OCTOBER 16, 2018

CT2106 – Object Orientated Programming

Assignment 2: Shopping Cart transactions

KILLIAN O DÁLAIGH
18101573

Table of Contents

Scenario 1: Output

********** Transaction 1 **********

Hello Niamh
niamhol@zmail.com
Your order
Order Number:f3cead9b-d129-440d-b1a0-b28a6fea4825
Has been payed for and your order is as follows:

Item Id: 64567845678 Iphone Price: 1000
Item Id: 6989434567 Iphone XS Price: 1500
Item Id: 98765434567 Iphone XR Price: 1200

Your total is:3700.0
This order is being delivered to:
Reaskaun,Ennis,V56IY3R,Ireland

And billed to:
Reaskaun,Ennis,V56IY3R,Ireland

Regards,
Assignment2

********** End Transaction 1 **********

Scenario 2: Output

********** Transaction 2 **********

Your cart items are:
Item Id: 64567845678 Iphone Price: 1000
Item Id: 6989434567 Iphone XS Price: 1500
Item Id: 98765434567 Iphone XR Price: 1200
Sorry your card type is invalid

Hello Brian
b.sweet@ymail.com
Your order
Order Number:076c895f-4cce-436c-a388-7919725582e0
has not been payed, as there was issues with your card.
The order has been cancelled

Regards,
Assignment2
********** End Transaction 2 **********

Java Code

TransactionTest

```
/**
 * Description: TransactionTest is a class that does one thing only
 * It provides methods for testing out different test scenarios
 * for our Shopping Cart Transaction classes
 * @author Killian O'Dálaigh
 * @version 10 October 2018
 */

import java.util.Calendar;

public class TransactionTest
{

    /**
     * main method to execute the TransactionTest methods
     */
    public static void main(String[] args)
    {
       TransactionTest test = new TransactionTest();
       test.transaction1(); // calls the method with our test scenario
       test.transaction2(); // calls the method with our test scenario 2
    }

    /*
     * Description: First test scenario
     * Return: void
     * Parameters: none
     */
    public void transaction1()
    {

        System.out.println("\n********************************* Transaction 1
************************************\n");
        //1. Create New Customer
        Customer customer = new Customer("Niamh", "O'Leary",
"niamhol@zmail.com");

        //2. Create shopping Cart
        ShoppingCart shoppingCart = new ShoppingCart(customer);
        //3. Add 3 items
        Item item1 = new Item("Iphone", 64567845678L);
        Item item2 = new Item("Iphone XS", 6989434567L);
```

```
        Item item3 = new Item("Iphone XR", 98765434567L);

        item1.setPrice(1000);
        item2.setPrice(1500);
        item3.setPrice(1200);

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

        //4. Finalize the cart and create an order
        Order order = new Order(customer, shoppingCart);

        //5. Add a delivery address for the order
        Address address = new Address("Reaskaun", "Ennis", "V56IY3R",
"Ireland");
        order.setDeliveryAddress(address);

        //6. Add a payment
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH, 9);
        calendar.set(Calendar.YEAR, 2020);

        Payment payment = new Payment(customer, 2345678675436543L, calendar,
address, "AIB", "MasterCard");

        //7. Validate the payment
        //8. Send success email
        order.processOrder();
        order.confirmOrder(payment.isValidCard());

        System.out.println("\n******************************** End
Transaction 1 ***********************************\n");

    }// End Test Scenario 1

    /*
     * Description: Second test scenario
     * Return: void
     * Parameters: none
     */
    public void transaction2() {

        System.out.println("\n******************************** Transaction 2
***********************************\n");

        //1. Create customer
```

```
        Customer customer = new Customer("Brian", "Sweetman",
"b.sweet@ymail.com");
        //2. Create Cart
        ShoppingCart shoppingCart = new ShoppingCart(customer);
        //3. The user adds 3 items
        Item item1 = new Item("Iphone", 64567845678L);
        Item item2 = new Item("Iphone XS", 6989434567L);
        Item item3 = new Item("Iphone XR", 98765434567L);

        item1.setPrice(1000);
        item2.setPrice(1500);
        item3.setPrice(1200);

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

        //4. Display the Cart
        System.out.println("Your cart items are:" +
shoppingCart.listCartItems());

        //5. Removes an item
        shoppingCart.removeItem(item1);

        //6. Confirms the cart
        Order order = new Order(customer, shoppingCart);
        Address address = new Address("Reaskaun", "Ennis", "V56IY3R",
"Ireland");
        order.setDeliveryAddress(address);

        //7. Confirms the order
        //8. The user submits a payment
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH, 9);
        calendar.set(Calendar.YEAR, 2020);

        Payment payment = new Payment(customer, 2345678675436543L, calendar,
address, "AIB", "NasterCard");

        //9. Payment not valid
        //10. Email
        order.confirmOrder(payment.isValidCard());

        System.out.println("******************************** End Transaction
2 ************************************");

    }// End Test Scenario 2
```

```
}// End Class TransactionTes
```

