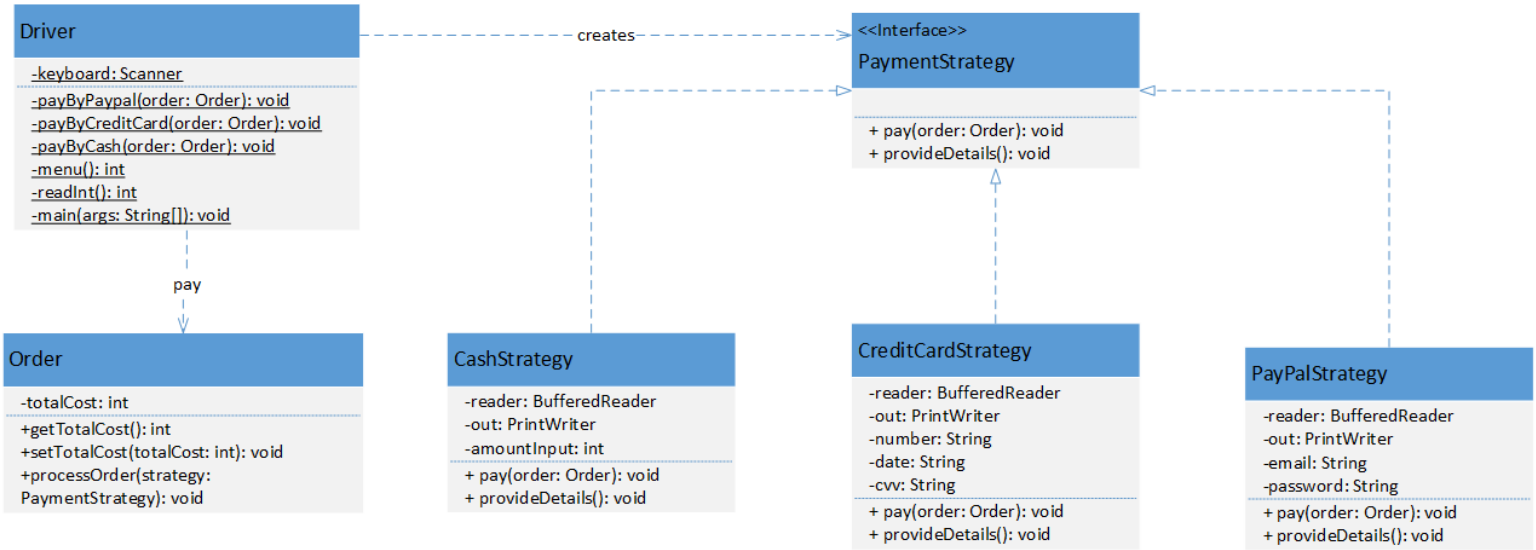
Queens College Airline Booking System (QCABS)

Group 3
Thayany Jeyakumaran

Chosen Design Pattern: Strategy Design Pattern

Description of Contribution to Project: Programmed the payment portion of the flight booking system using the strategy design pattern. Set up multiple strategies in separate classes representing different forms of payment.

UML Diagram:



Program:

