My approach,

My approach to this assignment was to make most things revolve around the customer,

to make the ordering process easy.

I decided that every customer will have their own shopping cart (not every shopping cart having a customer),

Every time a new customer instance is created we provide it with its own shopping cart,

items can be added and removed directly through the customer class to help make the code

easier to understand and easier to read. I did this by making an instance of the shopping cart class inside of the customer class and gave the customer some methods to control the cart functionality

I wanted to make the Item class super basic as I didn't see a need to complicate it

It is a simple class that takes in a name and a price in its constructor and has 2 getter methods for them

But I also wanted it to be impossible for an item to be free or a negative value so I fixed that using a ternary operator

For the address class I understood there was a requirement for the ability to add both the shipping and billing addresses,

however most people (atleast me anyway) don't use a different billing address, I added 2 constructors, one where it

only takes in a shipping address and sets the billing address as the same and then the other constructor that takes

in another address. There is also a very basic address checker that just checks the length of the address to ensure it

is atleast somewhat valid.

Then there was the payment class. In here I take in the card type, card number and the expiry date.

I check if each of them is valid. The card types accepted are mastercard and visa, it will not accept any other ones,

the card number must be 16 digits, and the expiry date must be in the future. I implemented that using the LocalDate library and string and array manupilated the dates

Then we have the most "important" class. The order class. In here I ensure that all the data we collected is valid,

I print out the customers items and its total price. I then use the email class to send of an email to the customer

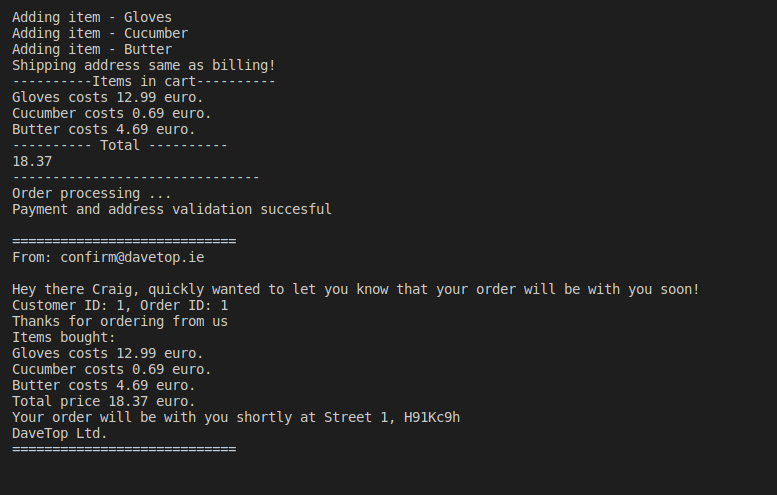
depending on their scenario. Such as all good and the order went through or whether something went wrong and their

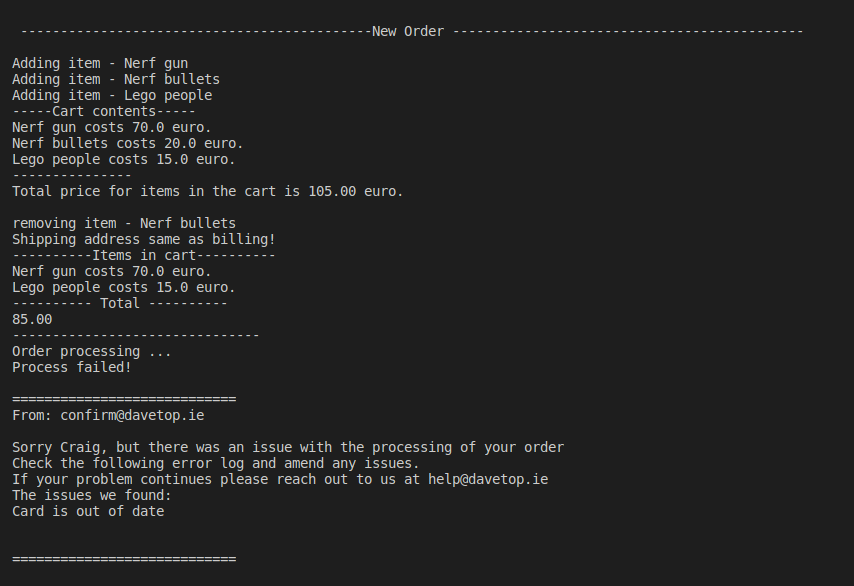
order couldn't process for whichever reason as well as mentioning said reasons in the "email".

