# CT2106 Assignment 2

Michael Mc Curtin

ID: 21459584

## Project Description

The project simulates an online shopping scenario (**TransactionTest)**.

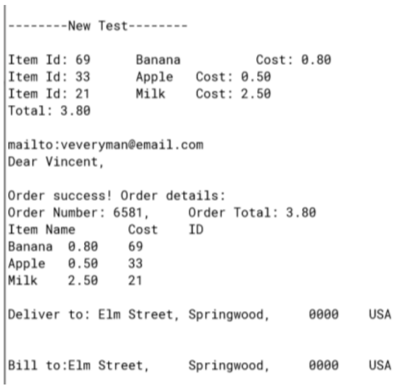
A **customer**, who has already created an account, selects some **items** to add to their **shopping cart**.

The details of their **order** are then finalised and the total is calculated. The customer adds **payment** information, which is validated. They also provide billing and payment **addresses**.

Finally, an **email** is generated to be sent to the customer.

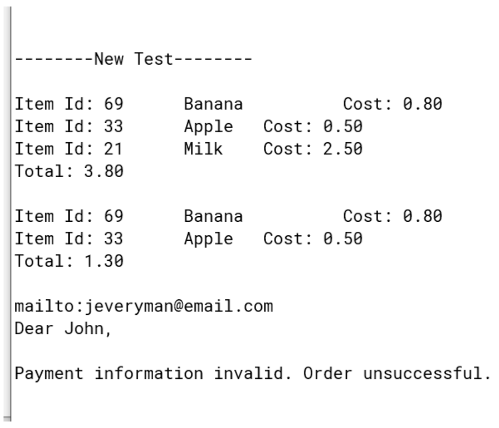
The functionality is tested with two scenarios.

## Scenario 1 Output



```
--------New Test--------

Item Id: 69      Banana          Cost: 0.80
Item Id: 33      Apple   Cost: 0.50
Item Id: 21      Milk    Cost: 2.50
Total: 3.80

mailto:veveryman@email.com
Dear Vincent,

Order success! Order details:
Order Number: 6581,     Order Total: 3.80
Item Name       Cost    ID
Banana  0.80    69
Apple   0.50    33
Milk    2.50    21

Deliver to: Elm Street, Springwood,     0000    USA


Bill to:Elm Street,     Springwood,     0000    USA
```

## Scenario 2 Output



```
--------New Test--------

Item Id: 69      Banana          Cost: 0.80
Item Id: 33      Apple   Cost: 0.50
Item Id: 21      Milk    Cost: 2.50
Total: 3.80

Item Id: 69      Banana          Cost: 0.80
Item Id: 33      Apple   Cost: 0.50
Total: 1.30

mailto:jeveryman@email.com
Dear John,

Payment information invalid. Order unsuccessful.
```

