getZip() {
        return zip;
    }
    public void setZip(String zip) {
        this.zip = zip;
    }

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

    //Returns it all as a string with each field displayed.
    @Override
    public String toString() {
        String out = "Street: " + street + "\nCity: " + city + "\nZip: " + zip +
"\nCountry: " + country;
        return out;
    }
}
```

Payment:

```
/**
 * The Payment class holds the following pieces of information:
• customer;
• credit card type;
```

• credit card number;
• date;
• address;
• credit card bank name
 *
 * @author (Oisin Mc Laughlin - 22441106)
 * @version (v1.0)
 */
```
public class Payment
{
   //Class fields
   private Customer customer;
   private String cardType;
   private long cardNum;
   private String date;
   private Address address;
   private String bankName;

   //Payment constructor, initialising fields
   public Payment(Customer customer, String cardType, long cardNum, String
date, Address address, String bankName)
   {
      this.customer = customer;
      this.cardType = cardType;
      this.cardNum = cardNum;
      this.date = date;
      this.address = address;
      this.bankName = bankName;
   }

   //cardType accessor
   public String getCardType() {
      return cardType;
   }

   //isValid method with cardType string as a parameter
   public boolean isValid(String cardType) {
      //converts parameter to lowercase
      String check = cardType.toLowerCase();
```

```
        //returns true if parameter is equal to "visa" or "mastercard"
        return check.equals("visa") || check.equals("mastercard");
    }

    //Converts all of the details to a string and returns it
    @Override
    public String toString() {
        String out = "Name: " + customer.getFirstName() + " " +
customer.getSurName() + "\nType: " + cardType + "\nNumber: " +cardNum +
"\nDate: " + date + "\nAddress: " + address + "\nBank: " + bankName;
        return out;
    }
}
```

Email:

```
/**
 * The role of the Email object is to send (you are required to just printout on
the screen) an email
message to the customer. If the Payment has been successful, then it will be a
positive message
giving the order number, the order details, the delivery, and billing addresses. If
the Payment
has been unsuccessful, then the message explains that the order has not been
made. In either
case, the customer must be addressed by their first name and the email
address is their email
address (from the Customer object)
 *
 * @author (Oisin Mc Laughlin - 22441106)
 * @version (v1.0)
 */
public class Email
{
    //Class fields
    private Order order;
    private Customer customer;
    private Address delivery;
    private Address billing;
    private Payment payment;
```

```
   //Constructor initialising each field
   public Email(Order order, Address delivery, Address billing, Payment
payment, Customer customer)
   {
      this.order = order;
      this.delivery = delivery;
      this.billing = billing;
      this.payment = payment;
      this.customer = customer;
   }

   //sendEmail method
   public void sendEmail() {
      //Calls isValid method from payment with the cardType as a parameter
      if (payment.isValid(payment.getCardType()) == true) {
         //If the card Type is valid (true)
         //Set orderStatus to processing
         order.setOrderStatus("Processing");
         //Print an email with customer details, order number, item details, total
and addresss.
         System.out.println("\nEmail Address: " + customer.getEmail() +
"\n\nOrder Number: " + order.getOrderNum() + "\nOrder Details: " +
order.getOrderDetails() + "\nTotal: " + order.getOrderTotal() + "\n\nId: " +
customer.getId() + "\nName: " + customer.getFirstName() + " " +
customer.getSurName() + "\n\nDelivery Address: \n" + delivery +
"\n\nPayment Details:\n" + payment + "\n\nBilling Address: \n" + billing +
"\n\nThank you for shopping with Dunnes Stores\n");
      }
      else {
         //If payment was not valid, email customer saying it wasn't valid
         System.out.println("\nEmail Address: " + customer.getEmail() + "\n\nWe
regret to inform you that your order was unsuccessful\nPayment has not been
processed\n\nPlease contact Dunnes Stores Support for help.\n");
      }
   }
}
```

Question 3:

After running the code and fixing issues the testing turned out how I expected it to go.

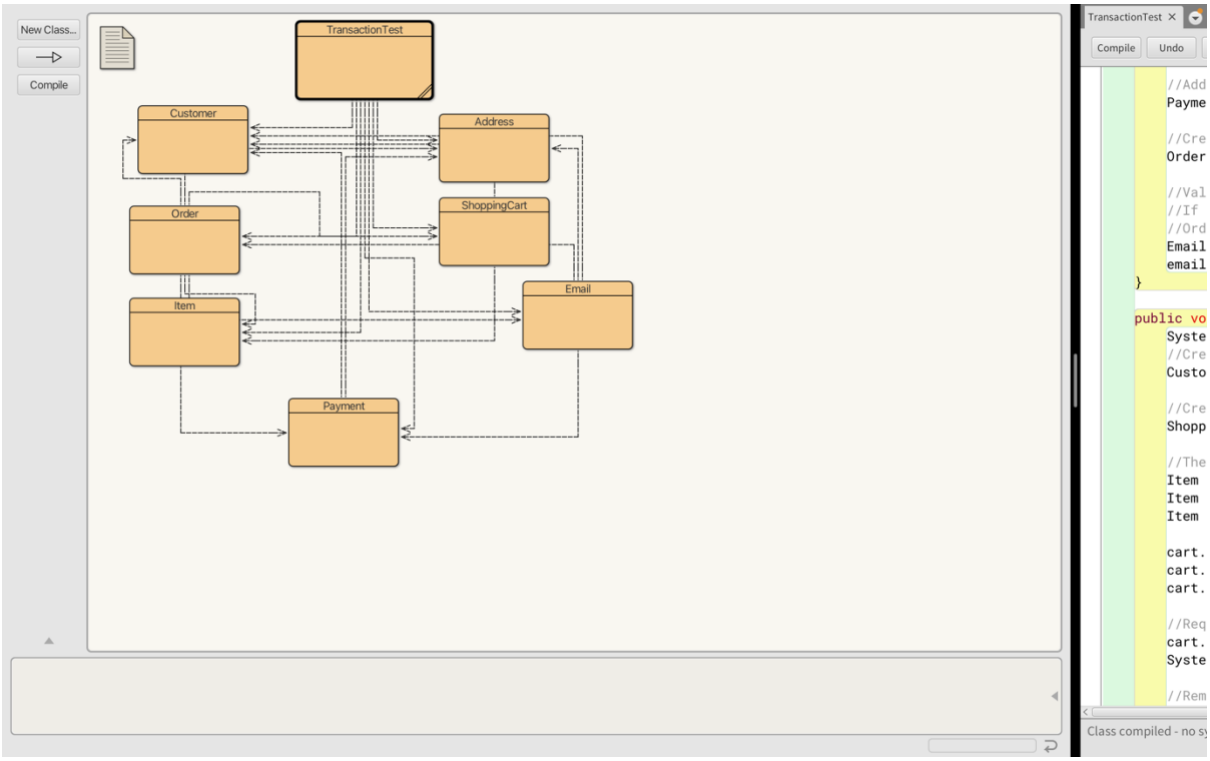
```
Scenario 1