I also tested a few extra scenarios outside of the scope of the assignment to provide extra detail on some of the functionality

of the program.

Screenshots of output for scenario 1 and 2, other scenarios can be seen at the bottom



TransactionTest.java

/\*

Author: Dawid Szczesny

ID: 21300293

\*/

public class TransactionTest {

// Main method

// Starting point of the program

public static void main(String[] args) {

TransactionTest test = new TransactionTest();

// Test methods, to test different scenarios

test.transactionTest1();

test.transactionTest2();

test.transactionTest3();

test.transactionTest4();

test.transactionTest5();

test.transactionTest6();

}

/\*

\* Scenario One

\* From the Lecture notes

\* Customer adds 3 items, and orders successfully

\*/

public void transactionTest1() {

System.out.println(

" --------------------------------------------New Order --------------------------------------------\n");

// Creates our customer. We set their name, last name, and email address.

Customer customer = new Customer("Craig", "Smith", "craig.smith@gmail.com");

// We create 3 items. We give their name and their price.

Item item1 = new Item("Gloves", 12.99F);

Item item2 = new Item("Cucumber", .69F);

Item item3 = new Item("Butter", 4.69F);

// We add the 3 items into our cart.

customer.addToCart(item1, item2, item3);

Address address = new Address("Street 1", "H91Kc9h");

customer.setAddress(address);

// We make a payment method, that takes the cart type either visa or mastercard,

// the card number and the expiry date.

Payment payment = new Payment("visa", "4587985698547854", "01/23");

// We create an order instance

// We begin the process and then confirm it

Order order = new Order(customer, payment);

order.process();

order.confirm();

}

/\*

\* Scenario 2

\* From Assignment 2

\* Customer adds 3 items, displays them and the total price

\* The the customer removes one item and continues to the payment method

\* In this scenario, the order fails, as the card is out of date by one month

\*/

public void transactionTest2() {

System.out.println(

" --------------------------------------------New Order --------------------------------------------\n");

Customer customer = new Customer("Craig", "Smith", "craig.smith@gmail.com");

Item item1 = new Item("Nerf gun", 70.0f);

Item item2 = new Item("Nerf bullets", 20.f);

Item item3 = new Item("Lego people", 15.f);

customer.addToCart(item1, item2, item3);

customer.displayCartContents();

customer.displayCartTotal();

customer.removeFromCart(item2);

customer.lockCart();

Address address = new Address("Street 1", "H91Kc9h");

customer.setAddress(address);

// create payment

// The card in this scenario is out of date and hence will fail the order.

Payment payment = new Payment("visa", "4587985698547854", "08/22");

Order order = new Order(customer, payment);

order.process();

order.confirm();

}

/\*

\* Scenario 3

\* Custom

\* We test the implementation of seperate billing addresses

\* We also check if mastercard is a valid card type

\*/

public void transactionTest3() {

System.out.println(

" --------------------------------------------New Order --------------------------------------------\n");

// The address constructor allows for us to omit the billing addresses which

// entails that it is the same as the shipping address

Customer customer = new Customer("Dave", "Thompson", "d.thompson@nuigalway.ie");

Item item1 = new Item("Keyboard", 155.0F);

Item item2 = new Item("Almond Milk", 2.55F);

Item item3 = new Item("Dental Floss", 1.67F);

customer.addToCart(item1, item2);

customer.lockCart();

customer.addToCart(item3);

customer.displayCartContents();

Address address = new Address("41 new road, New Ross, Co.Down", "F45K289", "Eyre Square, Galway", "H91ED63");

customer.setAddress(address);

Payment payment = new Payment("mastercard", "4587985698547854", "01/23");

Order order = new Order(customer, payment);

order.process();

order.confirm();

}

/\*

\* Scenario 4

\* Custom

\* We test what would happen if we gave an invalid email address. (There is no

\* '@' symbol in it)

\* We also give an invalid address, as the shippingAddress can't be one number

\* and the Eircode most be valid

\* We also set the card expiry date to a negative number.