## TransactionTest

```
/*
 * TransactionTest class runs two test shopping scenarios
 */
import java.text.SimpleDateFormat;

public class TransactionTest
{
   /*
    * Constructor for objects of class Customer
    */
   public TransactionTest()
   {
   }

   public static void main(String[] args) throws java.text.ParseException {

      // run both test scenarios

      test1();
      test2();
   }
```

```
        /*
         * test1
         * Create Customer object
         * Create Shopping Cart object for the Customer
         * Add 3 items with known cost to cart
         * Finalise the cart and create an order
         * Add a delivery address for the order
         * Add a payment type
         * Validate the payment
         * If successful, email the customer with a success email and the cost of the purchased items
        */
    public static void test1() throws java.text.ParseException {




        System.out.println("\n--------New Test--------\n");

        Customer customer = new Customer("Vincent", "Everyman", "veveryman@email.com");
        ShoppingCart cart = new ShoppingCart(customer.makeCustomerId()); // make cart with
randomly generated cart ID

        customer.assignCart(cart);

        customer.addItem("Banana", 0.8, 69);
        customer.addItem("Apple", 0.5, 33);
        customer.addItem("Milk", 2.5, 21);

        customer.displayCart();

        Order order = new Order(cart);
        order.makeOrderNo(); // randomly generate order number


        order.transferItems(); // transfer all items from cart to order then clear the cart


        order.makeOrderDetails(); // generate order details for email


        Address delivery = new Address();
        delivery.setAddress("Elm Street","Springwood", "0000", "USA");


        order.addAddress(delivery);


        Address billing = new Address();
        // Vincent's billing address happens to be the same as his delivery address
        billing.setAddress("Elm Street","Springwood", "0000", "USA");



        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd"); // sdf parse function is utilised
to pass through card date as parameter


        Payment payment = new Payment(customer, "Visa",1919191919191919L, sdf.parse("2022-11-
22"), billing, "Bank & Co");


        Email email = new Email(customer, order, billing, payment);


        email.sendEmail(); // generates email (text depends on payment validity), then prints it
    }
    public static void test2() throws java.text.ParseException {
        /*
         * The second scenario is a slight variation of the first:
         * The user adds three items
         * Requests a display of the shopping cart items and total
         * Removes one item
         * Confirms the cart and makes an order
         * The user submits a payment, however, the payment is not valid
         * The user is sent a regret email notifying them that the order was unsuccessful
         *
         */

        System.out.println("\n--------New Test--------\n");



        Customer customer = new Customer("John", "Everyman", "jeveryman@email.com");
        ShoppingCart cart = new ShoppingCart(customer.makeCustomerId()); // make cart with
randomly generated cart ID

        customer.assignCart(cart);

        customer.addItem("Banana", 0.8, 69);
        customer.addItem("Apple", 0.5, 33);
        customer.addItem("Milk", 2.5, 21);

        customer.displayCart();

        customer.removeItem(2); // remove "Milk" item at index 2

        customer.displayCart();
        Order order = new Order(cart);
        order.makeOrderNo(); // randomly generate order number
        order.transferItems(); // transfer all items from cart to order then clear the cart

        order.makeOrderDetails(); // generate order details for email

        Address delivery = new Address();
        delivery.setAddress("Elm Street","Springwood", "0000", "USA");

        order.addAddress(delivery);



        Address billing = new Address();
        // John's billing address happens to be the same as his delivery address
        billing.setAddress("Elm Street","Springwood", "0000", "USA");

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd"); // sdf parse function is utilised
to pass through card date as parameter

        Payment payment = new Payment(customer, "WasterCard",1818181818181818L,
sdf.parse("2023-12-20"), billing, "Bank & Co");

        Email email = new Email(customer, order, billing, payment);

        email.sendEmail(); // generates email (text depends on payment validity), then prints it
    }
}
```

## ShoppingCart

```
/*
 * ShoppingCart class
 * items can be added and removed while the cart is open
 * cart contents and total cost can be displayed to the user
 */

import java.util.ArrayList;
import java.time.LocalDateTime;

public class ShoppingCart
{
   // instance variables

   private long cartId;
   private Item item;
   private LocalDateTime time;
   private double totalCost = 0;
   private boolean closed = false; // cart is open when created

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

```
/**
 * Constructor for objects of class ShoppingCart
 */
public ShoppingCart(long cartId)
{
   this.cartId = cartId;
}

/*
 * addItem method
 * adds the specified item to the cart but only if the cart is open
 */

public void addItem(String name, double cost, long itemId) {
   if (this.closed == false) {
      Item newItem = new Item(name, cost, itemId);
      items.add(newItem);
   }
   else {
      System.out.println("Sorry! Cart is closed!");
   }

}

/*
 * removeItem method
 * removes the specified item from the cart
 */

public void removeItem(int index) {
    items.remove(index);
}

/*
 * displayCart method
 * prints out each item in the cart along with the total cost
 */

public void displayCart() {
            for (int i = 0; i < items.size(); i++) {
                System.out.println(items.get(i).toString());
            }

            System.out.println(String.format("Total: %.2f\n", calculateTotal()));
}

/*
 * calculateTotal method
 * adds the cost of each item together to find the total cost of the order
 */


public double calculateTotal() {
        totalCost = 0; // reset count

        for (int i = 0; i < items.size(); i++) {
            this.totalCost += (items.get(i).getCost());
        }

        return totalCost;
}
```

```
  // closeCart method sets the cart to closed

  public void closeCart() {

    this.closed = true;

  }


  // getItem method returns the specified item


  public Item getItem(int index) {

    return(items.get(index));

  }


  // countItems method returns the amount of items

    public int countItems() {

    return(items.size());

  }


}
```

## Order

```
/*
 * Order class creates an order by transferring the items from the shopping cart
 * also generates the order details to be sent to the customer
 */

import java.util.ArrayList;
import java.util.Random;

public class Order
{
   // instance variables

   private double orderTotal;
   private ShoppingCart cart;
   private long orderNo;
   private ArrayList<Item> items = new ArrayList<Item>();

   private String details;
   private Address delivery;