ShoppingCart

```java
/**
 * Description: ShoppingCart provides a place to hold all
 * the items a customer might wish to buy in a placeholder
 * that can be edited before an order is placed.
 * @author Killian O'Dálaigh
 * @version 10 October 2018
 */

import java.util.ArrayList;
import java.util.Calendar;
import java.util.UUID;

public class ShoppingCart
{
    // instance variables - replace the example below with your own
    private Calendar timeStamp; // Holds creation date/time of the Cart
    private final String cartID; // Hold a unique CartID
    private ArrayList<Item> items; // Hold all users items
    private int total; // Hold total price of items in Cart
    private Customer customer; // Hold the users Customer class
    private boolean cartClosed; // Holds Status of the Cart
                            // True == Open
                            // False == Closed

    /**
     * Constructor for objects of class ShoppingCart
     */
    public ShoppingCart(Customer customer)
    {
        this.timeStamp = getTimeStamp();
        this.cartID = makeCartId();
        this.customer = customer;
        this.items = new ArrayList<>();
        this.total = 0;
        this.cartClosed = false;
    }// End Constructor

    /*
     * Methods
     */

    // Creates a time-stamp for the cart creation
    private Calendar getTimeStamp() {
        return Calendar.getInstance();
    }

    // Creates a unique Cart ID
```

```java
    private String makeCartId() {
        String uniqueID = UUID.randomUUID().toString();
        return uniqueID;
    }

    // Closes the cart for editing
    public void closeCart() {
        this.cartClosed = true;
    }

    // Adds an item to the cart
    public void addItem(Item item) {
        if (cartClosed == false) {
            this.items.add(item);
        }
        else {
            System.out.println("Sorry, this cart is closed");
        }
    }

    // Removes and item to the cart
    public void removeItem(Item item) {
        if (cartClosed == false) {
            this.items.remove(item);
        }
        else {
            System.out.println("Sorry, this cart is closed");
        }
    }

    /*
     * Getters and Setters
     */

    public ArrayList<Item> getItems() {
        return items;
    }

    public void setItems(ArrayList<Item> items) {
        this.items = items;
    }

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
```

```
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public boolean isCartClosed() {
        return cartClosed;
    }

    public void setCartClosed(boolean cartOpen) {
        this.cartClosed = cartOpen;
    }

    public String getCartID() {
        return cartID;
    }

    public void setTimeStamp(Calendar timeStamp) {
        this.timeStamp = timeStamp;
    }

    public String listCartItems() {
        String string1 = "";

        for (int i=0; i<this.items.size(); i++) {
            string1 += ("\n" + this.items.get(i));
        }

        return string1;
    }

}// End Class ShoppingCart
```