\*/

public void transactionTest4() {

System.out.println(

" --------------------------------------------New Order --------------------------------------------\n");

Customer customer = new Customer("Dave", "Thompson", "d.thompsonnuigalway.ie");

Item item1 = new Item("Keyboard", 155.0F);

Item item2 = new Item("Almond Milk", 2.55F);

customer.addToCart(item1, item2);

customer.displayCartContents();

customer.displayCartTotal();

customer.lockCart();

Address address = new Address("1", "8291", "Eyre Square, Galway", "123");

customer.setAddress(address);

customer.setAddress(address);

Payment payment = new Payment("MyMasfer", "12311121", "-01/23");

Order order = new Order(customer, payment);

order.process();

order.confirm();

}

/\*

\* Scenario 5

\* Custom

\* Testing wrong card details

\*/

public void transactionTest5() {

System.out.println(

" --------------------------------------------New Order --------------------------------------------\n");

Customer customer = new Customer("Dave", "Thompson", "d.thompson@nuigalway.ie");

Item item1 = new Item("Typewriter", -5.0F);

Item item2 = new Item("Baked Potatoes", 2.55F);

Item item3 = new Item("Toilet Paper", 1.67F);

customer.addToCart(item1, item2, item3);

customer.displayCartContents();

customer.displayCartTotal();

customer.lockCart();

Address address = new Address("The Occupant, House Master", "H92 89XQ");

customer.setAddress(address);

Payment payment = new Payment("MyMasfer", "12311121", "12/23");

Order order = new Order(customer, payment);

order.process();

order.confirm();

}

/\*

\* Scenario 6

\* Custom

\* Testing if there were no items in the cart

\* We then add one item to ensure its working

\*/

public void transactionTest6() {

System.out.println(

" --------------------------------------------New Order --------------------------------------------\n");

Customer customer = new Customer("Dave", "Thompson", "d.thompson@nuigalway.ie");

customer.displayCartContents();

customer.displayCartTotal();

Address address = new Address("The Occupant, House Master", "H92 89XQ");

customer.setAddress(address);

customer.lockCart();

Payment payment = new Payment("visa", "4587452145632145", "12/23");

Order order = new Order(customer, payment);

order.process();

order.confirm();

Item item1 = new Item("Banana", 0.5F);

customer.addToCart(item1);

order.process();

order.confirm();

}

}

ShoppingCart.class

import java.util.ArrayList;

/\*

ShoppingCart class

Holds the items, has the ability to add, remove, and display each item

Method to display the contents of the cart and the total price of it

\*/

public class ShoppingCart {

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

private float totalPrice = 0.F;

private boolean locked = false;

// Adds items to the cart

// Has the ability to add mulitply Items add a time

public void add(Item... items) {

if(locked){

System.out.println("Cart is locked, cannot add items");

return;

}

for (Item item : items) {

System.out.println("Adding item - " + item.getName());

itemArray.add(item);

totalPrice += item.getPrice(); // Adds up price as we add items

}

}

// locks the cart so no more items can be added or removed

public void lock(){

locked = true;

}

// removes items

public void remove(Item item) {

if(locked){

System.out.println("Cart is locked, cannot remove items");

return;

}

System.out.println("removing item - " + item.getName());

itemArray.remove(item);

totalPrice -= item.getPrice(); // keeps track of price even when removing items

}

// displays the contents of the cart

public void displayContents() {

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

for (Item item : itemArray) {

System.out.printf("%s\n", item);

}

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

}

// display total price

public void displayTotal() {

System.out.printf("Total price for items in the cart is %.2f euro.\n\n", getTotalPrice());

}

//setters

public void setTotal(int total) {

totalPrice = total;

}

//getters

public float getTotalPrice() {

return totalPrice;

}

public ArrayList<Item> getItemArray() {

return itemArray;

}

}

Order.class

import java.util.ArrayList;

/\*

Order class

Check customer details

\*/

public class Order {

private static int orderNumber = 0; // static variable to make each order number unique

private Customer customer;

private Payment payment;

private ArrayList<Item> orderArray;

private float totalPrice = 0;

public Order(Customer customer, Payment payment) {

this.customer = customer;

this.payment = payment;

// for each new order we increment the order number to make it a unique order

// orderNumber is a static variable which allows it to retain its value.

orderNumber++;

}

public void process() {

// process order code

// copying the cart to our order cart array

orderArray = new ArrayList<Item>(customer.getCart().getItemArray());

// can also be done using an iterative array and using orderArray.add(item)

// we clear the customer's cart

customer.getCart().getItemArray().clear();

// keep the total price and then reset it for the customer

totalPrice = customer.getCart().getTotalPrice();

customer.getCart().setTotal(0);

}

// confirm order method

public void confirm() {

// ensures that there is something in our cart

if (orderArray.size() <= 0) {

System.out.println("Your cart is currently empty! Consider adding a few items and try again!\n");

return;

}

/\*

\* This method will confirm the items that the customer is about to purchase.