   /**
    * Constructor for objects of class Order
    */
   public Order(ShoppingCart cart)
   {
      this.cart = cart;
      this.orderTotal = cart.calculateTotal();
      this.orderNo = makeOrderNo();
   }
```

```
    /*
     * makeOrderNo
     * randomly generates an order number between 1-10000
     */
    public long makeOrderNo() {
        Random random = new Random();
        return(random.nextInt(10000));
    }

    /*
     * makeOrderDetails method
     * generates the order details, listing the details of each item in the order
    */

    public String makeOrderDetails() {
        details = String.format("Order Number: %d,\tOrder Total: %.2f \nItem Name\tCost\tID\n",
orderNo, orderTotal);

        for (int i = 0; i < items.size(); i++) {
            details += String.format("%s\t%.2f\t%d\t\n", items.get(i).getName(), items.get(i).getCost(),
items.get(i).getId());

        }
        return(details);
    }
    /*
     * addAddress method
     * adds the delivery address to the order details
     */

    public void addAddress(Address delivery) {
        this.delivery = delivery;
        details += String.format("\nDeliver to: %s\n", delivery.getAddress());
    }

    /*
     * transferItems method
     * transfers each item from the cart into the order
     * then removes the items from the cart and closes it
     */
    public void transferItems() {
        for (int i = 0; i < cart.countItems(); i++) {
            items.add(cart.getItem(i));
        }
        for (int i = 0; i < cart.countItems(); i++) {
            cart.removeItem(i);
        }
        cart.closeCart();
    }
    // getter method returns the generated order details

        public String getOrderDetails() {
        return(details);
    }
}
```

## Address

```
/*
 * Address class holds information about the customer's address
 * contains methods to set and get the address
 */

public class Address
{
   // instance variables

   private String street;
   private String city;
   private String zip;
   private String country;



   /**
    * Constructor for objects of class Address
    */
   public Address()
   {
   }

   // getter and setter methods

   public void setAddress(String street, String city, String zip, String country) {
      this.street = street;
      this.city = city;
      this.zip = zip;
      this.country = country;
   }
```

```
    public String getAddress() {
        return String.format("%s,\t%s,\t%s\t%s\n", street, city, zip, country);
    }

}
```

## Payment

```
/*
 * Payment class holds information about the customer's payment
 * contains the isValid method to validate the payment
 */

import java.util.Date;


public class Payment

{
   // instance variables

   private Customer customer;
   private String type;
   private String cardType;
   private long CCNum;
   private Date date;
   private Address address;
   private String bankName;
```

```
    /**
     * Constructor for objects of class Payment
     */
    public Payment(Customer customer, String cardType, long CCNum, Date date, Address address,
String bankName)
    {
        this.customer = customer;
        this.cardType = cardType;
        this.CCNum = CCNum;
        this.date = date;
        this.address = address;
        this.bankName = bankName;
    }

    /*
     * isValid method
     * checks whether the provided card type matches the valid types MasterCard or Visa
     */
    public boolean isValid() {
        if (this.cardType == "MasterCard" || this.cardType == "Visa") {
            return true;
        }
        else {
            return false;
        }
    }
}
```

## Email

```
/*
 * Email class generates the email to be sent to the customer
 * content of email depends on whether the payment is valid or not
 * i.e. the order details will only be sent if the payment is valid
*/

public class Email

{

   // instance variables

   private String firstName;
   private String lastName;
   private String emailAddress;
   private String introduction;

   private Order order;
   private Payment payment;
   private Address billing;

   private long orderNo;


   private String successfulMessage;
   private String failureMessage;
```

```
    // Email constructor constructs the body of the text to be sent to the customer

    public Email(Customer customer, Order order, Address billing, Payment payment)
    {
        this.firstName = customer.getfirstName();
        this.lastName = customer.getlastName();
        this.emailAddress = customer.getemailAddress();
        this.payment = payment;

        this.introduction = String.format("mailto:%s \nDear %s,\n", emailAddress, firstName);

        this.successfulMessage = String.format("Order success! Order details:\n%s\nBill
to:%s\n",order.getOrderDetails(), billing.getAddress());

        this.failureMessage = String.format("Payment information invalid. Order unsuccessful.\n");


    }
    /*
     * sendEmail method
     * greets the customer
     * sends email regarding a successful or unsuccessful order depending on payment validity
     */
    public void sendEmail()
    {
        System.out.println(introduction);

        if (payment.isValid()) {
            System.out.println(successfulMessage);
        }

        else {
            System.out.println(failureMessage);

        }
    }
}
```