**Title: Strategy Design Pattern**

# Assignment. 11

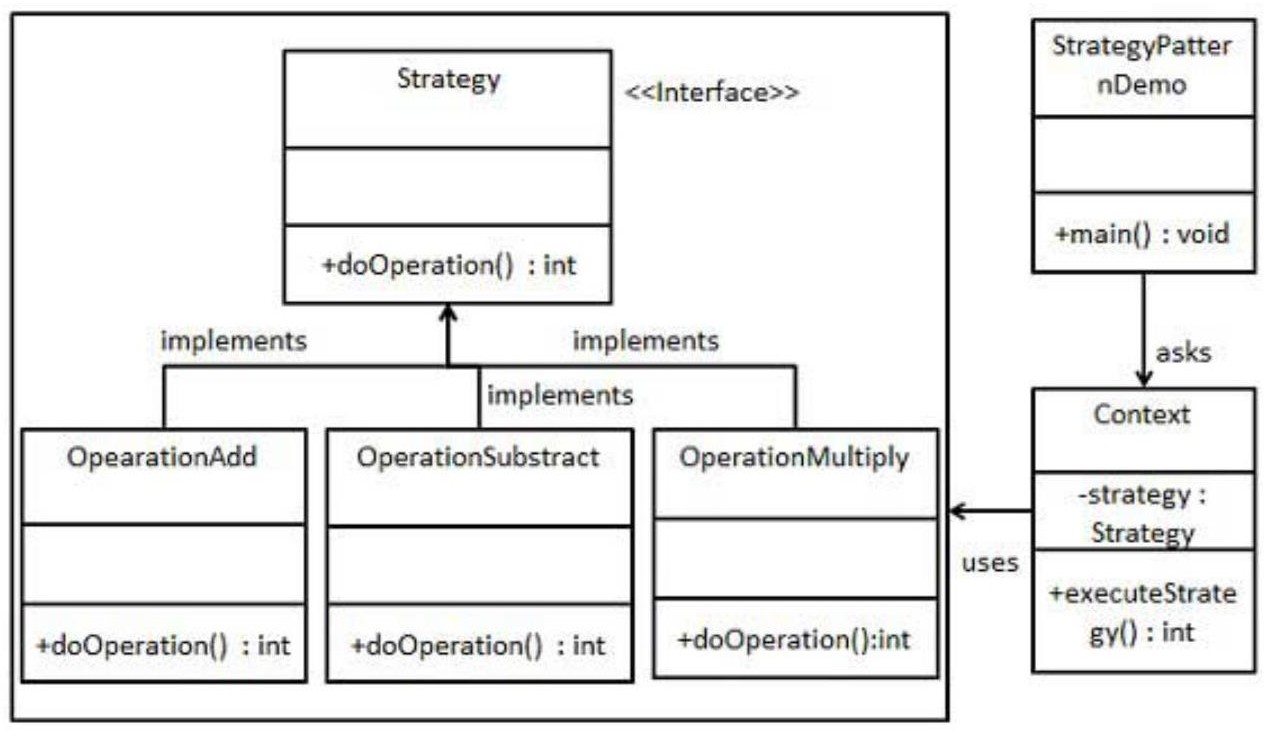
**Aim:** Implement and apply Strategy Design pattern for simple Shopping Cart where three payment strategies are used such as Credit Card, PayPal, BitCoin. Create the interface for strategy pattern and give concrete implementation for payment.

**Objectives:** To learn the 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.java

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.java

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));

}

}

Output:

10 + 5 = 15

10 - 5 = 5

10 \* 5 = 50

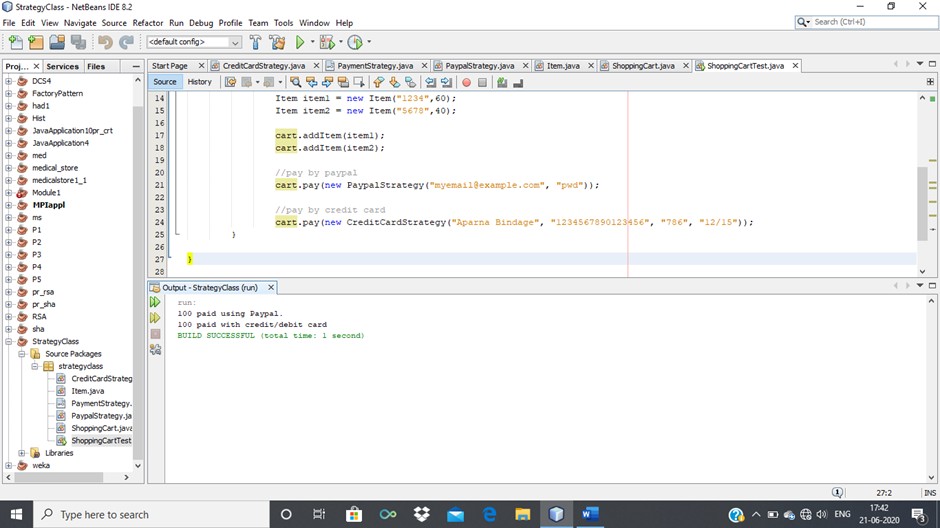
**Output:**

Strategy Pattern –

100 paid using Paypal.

100 paid with credit/debit card

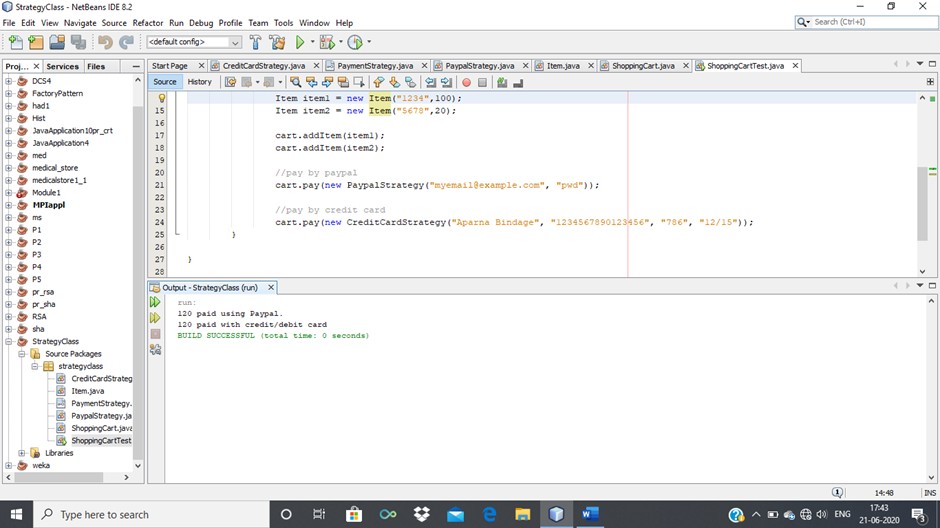
BUILD SUCCESSFUL (total time: 1 second)



120 paid using Paypal.

120 paid with credit/debit card

BUILD SUCCESSFUL (total time: 0 seconds)



## Test case or Validation:

1. Identify an algorithm (i.e. a behavior)
2. Specify the signature for that algorithm in an interface.
3. Bury the alternative implementation details in derived classes.

## Frequently Ask Question:

1. Give intents of Strategy Design pattern
2. How problems can be designed and solved using design pattern
3. Represent Solution of real world problem using Strategy design pattern
4. Give Real-World Analogy of strategy design pattern .
5. Design application by applying the Strategy design pattern .
6. Represent and implement Strategy design pattern to perform mathematical operations such as add,sub,mul,div
7. Represent and implement Strategy design pattern for sorting operation such as Quick sort ,Merge sort etc
8. Represent and implement Strategy design pattern for searching techniques such as Sequential Search, Binary Search etc.

