NAME- Harshita Totala ROLL NO- SEITA14 DIV- IT A
SUBJECT- Object Oriented Programming

ASSIGNMENT NO. 11

Title: Strategy Design Pattern

Aim: Implement and apply Strategy Design pattern for simple Shopping Cart where three payment strategies are used such as Credit Card, PayPal, BitCoin. Create the interface for strategy pattern and give concrete implementation for payment.

Objectives: To learn concept of strategy design pattern

Theory:

1. What is strategy design pattern
2. Design pattern representation
3. Intent
4. Solution of given context with diagram

In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern. In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object.The strategy object changes the executing algorithm of the context object.We are going to create a Strategy interface defining an action and concrete strategy classes implementing the Strategy interface. Context is a class which uses a Strategy.

Step 1
Create an interface.
Strategy.java

```
public interface Strategy
 {   publicintdoOperation(int
num1, int num2); }
```

Step 2
Create concrete classes implementing the same interface. OperationAdd.java

```
public class OperationAdd implements Strategy{
@Override
publicintdoOperation(int num1, int num2)
{
return num1 + num2;
```

```
}
}
```
OperationSubstract.java
```
public class OperationSubstract implements Strategy
{   @Override
publicintdoOperation(int num1, int num2)
 {
return num1 - num2;
 }
}
```

OperationMultiply.java
```
 public class OperationMultiply implements Strategy
{ @Override
publicintdoOperation(int num1, int num2)
{ return num1 * num2;
}
}
```
Step 3
Create Context Class.

Context.jav
a public
class
Context
```
{ private Strategy
strategy; public
Context(Strategy
strategy){
this.strategy =
strategy;
}
publicintexecuteStrategy(int num1, int num2){ return
strategy.doOperation(num1, num2);
}
}
```
Step 4
 Use the Context to see change in behaviour when it changes its
Strategy.
StrategyPatternDemo.jav
a public class
StrategyPatternDemo {
public static void
main(String[] args) {

```
Context context = new Context(new
OperationAdd()); System.out.println("10 + 5 = " +
context.executeStrategy(10, 5)); context = new
Context(new OperationSubstract());
System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

context = new Context(new OperationMultiply());
System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
}
}
```

Program:

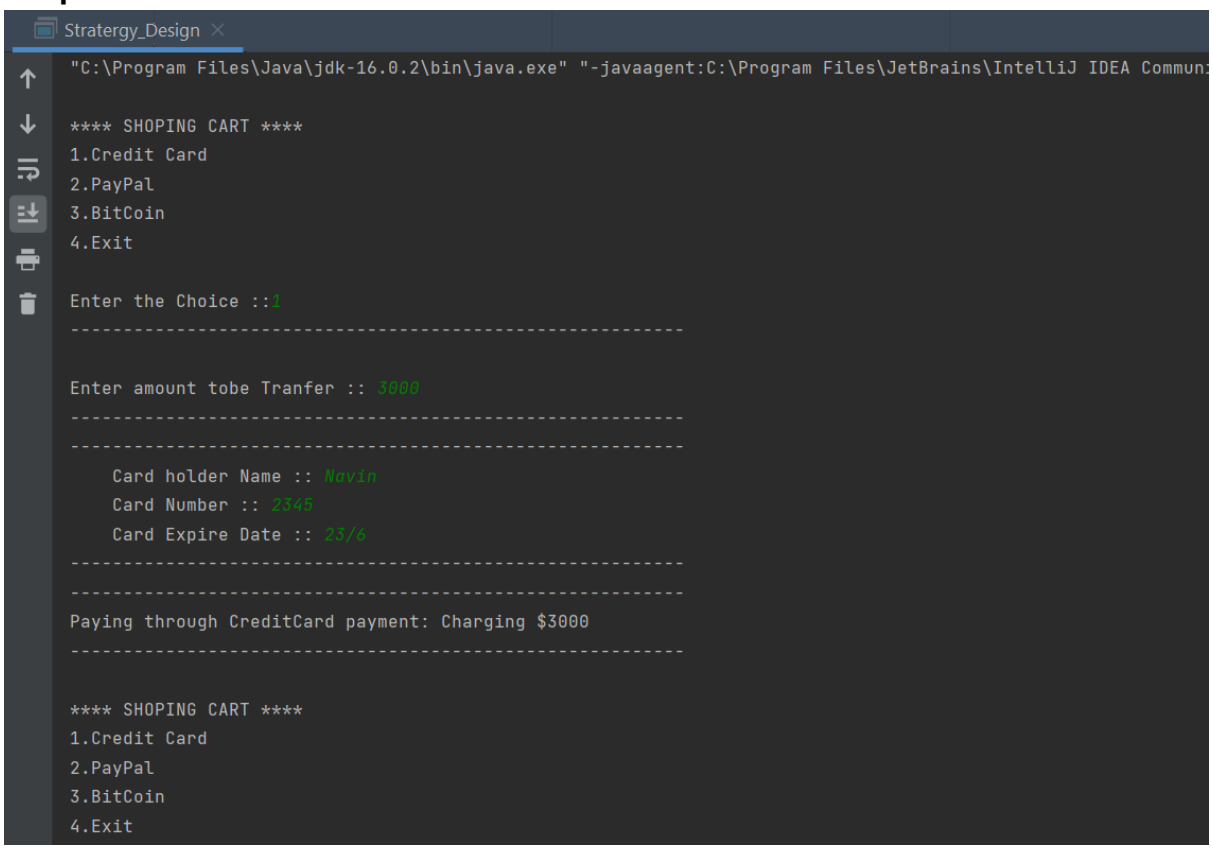
```
package com.company.assignment;



import java.util.Scanner;

//======================== INTERFACE PaymentProcessor
========================//
interface PaymentProcessor {

    void pay(int amount);//interface method pay

}

//======================== CLASS CreditCard ========================//
//implementing PaymentProcessor interface
class CreditCard implements PaymentProcessor {
    Scanner sc =new Scanner (System.in);//creating object of scanner class
    String name,ExpDate;//declaration of name,ExpDate
    double CardNo;//declaration of CardNo

    //Constructor of CreditCard class
    CreditCard(){
        super();//calling parent class constructor
        System.out.println("-----------------------------------------------
-----------");
        System.out.print("\tCard holder Name :: ");//printing on console
        this.name =sc.next();//taking Card holder Name as input from user
        System.out.print("\tCard Number :: ");//printing on console
        this.CardNo =sc.nextDouble();//taking Card Number as input from
user
        System.out.print("\tCard Expire Date :: ");//printing on console
        this.ExpDate =sc.next();//taking Card Expire Date as input from
user
        System.out.println("-----------------------------------------------
-----------");
    }

    @Override
    public void pay(int amount) {  //method for payment
        System.out.println("-----------------------------------------------
-----------");
        System.out.println("Paying through CreditCard payment: Charging $"
+ amount);
```