Order

```java
/**
 * Description: The order class is responsible for processing
 * an order and sending the user its information
 * @author Killian O'Dálaigh
 * @version 10 October 2018
 */

import java.util.UUID;
import java.util.ArrayList;

public class Order {

    private Customer customer;
    private ShoppingCart shoppingCart;
    private ArrayList<Item> orderItems;
    private String orderNumber;
    private Payment payment;
    private Email email;
    private double total;
    private Address deliveryAddress;
    private Address billingAddress;
    private boolean status; // True == Confirmed
                            // False == Not Confirmed

    /*
     * Class Constructor
     */
    public Order(Customer customer, ShoppingCart shoppingCart) {
        this.customer = customer;
        this.shoppingCart = shoppingCart;
        this.orderItems = new ArrayList<Item>(shoppingCart.getItems());
        this.orderNumber = makeOrderNumber();
        this.payment = null;
        this.email = null;
        this.status = false;
        this.total = getTotalItems();
        this.setDeliveryAddress(customer.getDeliveryAddress());
    }// End constructor

    /*
     * Methods
     */

    // Processes the Order
    public void processOrder() {

        // If there is no Delivery address prompt error message
```

```
        if (!(this.deliveryAddress.isEmpty())) {
            System.out.println("Error - You have no delivery address");
        }

        // If there is no Billing Address set it equal to the Delivery Address
        if (this.billingAddress == null) {
            this.billingAddress = this.deliveryAddress;
        }

        // Removes all the items in the shopping cart items array (Cleans the
Cart)
        this.shoppingCart.getItems().clear();

        // Close the cart for editing
        this.shoppingCart.closeCart();
    }

    // Confirms the Order and send out an email with the order details
    public void confirmOrder(boolean payment) {
        this.email = new Email(this.customer, this, payment);
        // Sends Email
        this.email.sendEmail(email.generateEmail(payment));
        this.status = true;
    }

    // Updates the users order
    public void update(Customer customer, ShoppingCart shoppingCart) {
        this.customer = customer;
        this.shoppingCart = shoppingCart;
        this.orderItems = shoppingCart.getItems();
        this.orderNumber = makeOrderNumber();
        this.payment = null;
        this.email = null;
        this.status = false;
    }

    // Makes a Unique Order number
    private String makeOrderNumber() {
        return (UUID.randomUUID().toString());
    }

    // Lists all the items ordered
    public String listOrderItems() {
        String string1 = "";

        for (int i=0; i<this.orderItems.size(); i++) {
            string1 += ("\n" + this.orderItems.get(i).toString());
        }
```

```
        return string1;
    }

    /*
     * Getters and Setters
     */

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public ArrayList<Item> getOrderItems() {
        return orderItems;
    }

    public void setOrderItems(ArrayList<Item> orderItems) {
        this.orderItems = orderItems;
    }

    public String getOrderNumber() {
        return orderNumber;
    }

    public void setOrderNumber(String orderNumber) {
        this.orderNumber = orderNumber;
    }

    public Payment getPayment() {
        return payment;
    }

    public void setPayment(Payment payment) {
        this.payment = payment;
    }

    public Email getEmail() {
        return email;
    }

    public void setEmail(Email email) {
        this.email = email;
    }
```

```java
    public Address getDeliveryAddress() {
        return deliveryAddress;
    }

    public void setDeliveryAddress(Address deliveryAddress) {
        this.deliveryAddress = deliveryAddress;
    }

    private double getTotalItems() {
        double total = 0;
        for(int i=0; i<this.orderItems.size(); i++) {
            total += this.orderItems.get(i).getPrice();
        }
        return total;
    }

    public ShoppingCart getShoppingCart() {
        return shoppingCart;
    }

    public void setShoppingCart(ShoppingCart shoppingCart) {
        this.shoppingCart = shoppingCart;
    }

    public double getTotal() {
        return total;
    }

    public void setTotal(double total) {
        this.total = total;
    }

    public boolean isStatus() {
        return status;
    }

    public void setStatus(boolean status) {
        this.status = status;
    }

    public Address getBillingAddress() {
        return billingAddress;
    }

    public void setBillingAddress(Address billingAddress) {
        this.billingAddress = billingAddress;
    }
```

```
}// End Class Order
```