Added item to cart:
Item Id: 340    Milk    Price 2.09

Added item to cart:
Item Id: 231    Bread   Price 1.9

Added item to cart:
Item Id: 653    Pancakes        Price 4.1


Email Address: o.mclaughlin2@universityofgalway.ie

Order Number: 7.0
Order Details: [Item Id: 340    Milk    Price 2.09      , Item Id: 231  Bread   Price 1.9       , Item Id: 653  Pancakes        Price 4.1       ]
Total: 8.09

Id: 14.0
Name: Oisin Mc Laughlin

Delivery Address:
Street: Dock Street
City: Galway
Zip: H91KH32
Country: Ireland

Payment Details:
Name: Oisin Mc Laughlin
Type: Visa
Number: 4219987690235476
Date: 12/10/23
Address: Street: Ard Foyle
City: Moville
Zip: F93YRD0
Country: Ireland
Bank: Bank Of Ireland

Can only enter input while your program is running
```

```
Country: Ireland
Bank: Bank Of Ireland

Billing Address:
Street: Ard Foyle
City: Moville
Zip: F93YRD0
Country: Ireland

Thank you for shopping with Dunnes Stores



Scenario 2

Added item to cart:
Item Id: 672    Spuds   Price 4.4

Added item to cart:
Item Id: 389    Butter  Price 3.2

Added item to cart:
Item Id: 391    Jam     Price 5.8

Items in cart:
Item Id: 672    Spuds   Price 4.4
Item Id: 389    Butter  Price 3.2
Item Id: 391    Jam     Price 5.8
Cart Total: €13.4

Removed item from cart:
Item Id: 672    Spuds   Price 4.4

Items in cart:
Item Id: 389    Butter  Price 3.2
Item Id: 391    Jam     Price 5.8
Cart Total: €9.0


Email Address: c.gray3@universityofgalway.ie
```

Can only enter input while your program is running

```
Country: Ireland

Thank you for shopping with Dunnes Stores



Scenario 2

Added item to cart:
Item Id: 672    Spuds   Price 4.4

Added item to cart:
Item Id: 389    Butter  Price 3.2

Added item to cart:
Item Id: 391    Jam     Price 5.8

Items in cart:
Item Id: 672    Spuds   Price 4.4
Item Id: 389    Butter  Price 3.2
Item Id: 391    Jam     Price 5.8
Cart Total: €13.4

Removed item from cart:
Item Id: 672    Spuds   Price 4.4

Items in cart:
Item Id: 389    Butter  Price 3.2
Item Id: 391    Jam     Price 5.8
Cart Total: €9.0


Email Address: c.gray3@universityofgalway.ie

We regret to inform you that your order was unsuccessful
Payment has not been processed

Please contact Dunnes Stores Support for help.
```

Can only enter input while your program is running


New Class...
Compile
TransactionTest
Customer
Address
Order
ShoppingCart
Item
Email
Payment
TransactionTest
Compile
Undo
Class compiled - no s