Driver

Driver.java

```java
import java.io.BufferedReader;

/**
 * Airline Booking
 *
 * Payment System
 *
 */
public class Driver {

    /**
     * main method to start Java application
     * @param args program arguments
     */
    public static void main(String[] args) {

        //order
        Order order = new Order(100);

        System.out.println("Total price is $" + order.getTotalCost());

        // display menu and read selection
        int selection = menu();

        // run until user quits
        while (selection != 0 && order.getTotalCost() > 0) {

            // process user command
            switch (selection) {

                case 1: // Pay by cash
                    payByCash(order);
                    break;
                case 2:// Pay by credit card
                    payByCreditCard(order);
                    break;
                case 3:// Pay by Paypal
                    payByPaypal(order);
                    break;

                default:
                    System.out.println("Invalid selection");
                    break;
            }

            if (order.getTotalCost() > 0) {

                System.out.println();

                System.out.println("Total price is $" + order.getTotalCost());

                // display menu and read selection
                selection = menu();
            }
        }

        System.out.println("\nThank you for using the payment system");
    }
```

```java
    /**
     * pay by paypal
     * @param order order
     * @return true/false
     */
    private static void payByPaypal(Order order) {

        System.out.print("Enter the email: ");
        String email = keyboard.nextLine();

        System.out.print("Enter the password: ");
        String password = keyboard.nextLine();

        String input = email + "\n" + password + "\n";

        //create the input
        Reader inputString = new StringReader(input);
        BufferedReader reader = new BufferedReader(inputString);

        //create the output
        StringWriter out    = new StringWriter();
        PrintWriter  writer = new PrintWriter(out);

        PaymentStrategy strategy = new PayPalStrategy(reader, writer);

        //provide the details
        strategy.provideDetails();

        //pay
        strategy.pay(order);

        System.out.println(out.toString());

    }
```

```java
    /**
     * pay by credit card
     * @param order order
     * @return true/false
     */
    private static void payByCreditCard(Order order) {

        System.out.print("Enter the card number: ");
        String number = keyboard.nextLine();

        System.out.print("Enter the expiration date: ");
        String date = keyboard.nextLine();

        System.out.print("Enter the cvv number: ");
        String cvv = keyboard.nextLine();

        String input = number + "\n" + date + "\n" + cvv + "\n";

        //create the input
        Reader inputString = new StringReader(input);
        BufferedReader reader = new BufferedReader(inputString);

        //create the output
        StringWriter out    = new StringWriter();
        PrintWriter  writer = new PrintWriter(out);

        PaymentStrategy strategy = new CreditCardStrategy(reader, writer);

        //provide the details
        strategy.provideDetails();

        //pay
        strategy.pay(order);

        System.out.println(out.toString());

    }
```

```java
    /**
     * pay by cash
     * @param order order
     * @return true/false
     */
    private static void payByCash(Order order) {

        System.out.print("Enter the amount of cash: ");
        int cash = readInt();

        //create the input
        Reader inputString = new StringReader(String.valueOf(cash));
        BufferedReader reader = new BufferedReader(inputString);

        //create the output
        StringWriter out    = new StringWriter();
        PrintWriter  writer = new PrintWriter(out);

        PaymentStrategy strategy = new CashStrategy(reader, writer);

        //provide the details
        strategy.provideDetails();

        //pay
        strategy.pay(order);

        System.out.println(out.toString());

    }
```

```java
    /**
     * display menu and return selection
     *
     * @return selection
     */
    private static int menu() {

        System.out.println("1. Pay by cash");
        System.out.println("2. Pay by credit card");
        System.out.println("3. Pay by Paypal");
        System.out.println("0. Quit");

        return readInt();
    }

    /**
    * create the Scanner to read from standard input
    */
    private static Scanner keyboard = new Scanner(System.in);

    /**
     * read an integer from console
     *
     * @return integer an integer
     */
    public static int readInt() {

        int integerNumber;

        // loop until user enters an integer
        while (true) {

            try {

                integerNumber = Integer.parseInt(keyboard.nextLine());
                break; // valid integer number

            } catch (NumberFormatException e) {
                System.out.print("Invalid input. Try again: ");
            } // end try
        } // end while

        return integerNumber;
    }
}
```

Order:

Order.java

```java
/**
 * Order class represents an order
 */
public class Order {

    /**
     * total cost
     */
    private int totalCost;

    /**
     * constructor
     * @param totalCost
     */
    public Order(int totalCost) {
        this.totalCost = totalCost;
    }

    /**
     * get total cost
     * @return total cost
     */
    public int getTotalCost() {
        return totalCost;
    }

    /**
     * set total cost
     * @param totalCost total cost
     */
    public void setTotalCost(int totalCost) {
        this.totalCost = totalCost;
    }

    /**
     * pay this order
     * @param strategy specific strategy
     */
    public void processOrder(PaymentStrategy strategy) {
        strategy.provideDetails();
        strategy.pay(this);
    }
}
```

PaymentStrategy:

PaymentStrategy.java

```java
/**
 * PaymentStrategy class represents the payment strategy
 * that defines the prototypes that the specific payment
 * should follow
 *
 */
public interface PaymentStrategy {

    /**
     * pay the amount
     * @param order order to pay
     */
     public void pay(Order order);

     /**
      * provide the details for the payment processing
      */
     public void provideDetails();
}

```

CashStrategy:

CashStrategy.java

```java
import java.io.BufferedReader;

/**
 * Pay by cash strategy
 */
public class CashStrategy implements PaymentStrategy {

    /**
     * buffer reader
     */
    private BufferedReader reader;

    /**
     * output
     */
    private PrintWriter out;

    /**
     * input amount
     */
    private int amountInput;

    /**
     * constructor
     *
     * @param reader reader
     * @param out output
     */
    public CashStrategy(BufferedReader reader, PrintWriter out) {
        this.reader = reader;
        this.out = out;
    }

    @Override
    public void pay(Order order) {

        if (amountInput >= order.getTotalCost()) {

            out.print("Payment Successful");

            //pay all cost
            order.setTotalCost(0);

        }else {

            //pay some cost
            order.setTotalCost(order.getTotalCost() - amountInput);

            out.print("Pay only $" + amountInput);
        }
    }

    @Override
    public void provideDetails() {

        try {
            amountInput = Integer.parseInt(reader.readLine());
        }catch(Exception e) {
            amountInput = 0;
        }
    }

}
```

CreditCardStrategy:

CreditCardStrategy.java

```java
import java.io.BufferedReader;

/**
 * Pay by CreditCard strategy
 */
public class CreditCardStrategy implements PaymentStrategy {

    /**
     * buffer reader
     */
    private BufferedReader reader;

    /**
     * output
     */
    private PrintWriter out;

    /**
     * constructor
     *
     * @param reader reader
     * @param out output
     */
    public CreditCardStrategy(BufferedReader reader, PrintWriter out) {
        this.reader = reader;
        this.out = out;
    }

    /**
     * credit card number
     */
    private String number;

    /**
     * date
     */
    private String date;

    /**
     * cvv
     */
    private String cvv;


    @Override
    public void pay(Order order) {

        if (number != null && !number.equals("") &&
                date != null && !date.equals("") &&
                cvv != null && !cvv.equals("")) {
            out.print("Payment Successful");

            order.setTotalCost(0);

        }else {
            out.print("Invalid credit card! Payment Failed");
        }
    }

    @Override
    public void provideDetails() {

        try {
            number = reader.readLine();
            date = reader.readLine();
            cvv = reader.readLine();
        } catch (IOException e) {
            number = "";
            date = "";
            cvv = "";
        }
    }

}
```

PaypalStrategy:

PayPalStrategy.java

```java
import java.io.BufferedReader;

/**
 * Pay by PayPal strategy
 */
public class PayPalStrategy implements PaymentStrategy {

    /**
     * buffer reader
     */
    private BufferedReader reader;

    /**
     * output
     */
    private PrintWriter out;

    /**
     * email
     */
    private String email = "";

    /**
     * password
     */
    private String password = "";

    /**
     * constructor
     *
     * @param reader reader
     * @param out output
     */
    public PayPalStrategy(BufferedReader reader, PrintWriter out) {
        this.reader = reader;
        this.out = out;
    }


    @Override
    public void pay(Order order) {

        if (email != null && !email.equals("") && password != null &&
                !password.equals("")) {
            out.print("Payment Successful");

            order.setTotalCost(0);
        }else {
            out.print("Wrong email or password! Payment Failed");
        }
    }

    @Override
    public void provideDetails() {

        try {
            email = reader.readLine();
            password = reader.readLine();
        } catch (IOException e) {
            email = "";
            password = "";
        }
    }

}
```

Unit Tests:

1. First unit test checks to ensure payment goes through when enough cash is available for payment

```
@Test
/**
 * test pay by cash, enough cash to pay
 */
void testEnoughCash() {

    //create the input
    Reader inputString = new StringReader("10");
    BufferedReader reader = new BufferedReader(inputString);

    //create the output
    StringWriter out    = new StringWriter();
    PrintWriter  writer = new PrintWriter(out);

    PaymentStrategy strategy = new CashStrategy(reader, writer);

    //provide the details
    strategy.provideDetails();

    Order order = new Order(10);

    //pay
    strategy.pay(order);
    assertEquals(0, order.getTotalCost());

    assertEquals("Payment Successful", out.toString());
}
```

2. Second unit test checks to ensure another form of payment is requested when enough cash isn't available for payment

```
@Test
/**
 * Test pay by cash, not enough cash
 */
void testNotEnoughCash() {

    //create the input
    Reader inputString = new StringReader("10");
    BufferedReader reader = new BufferedReader(inputString);

    //create the output
    StringWriter out    = new StringWriter();
    PrintWriter  writer = new PrintWriter(out);

    PaymentStrategy strategy = new CashStrategy(reader, writer);

    //provide the details
    strategy.provideDetails();

    Order order = new Order(20);

    //pay
    strategy.pay(order);
    assertEquals(10, order.getTotalCost());

    assertEquals("Pay only $10", out.toString());
}
```

Component Test:

In the case that only part of a payment was made through one payment strategy, can the program go back and request the remaining payment through another payment strategy and successfully execute.

```java
//Component Testing
@Test
/**
 * Test pay by cash, not enough cash
 */
void testPayManyTimes() {

    //create the input
    Reader inputString = new StringReader("10");
    BufferedReader reader = new BufferedReader(inputString);

    //create the output
    StringWriter out    = new StringWriter();
    PrintWriter  writer = new PrintWriter(out);

    PaymentStrategy strategy = new CashStrategy(reader, writer);

    //provide the details
    strategy.provideDetails();

    Order order = new Order(20);

    //pay
    strategy.pay(order);
    assertEquals(10, order.getTotalCost());

    assertEquals("Pay only $10", out.toString());


    //pay again by credit card
    inputString = new StringReader("1234567890\n11/02/2024\n4356\n");
    reader = new BufferedReader(inputString);

    //create the output
    out    = new StringWriter();
    writer = new PrintWriter(out);

    strategy = new CreditCardStrategy(reader, writer);

    //provide the details
    strategy.provideDetails();

    strategy.pay(order);
    assertEquals(0, order.getTotalCost());

    assertEquals("Payment Successful", out.toString());

    System.out.println("The order ($20) was paid $10 by cash and");
    System.out.println("$10 by credit card (number: 1234567890, date: 11/02/2024, cvv: 4356)");

}
```