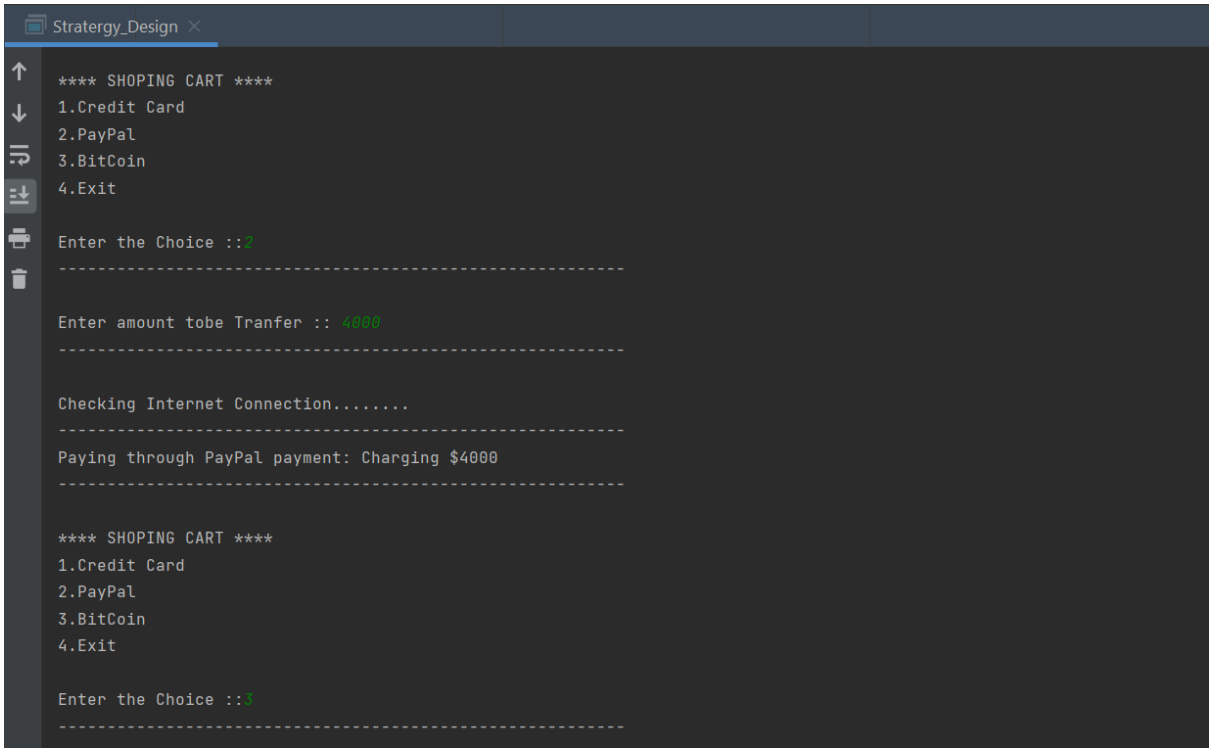
```
        System.out.println("-----------------------------------------------
-----------");
    }


}


//======================== CLASS PayPal ========================//
//implementing PaymentProcessor interface
class PayPal implements PaymentProcessor {

    //Constructor of PayPal class
    PayPal(){
        super();//calling parent class constructor
        System.out.println("\nChecking Internet Connection........");
    }

    @Override
    public void pay(int amount) {  //method for payment
        System.out.println("-----------------------------------------------
-----------");
        System.out.println("Paying through PayPal payment: Charging $" +
amount);
        System.out.println("-----------------------------------------------
-----------");
    }

}

//======================== CLASS BitCoin ========================//
//implementing PaymentProcessor interface
class BitCoin implements PaymentProcessor {
    Scanner sc =new Scanner (System.in);//creating object of scanner class
    String add;//declaration of add

    //Constructor of BitCoin class
    BitCoin(){
        super();//calling parent class constructor
        System.out.print("\nEnter Transaction 'Input Address' ::
");//asking user of address
        add= sc.next();//taking 'INPUT ADDRESS' as input from user

    }

    @Override
    public void pay(int amount) {  //method for payment
        System.out.println("-----------------------------------------------
-----------");
        System.out.println("Paying through BitCoin payment: Charging $" +
amount);
        System.out.println("-----------------------------------------------
-----------");
    }

}


//======================== CLASS Order ========================//

class Order {
```