\*/

// prints out the items

System.out.println("----------Items in cart----------");

orderArray.forEach((item)->System.out.println(item)); // prints out each item in the array

// prints the total

System.out.printf("---------- Total ----------\n%.2f\n", totalPrice);

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

System.out.println("Order processing ... ");

// We check that the customer has a valid email address before progressing

if (customer.getValidEmail()) {

// If they do we send them on a confirmation email, on whether their order went through or not

new Email(getStatus(), this);

return;

}

// We should only run this code when the email is invalid

System.out.println("Order processing failed due to invalid email\n\n");

return;

}

// getters

public int getOrderNumber() {

return orderNumber;

}

public ArrayList<Item> getOrderItems() {

return orderArray;

}

public float getTotalPrice() {

return totalPrice;

}

public boolean getStatus() {

// this will check if all the details are correct.

return payment.checkPaymentDetails() && customer.getAddress().getValidation();

}

public Customer getCustomer() {

return customer;

}

public Payment getPaymentDetails() {

return payment;

}

}

Address.class

/\*

Address Class

Responsible for holding and validating the address of the customer

\*/

public class Address {

private boolean addressValid = false;

private String shippingAddress;

private String shippingEircode;

private String billingAddress;

private String billingEircode;

// An error log string. This will be sent to the customer via email

private String invalidAddressExcuse = "";

// Address constructor for same shipping and Billing addresses

public Address(String shippingAddress, String shippingEircode) {

this.shippingAddress = billingAddress = shippingAddress;

this.shippingEircode = billingEircode = shippingEircode;

// checks if the address is valid

isValid();

// Printing to customer that we are using the same address for both the shipping

// and billing

System.out.println("Shipping address same as billing!");

}

// Address constructor for different shipping and billing addresses

public Address(String shippingAddress, String shippingEircode, String billingAddress, String billingEircode) {

this.shippingAddress = shippingAddress;

this.shippingEircode = shippingEircode;

this.billingAddress = billingAddress;

this.billingEircode = billingEircode;

// checks if the address is valid

isValid();

}

// getters

public String getInvalidationExcuse() {

return invalidAddressExcuse;

}

public boolean getValidation() {

return addressValid;

}

// validation method. This method is set to private to ensure no external class can run this method as there is no need to

private void isValid() {

// checks if the eircode is composed of 7 characters; ignoring the spaces, and

// check if the address itself is greater than 5 characters

addressValid = shippingEircode.replace(" ", "").length() == 7

&& billingEircode.replace(" ", "").length() == 7

&& shippingAddress.length() > 5 && this.billingAddress.length() > 5;

if (!addressValid) {

invalidAddressExcuse += "Invalid Address";

}

}

// Override method, used for when printing address

@Override

public String toString() {

return shippingAddress + ", " + shippingEircode;

}

}

Payment.class

import java.time.LocalDate; // to get current date

/\*

Payment class

Takes card type, card number, expiry date

\*/

public class Payment {

private String cardType;

private String cardNumber;

private String expiryDate;

private String inValidExcuse = "";

// constructor

public Payment(String cardType, String cardNumber, String expiryDate) {

this.cardType = cardType.toLowerCase(); // sets card type to lowercase for simplicity

this.cardNumber = cardNumber;

this.expiryDate = expiryDate;

}

// checks all payment details

// ensures card number is 16 digits

// that the card is either visa or mastercard

// and that the current date is before the expiry datte

public boolean checkPaymentDetails() {

if (cardNumber.length() != 16) {

inValidExcuse += "Card number not 16 digits\n";

}

if ((cardType != "visa" && cardType != "mastercard")) {

inValidExcuse += "Card type invalid\n";