ALGORITHM:

1 MAKE A PAYMENT PROCESS CONSTRUCTOR CLASS

2 IMPLEMENT CREDIT CARD CLASS WITH PAYMENT PROCESSOR

3 IMPLEMENT PAYPALCLASS WITH PAYMENT PROCESSOR

4 IMPLEMENT BITCOIN CLASS WITH PAYMENT PROCESSOR

5 IN MAIN CLASS CALL ALL THE FUNCTION.

6 END

CODE:

package com.company;  
import java.util.Scanner;  
  
interface PaymentProcessor {  
 void pay(int amount);  
}  
  
class CreditCard implements PaymentProcessor {  
 Scanner sc =new Scanner (System.*in*);  
 String name,ExpDate;  
 double CardNo;  
  
 CreditCard(){  
 super();  
 System.*out*.println("----------------------------------------------------------");  
 System.*out*.print("\tCard holder Name :: ");  
 this.name =sc.next();  
 System.*out*.print("\tCard Number :: ");  
 this.CardNo =sc.nextDouble();  
 System.*out*.print("\tCard Expiry Date :: ");  
 this.ExpDate =sc.next();  
 System.*out*.println("----------------------------------------------------------");  
 }  
  
 @Override  
 public void pay(int amount) {  
 System.*out*.println("----------------------------------------------------------");  
 System.*out*.println("Paying through Credit Card payment: Charging $" + amount);  
 System.*out*.println("----------------------------------------------------------");  
 }  
}  
  
class PayPal implements PaymentProcessor {  
  
 PayPal(){  
 super();  
 System.*out*.println("\nChecking Internet Connection........");  
 }  
  
 @Override  
 public void pay(int amount) {  
 System.*out*.println("----------------------------------------------------------");  
 System.*out*.println("Paying through PayPal payment: Charging $" + amount);  
 System.*out*.println("----------------------------------------------------------");  
 }  
 }  
  
class BitCoin implements PaymentProcessor {  
 Scanner sc =new Scanner (System.*in*);  
 String add;  
  
 BitCoin(){  
 super();  
 System.*out*.print("\nEnter Transaction 'Input Address' :: ");  
 add= sc.next();  
 }  
  
 @Override  
 public void pay(int amount) {  
 System.*out*.println("----------------------------------------------------------");  
 System.*out*.println("Paying through BitCoin payment: Charging $" + amount);  
 System.*out*.println("----------------------------------------------------------");  
 }  
}  
  
class Order {  
  
 private final PaymentProcessor paymentProcessor;  
 private final int amount;  
  
 public Order(int amount, PaymentProcessor paymentProcessor) {  
 this.amount = amount;  
 this.paymentProcessor = paymentProcessor;  
 }  
  
 public void process() {  
 paymentProcessor.pay(amount);  
 }  
}  
  
public class AssignmentNo11 {  
 public static void main(String[] args) {  
 int c,amt=0;  
 Order order;  
 Scanner sc = new Scanner(System.*in*);  
 while(true) {  
 System.*out*.println();  
 System.*out*.println("\*\*\*\* SHOPPING CART \*\*\*\*");  
 System.*out*.print("1.Credit Card \n2.PayPal \n3.BitCoin \n4.Exit");  
 System.*out*.print("\n\nEnter your Choice ::");  
 c=sc.nextInt();  
 System.*out*.println("----------------------------------------------------------");  
 if(c==1||c==2||c==3) {  
 System.*out*.print("\nEnter the amount to be Transfer :: ");  
 amt = sc.nextInt();  
 System.*out*.println("----------------------------------------------------------");  
 }  
 switch (c) {  
 case 1 -> {  
 order = new Order(amt, new CreditCard());  
 order.process();  
 }  
 case 2 -> {  
 order = new Order(amt, new PayPal());  
 order.process();  
 }  
 case 3 -> {  
 order = new Order(amt, new BitCoin());  
 order.process();  
 }  
 case 4 -> {  
 System.*out*.println("\nThank you For Shopping !!!! ");  
 System.*out*.println("----------------------------------------------------------");  
 return;  
 }  
 default -> {  
 System.*out*.println("Invalid Payment Mode !!!");  
 System.*out*.println("----------------------------------------------------------");  
 }  
 }  
 }  
 }  
}

OUTPUT:





CONCLUSION

IN THIS ASSIGNMENT WE HAVE SUCCESSFULLY LEARNED THE CONCEPT OF **Strategy Design Pattern** IN JAVA.