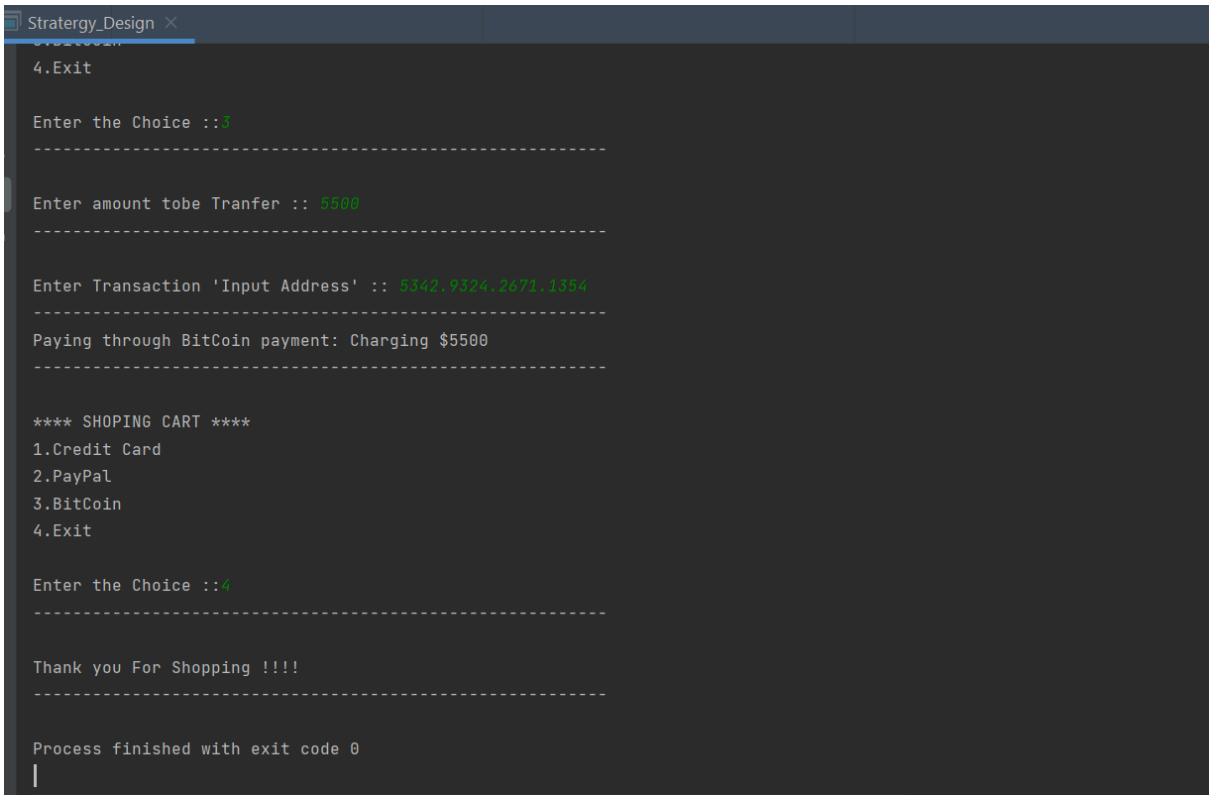
```
    private final PaymentProcessor paymentProcessor;//declaration of
paymentProcessor object
    private final int amount;//declaration of amount

    //Order Method
    public Order(int amount, PaymentProcessor paymentProcessor) {
        this.amount = amount;//storing value
        this.paymentProcessor = paymentProcessor;//storing value
    }

    //process Method
    public void process() {
        paymentProcessor.pay(amount);//calling pay method
    }

}


//======================== CLASS Main ========================//
public class Stratergy_Design {
    //calling static void main method
    public static void main(String[] args) {
        int c,amt=0;//declaration of c, amt
        Order order;//reference of order assign to order obj
        Scanner sc = new Scanner(System.in);//creating object of scanner
class
        while(true) {//while loop for menu driven
            System.out.println();
            //menu bar
            System.out.println("**** SHOPING CART ****");
            System.out.print("1.Credit Card \n2.PayPal \n3.BitCoin
\n4.Exit");
            System.out.print("\n\nEnter the Choice ::");
            c=sc.nextInt();//taking input from user
            System.out.println("-----------------------------------------
---------------");
            if(c==1||c==2||c==3) {//check whether 0<c<4
                System.out.print("\nEnter amount tobe Tranfer :: ");
                amt = sc.nextInt();//taking amt as input from user
                System.out.println("-------------------------------------
------------------");
            }
            //switch case
            switch(c) {
                case 1://for input c ==1
                    order = new Order(amt, new CreditCard());//creating obj
of order class
                    order.process();//calling process method of order class
                    break;

                case 2://for input c == 2
                    order = new Order(amt, new PayPal());//creating obj of
order class
                    order.process();//calling process method of order class
                    break;

                case 3://for input c == 3
                    order = new Order(amt, new BitCoin());//creating obj of
order class
                    order.process();//calling process method of order class
```

```
                    break;

                case 4:
                    System.out.println("\nThank you For Shopping !!!!
");//printing on console
                    System.out.println("----------------------------------
------------------------");
                    return;//stop execution of program

                default:
                    System.out.println("Invalid Payment Mode !!!");//
default
                    System.out.println("----------------------------------
------------------------");
            }
        }
    }
}
```

Output:


Stratergy_Design
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Communi
**** SHOPING CART ****
1.Credit Card
2.PayPal
3.BitCoin
4.Exit
Enter the Choice ::1
Enter amount tobe Tranfer :: 3000
Card holder Name :: Navin
Card Number :: 2345
Card Expire Date :: 23/6
Paying through CreditCard payment: Charging $3000
**** SHOPING CART ****
1.Credit Card
2.PayPal
3.BitCoin
4.Exit





Conclusion- Hence, we have applied the concept of class, object and constructor and perform Strategy Design pattern.