}

// Check Expiry Date against current date

// gets the current date and converts it to a string array of its year, month,

// and day

// format progress

// YYYY-MM-DD as Date data type -> YYYY-MM-DD as String data type -> "YYYY",

// "MM","DD" as String array

String[] currentDate = LocalDate.now().toString().split("-");

// we parse and split our string array taking the year and month and

// reorganising it to MMYY as a number

int cd = Integer.valueOf(currentDate[0].substring(2) + currentDate[1]);

// we make sure there are no negative dates by deleting the minus sign

// we parse and split our string into the desired format

expiryDate = expiryDate.replace("-", "");

int exDate = Integer.valueOf(expiryDate.substring(3) + expiryDate.substring(0, 2));

// we compare the dates

if (cd > exDate) {

inValidExcuse += "Card is out of date\n";

}

// checks if the explaination for failure is more than 25 characters

if (inValidExcuse.length() > 0) {

System.out.println("Process failed!");

return false;

}

return true;

}

// getter

public String getInvalidationExcuse() {

return inValidExcuse;

}

}

Email.class

/\*

\* Email class

\* Ran from the order class

\* Primarily a print class

\*/

public class Email {

public Email(boolean success, Order order) {

// We check if all the details are correct

if (!success) {

// order unsuccessful email

// this code runs if there is some error with either the address or the payment details

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

System.out.println("From: confirm@davetop.ie\n");

System.out.printf("Sorry %s, but there was an issue with the processing of your order\n",

order.getCustomer().getName());

System.out.println(

"Check the following error log and amend any issues.\nIf your problem continues please reach out to us at help@davetop.ie");

System.out.println("The issues we found:\n" + order.getPaymentDetails().getInvalidationExcuse()

+ order.getCustomer().getAddress().getInvalidationExcuse() + "\n");

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

return;

}

// order sucessful email

// this code runs when everything is fine and the code works perfectly.

System.out.println("Payment and address validation succesful\n");

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

System.out.println("From: confirm@davetop.ie\n");

System.out.printf("Hey there %s, quickly wanted to let you know that your order will be with you soon!\n",

order.getCustomer().getName());

System.out.println("Customer ID: " + order.getCustomer().getID() + ", Order ID: " + order.getOrderNumber());

System.out.println("Thanks for ordering from us");

System.out.println("Items bought: ");

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

System.out.println(item);

}

System.out.printf("Total price %.2f euro.\n", order.getTotalPrice());

System.out.println("Your order will be with you shortly at " + order.getCustomer().getAddress());

System.out.println("DaveTop Ltd.");

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

}

}

Item.class

/\*

\* Item class

\* Just has the item name and its price

\*/

public class Item{

private String name;

private float price;

public Item(String name, float price){

this.name = name;

// We check to see if the price is a positive number or non-zero, if it isn't we set it to one cent.

this.price = price > 0 ? price : 0.01F;

}

// getters

public String getName(){

return name;

}

public float getPrice(){

return price;

}

// Override print method so when we print we'll get the item name and cost

@Override

public String toString(){

return getName() + " costs " + getPrice() + " euro.";

}

}

Customer.class

/\*

Customer class, responsible for holding info about the customer

\*/

public class Customer {

private String firstName;

private String surName;

private String email;

private boolean validEmail = false;

private Address address;

private ShoppingCart cart;

// we create a static variable to ensure that it does not restart count after

// every new customer, to ensure no two customers have the same customerID

private static int customerID = 0;

public Customer(String firstName, String surName, String email) {

this.firstName = firstName;

this.surName = surName;

this.email = email;

validEmail = this.email.contains("@");

customerID++;

cart = new ShoppingCart();

}

// setters

// sets the address to customer

public void setAddress(Address address) {

this.address = address;

}

public void addToCart(Item ... items){

cart.add(items);

}

public void removeFromCart(Item item){

cart.remove(item);

}

// getters

// checks if the customer's email has the correct format.

public boolean getValidEmail() {

return validEmail;

}

public ShoppingCart getCart() {

return cart;

}

public Address getAddress() {

return address;

}

public String getName() {

return firstName;

}

public String getSureName(){

return surName;

}

public int getID() {

return customerID;

}

// getters for cart info

public void displayCartContents(){

cart.displayContents();

}

public void displayCartTotal(){

cart.displayTotal();

}

}

