Object-Oriented Programming Assignment 2 – Ordering System

Output scenario 1

A screenshot of a computer

Description automatically generated with medium confidenceOutput scenario 2Table

Description automatically generated

# TransactionTest

This is our main class where we test our two different scenarios of a successful payment and an unsuccessful payment. In this class we call our methods from connecting classes. We use the new keyword to instantiate the class by allocating desired memory for the associated new objects for each customer. We set the prices of our items using the setPrice() method and add them using our add.item() method.

/\*\*

\* Gavin Skehan

\* Assignment 2

\* 14/10/22

\*/

public class TransactionTest

{

public static void main(String[] args){

TransactionTest test = new TransactionTest();

test.transaction1();

test.transaction2();

}

public void transaction1(){

// 1. create new customer

Customer customer = new Customer("Niamh", "O'Leary", "niamhol@zmail.com");

// create shopping cart for the customer

ShoppingCart cart = new ShoppingCart(customer);

// 2. Create and add items

Item item1 = new Item("toothbrush");

Item item2 = new Item("dog food");

Item item3 = new Item("dougnuts");

// 3. add items with known cost

item1.setPrice(2f);

item2.setPrice(3.50f);

item3.setPrice(5.25f);

cart.addItem(item1);

cart.addItem(item2);

cart.addItem(item3);

// 4. Create a new order

cart.close();

Order order = new Order(cart, customer);

// 5. Create and Address

Address billingAddress = new Address("16 downstreet", "Portloaise",

"Co.Laois", "R23 UX5V");

Address shippingAddress = new Address("16 downstreet", "Portlaoise",

"Co.Laois", "R23 UX5V");

order.setShippingAddress(shippingAddress);

// 6. Add a Payment type

Payment payment = new Payment(billingAddress, "mastercard",

"1234 5678 9876 5432", "01/01/2023");

order.setPayment(payment);

// 8. Email

Email email = new Email(order);

email.send();

}

public void transaction2(){

Customer customer = new Customer("Cathal", "Farrel",

"cfarrel@zmail.com");

ShoppingCart cart = new ShoppingCart(customer);

Item item1 = new Item("beer");

Item item2 = new Item("popcorn");

Item item3 = new Item("movie");

item1.setPrice(18f);

item2.setPrice(2.75f);

item3.setPrice(5.25f);

cart.addItem(item1);

cart.addItem(item2);

cart.addItem(item3);

System.out.println(cart);

cart.removeItem(item3);

cart.close();

Order order = new Order(cart, customer);

Address billingAddress = new Address("6 Crossroads", "Galway",

"Co.Galway", "U56 YY23");

Address shippingAddress = new Address("6 Crossroads", "Galway",

"Co.Galway", "U56 YY23");

order.setShippingAddress(shippingAddress);

Payment payment = new Payment(billingAddress, "Revolut",

"1111 2222 3333 4444", "10/11/2024");

order.setPayment(payment);

Email email = new Email(order);

email.send();

}

}

# ShoppingCart

In this class we import various libraries to support our class. We use the SimpleDateFormat() to print out the current date. We use this keyword to reference the current object inside the method and the new keyword to make an instance of the class (new ArrayList<>()). We create an add method to add items, removes method to remove items, our close method finalizes the cart and clear method resets the cart. We use this class to display the cart and the price of the cart. @Override ensures the method returns different values with regard to new customers.

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

public class ShoppingCart

{

private static int prevId = 0;

private final int id;

private float total;

private Customer customer;

private ArrayList<Item> items;

private boolean closed = false;

public ShoppingCart(Customer customer)

{

Date curDate = new Date( );

SimpleDateFormat ft =

new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz"); // date format

System.out.println("Current Date: " + ft.format(curDate)); // prints curDate

this.customer = customer;

items = new ArrayList<>();

// counts customers in order

id = prevId + 1; // tracks customers

prevId = id;

}

public void addItem(Item item){

if(closed) {

System.out.println("Shopping cart is closed");

System.out.println("You cant add more items");

return;

}

else{

items.add(item); // adds item

total += item.getPrice();

}

}

public void removeItem(Item item){

if(closed){

System.out.println("Shopping cart is closed");

System.out.println("You cant remove items");

return;

}

else{

items.remove(item); // removes item

total -= item.getPrice();

}

}

public float getTotal(){

return total;

}

public void close(){

System.out.println("You closed your shopping cart");

closed = true;

}

public void clear(){

items = new ArrayList<>();

total = 0;

}

public ArrayList<Item> getItems(){

return items;

}

@Override

public String toString(){

String out = "\n" +

"Shopping Cart\n" +

"Cart No: #" + id + "\n" +

"Customer: " + customer.getFirstName() + " " + customer.getSurname() +

" Items:\n";

for (Item item : items) {

out += "\t- " + item + "\n";

}

out += "Total: €" + String.format("%.02f", total) + "\n";

return out;

}

}

# Customer

This keyword refers to the current class instance variables. This class is used to retrieve the details about the customers (Id, full name, address) using the specific methods.