Address

```java
/**
 * Description: The order class is responsible for processing
 * an order and sending the user its information
 * @author Killian O'Dálaigh
 * @version 10 October 2018
 */

import java.util.UUID;
import java.util.ArrayList;

public class Order {

    private Customer customer;
    private ShoppingCart shoppingCart;
    private ArrayList<Item> orderItems;
    private String orderNumber;
    private Payment payment;
    private Email email;
    private double total;
    private Address deliveryAddress;
    private Address billingAddress;
    private boolean status; // True == Confirmed
                            // False == Not Confirmed

    /*
     * Class Constructor
     */
    public Order(Customer customer, ShoppingCart shoppingCart) {
        this.customer = customer;
        this.shoppingCart = shoppingCart;
        this.orderItems = shoppingCart.getItems();
        this.orderNumber = makeOrderNumber();
        this.payment = null;
        this.email = null;
        this.status = false;
        this.total = getTotalItems();
        this.setDeliveryAddress(customer.getDeliveryAddress());
    }// End constructor

    /*
     * Methods
     */

    // Processes the Order
    public void processOrder() {

        // If there is no Delivery address prompt error message
```

```
        if (!(this.deliveryAddress.isEmpty())) {
            System.out.println("Error - You have no delivery address");
        }

        // If there is no Billing Address set it equal to the Delivery Address
        if (this.billingAddress == null) {
            this.billingAddress = this.deliveryAddress;
        }

        // Removes all the items in the shopping cart items array (Cleans the
Cart)
        for (int i=0; i<this.shoppingCart.getItems().size(); i++) {
            this.shoppingCart.getItems().remove(i);
        }

        // Close the cart for editing
        this.shoppingCart.closeCart();
    }

    // Confirms the Order and send out an email with the order details
    public void confirmOrder(boolean payment) {
        this.email = new Email(this.customer, this, payment);
        // Sends Email
        this.email.sendEmail(email.generateEmail(payment));
        this.status = true;
    }

    // Updates the users order
    public void update(Customer customer, ShoppingCart shoppingCart) {
        this.customer = customer;
        this.shoppingCart = shoppingCart;
        this.orderItems = shoppingCart.getItems();
        this.orderNumber = makeOrderNumber();
        this.payment = null;
        this.email = null;
        this.status = false;
    }

    // Makes a Unique Order number
    private String makeOrderNumber() {
        return (UUID.randomUUID().toString());
    }

    // Lists all the items ordered
    public String listOrderItems() {
        String string1 = "";

        for (int i=0; i<this.orderItems.size(); i++) {
```

```
            string1 += ("\n" + this.orderItems.get(i).toString());
        }

        return string1;
    }

    /*
     * Getters and Setters
     */

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public ArrayList<Item> getOrderItems() {
        return orderItems;
    }

    public void setOrderItems(ArrayList<Item> orderItems) {
        this.orderItems = orderItems;
    }

    public String getOrderNumber() {
        return orderNumber;
    }

    public void setOrderNumber(String orderNumber) {
        this.orderNumber = orderNumber;
    }

    public Payment getPayment() {
        return payment;
    }

    public void setPayment(Payment payment) {
        this.payment = payment;
    }

    public Email getEmail() {
        return email;
    }

    public void setEmail(Email email) {
        this.email = email;
```

```
    }

    public Address getDeliveryAddress() {
        return deliveryAddress;
    }

    public void setDeliveryAddress(Address deliveryAddress) {
        this.deliveryAddress = deliveryAddress;
    }

    private double getTotalItems() {
        double total = 0;
        for(int i=0; i<this.orderItems.size(); i++) {
            total += this.orderItems.get(i).getPrice();
        }
        return total;
    }

    public ShoppingCart getShoppingCart() {
        return shoppingCart;
    }

    public void setShoppingCart(ShoppingCart shoppingCart) {
        this.shoppingCart = shoppingCart;
    }

    public double getTotal() {
        return total;
    }

    public void setTotal(double total) {
        this.total = total;
    }

    public boolean isStatus() {
        return status;
    }

    public void setStatus(boolean status) {
        this.status = status;
    }

    public Address getBillingAddress() {
        return billingAddress;
    }

    public void setBillingAddress(Address billingAddress) {
        this.billingAddress = billingAddress;
```

```
    }

}// End Class Order
```

