

Queens College Airline Booking System (QCABS)

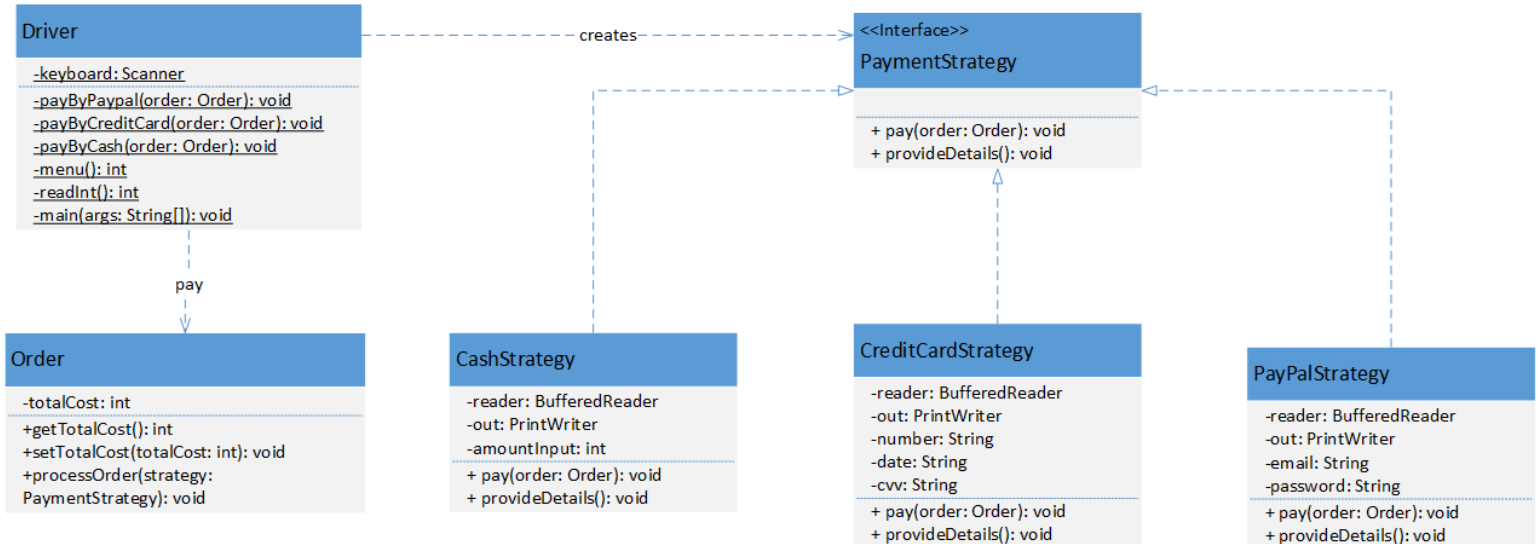
Group 3

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 X
1 import java.io.BufferedReader;
2
3 /**
4  * Airline Booking
5  * Payment System
6  */
7 public class Driver {
8
9     /**
10      * main method to start Java application
11      * @param args program arguments
12      */
13     public static void main(String[] args) {
14
15         //order
16         Order order = new Order(100);
17
18         System.out.println("Total price is $" + order.getTotalCost());
19
20         // display menu and read selection
21         int selection = menu();
22
23         // run until user quits
24         while (selection != 0 && order.getTotalCost() > 0) {
25
26             // process user command
27             switch (selection) {
28
29                 case 1: // Pay by cash
30                     payByCash(order);
31                     break;
32                 case 2: // Pay by credit card
33                     payByCreditCard(order);
34                     break;
35                 case 3: // Pay by Paypal
36                     payByPaypal(order);
37                     break;
38
39                 default:
40                     System.out.println("Invalid selection");
41                     break;
42             }
43
44             if (order.getTotalCost() > 0) {
45
46                 System.out.println();
47
48                 System.out.println("Total price is $" + order.getTotalCost());
49
50                 // display menu and read selection
51                 selection = menu();
52             }
53         }
54
55         System.out.println("\nThank you for using the payment system");
56     }
57 }
```