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;

customerId = makeCustomerId();

}

public long getId(){

return customerId;

}

public String getFirstName(){

return firstName;

}

public String getSurname(){

return surname;

}

public String getEmailAddress() {

return emailAddress;

}

private long makeCustomerId() {

return (long)Math.random() \* 99999999999999L; // L needed for large number

}

}

# Item

In this class we get all the item details (this.name, makeId(), setPrice) and return it as a string using the toString() method to be printed in the output.

public class Item

{

private String name;

private float price;

private static int prevId = 0;

private int id;

public Item(String itemName){

this.name = name;

id = makeId();

}

public Item (String name, float price) {

this.name = name;

this.price = price;

id = makeId();

}

public int makeId(){

int id = prevId + 1;

prevId = id;

return id;

}

public void setPrice(float price){

this.price = price;

}

public float getPrice(){

return price;

}

public String toString(){ // returns a string representation of the method

String out = "Item ID: " + id + "\t" + name + "\tPrice: " + price;

return out;

}

}

# Order

We call our order class once the shopping cart has closed and the order is successful. In the class we finalize the order by setting the address (this.shippingAddress) and check if the card payment is valid setPayment(). If it is valid we call our toSting method() once again with the price of the items.

import java.util.ArrayList;

public class Order

{

private static int prevId = 0;

private int id;

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

private Address shippingAddress;

private Payment payment;

private float total;

private Customer customer;

private boolean paymentSuccessful = false;

public Order(ShoppingCart cart, Customer customer)

{

this.customer = customer;

items.addAll(cart.getItems());

total = cart.getTotal();

cart.clear();

id = prevId + 1;

prevId = id;

}

public void setShippingAddress(Address shippingAddress) {

this.shippingAddress = shippingAddress;

}

public void setPayment(Payment payment) {

if (!payment.validated()) {

// Your payment is not valid

return;

}

else{

this.payment = payment;

this.paymentSuccessful = true;

}

}

@Override

public String toString() {

String out = "Order Items: \n";

for (Item item : items) {

out += "\t- " + item + "\n";

}

out += "Total: €" + String.format("%.02f", total);

return out;

}

public Customer getCustomer() {

return customer;

}

public boolean isPaymentSuccessful() {

return paymentSuccessful;

}

public int getId() {

return id;

}

public Address getShippingAddress() {

return shippingAddress;

}

public Payment getPayment() {

return payment;

}

}

# Payment

In this class we set the billing address and also we have a validate check on the card, validateCardType() and validateCardNumber(). We use the Boolean method to return the validation results and if valid we return the billing address.

public class Payment

{

private Address billingAddress;

private String cardType;

private String cardNum;

private boolean valid = false;

public Payment(Address billingAddress, String cardType, String cardNum, String expDate){

this.billingAddress = billingAddress;

this.cardType = cardType;

this.cardNum = cardNum;

valid = validate();

}

private boolean validateCardType() {

String cType = cardType.toLowerCase(); // coverts a given string to lowercase eg M==m

String[] validCardTypes = {"mastercard", "visa"}; // valid car types

for (String validType : validCardTypes) {

if (cType.equals(validType)) {

return true;

}

}

return false;

}

private boolean validateCardNumber() {

String cardNumber = cardNum.replaceAll(" ", ""); // replaces a sequence of characters

if (cardNumber.length() != 16) { // card number must have 16 digits

return false;

}

return true;

}

private boolean validate() {

if (!validateCardType()) {

// Invalid card type

return false;

}

if (!validateCardNumber()) {

// Invalid card number

return false;

}

return true;

}

public boolean validate() {

return valid;

}

public Address getBillingAddress() {

return billingAddress;

}

}

# Address

We use methods in this class to set the address of the customer overriding any previous addresses to match the current customer.

public class Address

{

private String street;

private String city;

private String county;

private String zip;

public Address(String street, String city, String county, String zip)

{

this.street = street;

this.city = city;

this.county = county;

this.zip = zip;

}

public void setStreet(String street) { // sets address

this.street = street;

}

public void setCity(String city) {

this.city = city;

}

public void setCounty(String county) {

this.county = county;

}

public void setZip(String zip) {

this.zip = zip;

}

@Override // defines the specific address to the customer

public String toString() {

return

"\tStreet: " + street + "\n" +

"\tCity: " + city + "\n" +

"\tCountry: " + county + "\n" +

"\tZIP: " + zip;

}

}

# Email

In this class we use the if else statements to send the appropriate emails with regard to the payment being successful or unsuccessful.

public class Email

{

private Order order;

public Email(Order order)

{

this.order = order;

}

public void send(){

String sendEmail = "\n";

if(order.isPaymentSuccessful()){

sendEmail +=

"Payment Successful \n" +

"Email: " + order.getCustomer().getEmailAddress() + "\n" +

"Customer: " + order.getCustomer().getFirstName() + " " +

order.getCustomer().getSurname() + "\n" +

"Order No.: #" + order.getId() + "\n" +

order + "\n" +

"Delivery Address:\n" + order.getShippingAddress() + "\n" +

"Billing Address:\n" + order.getPayment().getBillingAddress() + "\n";

}

else{

sendEmail +=

"Payment Unsuccessful \n" +

"Email: " + order.getCustomer().getEmailAddress() + "\n" +

"Dear Customer, " + order.getCustomer().getFirstName() + " " +

order.getCustomer().getSurname() + "\n" +

"Your order could not be placed." + "\n" +

"Please verify your payment method!" + "\n" +

"Order unsuccessful, please try again.\n";

}

System.out.println(sendEmail);

}

}