Payment

```
/**
 * Description: The Payment class provides validation on
 * card payment and bank details so that a customer can
 * buy the items they order
 * @author Killian O'Dálaigh
 * @version 10 October 2018
 */

import java.util.Calendar;

public class Payment {

    private Customer customer;
    private String cardType;
    private long cardNum;
    private Calendar cardDate;
    private Address address;
    private String bankName;
    private boolean validCard;

    /*
     * Class Constructor
     */
    public Payment(Customer customer, long cardNum, Calendar cardDate, Address
address, String bankName, String cardType) {

        this.customer = customer;
        setCardType(cardType);
        setCardNum(cardNum);
        setCardDate(cardDate);
        this.address = address;
        this.bankName = bankName;
        validCard = isValid(this);

    }// End constructor

    /*
     * Methods
     */

    /* Checks to see if the Payment details entered are valid card details
     * Returns: True is it is Valid
     *          False if it is not Valid
     */
    private boolean isValid(Payment payment) {

        boolean cardNum1 = false;
```

```
        boolean cardType1 = false;
        boolean cardDate1 = false;

        if ((this.cardType != null)&&(!this.cardType.isEmpty())) {
            cardType1 = true;
        }

        if ((this.cardNum != 0)) {
            cardNum1 = true;
        }

        if((this.cardDate.after(Calendar.getInstance())&&(this.cardDate !=
null))) {
            cardDate1 = true;
        }

        if (cardType1 == true && cardNum1 == true && cardDate1 == true)
        {
            return true;
        }
        else {
            return false;
        }
    }

    /* Checks the Date on the Card to make sure it is not expired
     * Returns: True if the date is Valid
     *          False if the date is not Valid
     */
    private boolean checkCardDate(Calendar cardDate) {
        if (cardDate.after(Calendar.getInstance())) {
            return true;
        }
        else {
            System.out.println("Sorry the date on your card is invalid");
            return false;
        }

    }

    /* Checks the Type on the Card to make sure it is either MasterCard or
Visa
     * Returns: True if the Type is Valid
     *          False if the Type is not Valid
     */
    private boolean checkCardType(String cardType) {
        if(cardType.equals("MasterCard") || cardType.equals("Visa")) {
            return true;
```

```
        }
        else {
            System.out.println("Sorry your card type is invalid");
            return false;
        }
    }

    /* Checks to see if the Card Number is a Valid 16 digit number
     * Returns: True if the Number is Valid
     *          False if the Number is not Valid
     */
    private boolean checkCardNum(long cardNum) {
        if ((cardNum > 1000000000000000L) && (cardNum < 9999999999999999L)) {
            return true;
        }
        else {
            System.out.println("Sorry your card number is invalid");
            return false;
        }
    }

    /*
     * Getters and Setters
     */

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public String getCardType() {
        return cardType;
    }

    public void setCardType(String cardType) {
        if(checkCardType(cardType)) {
            this.cardType = cardType;
        }
        else {
            this.cardType = null;
        }
    }

    public long getCardNum() {
        return cardNum;
```

```
    }

    public void setCardNum(long cardNum) {
        if(checkCardNum(cardNum)) {
            this.cardNum = cardNum;
        }
        else {
            this.cardNum = 0L;
        }
    }

    public Calendar getCardDate() {
        return cardDate;
    }

    public void setCardDate(Calendar cardDate) {
        if (checkCardDate(cardDate)) {
            this.cardDate = cardDate;
        }
        else {
            this.cardDate = null;
        }
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public String getBankName() {
        return bankName;
    }

    public void setBankName(String bankName) {
        this.bankName = bankName;
    }

    public boolean isValidCard() {
        return validCard;
    }

    public void setValidCard(boolean validCard) {
        this.validCard = validCard;
    }
}// End Class Payment
```

Email

```
/**
 * Description: The Email class is responsible for creating and sending
 * email to the client, whether the bought any items or not.
 * @author Killian O'Dálaigh
 * @version 10 October 2018
 */

public class Email {

    private Customer customer;
    private Order order;

    /*
     * Class Constructor
     */
    public Email(Customer customer, Order order, boolean payment) {
        this.customer = customer;
        this.order = order;
    }// End Constructor

    /*
     * Methods
     */

    // Creates a set EMail to be sent to the User
    public String generateEmail(boolean payment) {
        String greetings = ("\nHello " + customer.getFirstName() + "\n");
        String emailContent =  null;
        String emailAddress = (customer.getEmailAddress()+"\n");
        String signOff = ("\n\nRegards,\nAssignment2");

        if (payment == true) {
            emailContent = ("Your order\nOrder Number:" +
this.order.getOrderNumber() + "\nHas been payed for and your order is as
follows:\n" + this.order.listOrderItems() + "\n\nYour total is:" +
this.order.getTotal()
                            + "\nThis order is being delivered to:\n" +
this.order.getDeliveryAddress().catAddress() + "\n\nAnd billed to:\n" +
this.order.getBillingAddress().catAddress());
        }
        else {
            emailContent = ("Your order\nOrder Number:" +
this.order.getOrderNumber() + "\nhas not been payed,  as there was issues with
your card.\nThe order has been cancelled");
        }
        return (greetings+emailAddress+emailContent+signOff);
    }
```

```
    // Sends the Email that is passed into this function
    public void sendEmail(String fullEmail) {
        System.out.println(fullEmail);
    }

}// End Class Email
```