```

65- /**
66  * pay by paypal
67  * @param order order
68  * @return true/false
69  */
70- private static void payByPaypal(Order order) {
71
72     System.out.print("Enter the email: ");
73     String email = keyboard.nextLine();
74
75     System.out.print("Enter the password: ");
76     String password = keyboard.nextLine();
77
78     String input = email + "\n" + password + "\n";
79
80     //create the input
81     Reader inputString = new StringReader(input);
82     BufferedReader reader = new BufferedReader(inputString);
83
84     //create the output
85     StringWriter out = new StringWriter();
86     PrintWriter writer = new PrintWriter(out);
87
88     PaymentStrategy strategy = new PayPalStrategy(reader, writer);
89
90     //provide the details
91     strategy.provideDetails();
92
93     //pay
94     strategy.pay(order);
95
96     System.out.println(out.toString());
97 }
98
99
100- /**
101  * pay by credit card
102  * @param order order
103  * @return true/false
104  */
105- private static void payByCreditCard(Order order) {
106
107     System.out.print("Enter the card number: ");
108     String number = keyboard.nextLine();
109
110     System.out.print("Enter the expiration date: ");
111     String date = keyboard.nextLine();
112
113     System.out.print("Enter the cvv number: ");
114     String cvv = keyboard.nextLine();
115
116     String input = number + "\n" + date + "\n" + cvv + "\n";
117
118     //create the input
119     Reader inputString = new StringReader(input);
120     BufferedReader reader = new BufferedReader(inputString);
121
122     //create the output
123     StringWriter out = new StringWriter();
124     PrintWriter writer = new PrintWriter(out);
125
126     PaymentStrategy strategy = new CreditCardStrategy(reader, writer);
127
128     //provide the details
129     strategy.provideDetails();
130
131     //pay
132     strategy.pay(order);
133
134     System.out.println(out.toString());
135 }
136
137

```

```

137
138- /**
139  * pay by cash
140  * @param order order
141  * @return true/false
142  */
143- private static void payByCash(Order order) {
144
145     System.out.print("Enter the amount of cash: ");
146     int cash = readInt();
147
148     //create the input
149     Reader inputString = new StringReader(String.valueOf(cash));
150     BufferedReader reader = new BufferedReader(inputString);
151
152     //create the output
153     StringWriter out = new StringWriter();
154     PrintWriter writer = new PrintWriter(out);
155
156     PaymentStrategy strategy = new CashStrategy(reader, writer);
157
158     //provide the details
159     strategy.provideDetails();
160
161     //pay
162     strategy.pay(order);
163
164     System.out.println(out.toString());
165
166 }
167

```

```

168- /**
169  * display menu and return selection
170  *
171  * @return selection
172  */
173- private static int menu() {
174
175     System.out.println("1. Pay by cash");
176     System.out.println("2. Pay by credit card");
177     System.out.println("3. Pay by Paypal");
178     System.out.println("0. Quit");
179
180     return readInt();
181 }
182
183- /**
184  * create the Scanner to read from standard input
185  */
186 private static Scanner keyboard = new Scanner(System.in);
187
188- /**
189  * read an integer from console
190  *
191  * @return integer an integer
192  */
193- public static int readInt() {
194
195     int integerNumber;
196
197     // loop until user enters an integer
198     while (true) {
199
200         try {
201
202             integerNumber = Integer.parseInt(keyboard.nextLine());
203             break; // valid integer number
204
205         } catch (NumberFormatException e) {
206             System.out.print("Invalid input. Try again: ");
207         } // end try
208     } // end while
209
210     return integerNumber;
211 }
212 }
213

```

Order:

Order.java X

```
1  /**
2   * Order class represents an order
3   */
4   public class Order {
5
6   /**
7    * total cost
8    */
9    private int totalCost;
10
11 /**
12  * constructor
13  * @param totalCost
14  */
15 public Order(int totalCost) {
16     this.totalCost = totalCost;
17 }
18
19 /**
20  * get total cost
21  * @return total cost
22  */
23 public int getTotalCost() {
24     return totalCost;
25 }
26
27 /**
28  * set total cost
29  * @param totalCost total cost
30  */
31 public void setTotalCost(int totalCost) {
32     this.totalCost = totalCost;
33 }
34
35 /**
36  * pay this order
37  * @param strategy specific strategy
38  */
39 public void processOrder(PaymentStrategy strategy) {
40     strategy.provideDetails();
41     strategy.pay(this);
42 }
43 }
44
```

PaymentStrategy:

```
PaymentStrategy.java X
1  /**
2   * PaymentStrategy class represents the payment strategy
3   * that defines the prototypes that the specific payment
4   * should follow
5   *
6   */
7  public interface PaymentStrategy {
8
9      /**
10     * pay the amount
11     * @param order order to pay
12     */
13     public void pay(Order order);
14
15     /**
16     * provide the details for the payment processing
17     */
18     public void provideDetails();
19 }
20
```

CashStrategy:

```
CashStrategy.java X
1+ import java.io.BufferedReader;
3
4- /**
5  * Pay by cash strategy
6  */
7  public class CashStrategy implements PaymentStrategy {
8
9-     /**
10     * buffer reader
11     */
12     private BufferedReader reader;
13
14-     /**
15     * output
16     */
17     private PrintWriter out;
18
19-     /**
20     * input amount
21     */
22     private int amountInput;
23
24-     /**
25     * constructor
26     *
27     * @param reader reader
28     * @param out output
29     */
30-     public CashStrategy(BufferedReader reader, PrintWriter out) {
31         this.reader = reader;
32         this.out = out;
33     }
34
35-     @Override
36     public void pay(Order order) {
37
38         if (amountInput >= order.getTotalCost()) {
39
40             out.print("Payment Successful");
41
42             //pay all cost
43             order.setTotalCost(0);
44
45         } else {
46
47             //pay some cost
48             order.setTotalCost(order.getTotalCost() - amountInput);
49
50             out.print("Pay only $" + amountInput);
51         }
52     }
53
54-     @Override
55     public void provideDetails() {
56
57         try {
58             amountInput = Integer.parseInt(reader.readLine());
59         } catch (Exception e) {
60             amountInput = 0;
61         }
62     }
63
64 }
```

CreditCardStrategy:

```
CreditCardStrategy.java X
1 import java.io.BufferedReader;
2
3
4
5 /**
6  * Pay by CreditCard strategy
7  */
8 public class CreditCardStrategy implements PaymentStrategy {
9
10    /**
11     * buffer reader
12     */
13    private BufferedReader reader;
14
15    /**
16     * output
17     */
18    private PrintWriter out;
19
20    /**
21     * constructor
22     *
23     * @param reader reader
24     * @param out output
25     */
26    public CreditCardStrategy(BufferedReader reader, PrintWriter out) {
27        this.reader = reader;
28        this.out = out;
29    }
30
31    /**
32     * credit card number
33     */
34    private String number;
35
36    /**
37     * date
38     */
39    private String date;
40
41    /**
42     * cvv
43     */
44    private String cvv;
45
46
47    @Override
48    public void pay(Order order) {
49
50        if (number != null && !number.equals("") &&
51            date != null && !date.equals("") &&
52            cvv != null && !cvv.equals("")) {
53            out.print("Payment Successful");
54
55            order.setTotalCost(0);
56
57        } else {
58            out.print("Invalid credit card! Payment Failed");
59        }
60    }
61
62    @Override
63    public void provideDetails() {
64
65        try {
66            number = reader.readLine();
67            date = reader.readLine();
68            cvv = reader.readLine();
69        } catch (IOException e) {
70            number = "";
71            date = "";
72            cvv = "";
73        }
74    }
75
76 }
```


PaypalStrategy:

```
PayPalStrategy.java X
1 import java.io.BufferedReader;
4
5 /**
6  * Pay by PayPal strategy
7  */
8 public class PayPalStrategy implements PaymentStrategy {
9
10 /**
11  * buffer reader
12  */
13 private BufferedReader reader;
14
15 /**
16  * output
17  */
18 private PrintWriter out;
19
20 /**
21  * email
22  */
23 private String email = "";
24
25 /**
26  * password
27  */
28 private String password = "";
29
30 /**
31  * constructor
32  *
33  * @param reader reader
34  * @param out output
35  */
36 public PayPalStrategy(BufferedReader reader, PrintWriter out) {
37     this.reader = reader;
38     this.out = out;
39 }
40
41
42 @Override
43 public void pay(Order order) {
44
45     if (email != null && !email.equals("") && password != null &&
46         !password.equals("")) {
47         out.print("Payment Successful");
48
49         order.setTotalCost(0);
50     } else {
51         out.print("Wrong email or password! Payment Failed");
52     }
53 }
54
55 @Override
56 public void provideDetails() {
57     try {
58         email = reader.readLine();
59         password = reader.readLine();
60     } catch (IOException e) {
61         email = "";
62         password = "";
63     }
64 }
65
66
67 }
```

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.

```
//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)");
}
```


Finished after 0.215 seconds

Runs: 3/3  Errors: 0  Failures: 0

✓  PaymentTest [Runner: JUnit 5] (0.048 s)

✓  testEnoughCash() (0.038 s)

✓  testPayManyTimes() (0.007 s)

✓  testNotEnoughCash() (0.001 s)