Strategy design pattern is one of the **behavioral design pattern**. Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime.

Strategy pattern is also known as **Policy Pattern**. We define multiple algorithms and let client application pass the algorithm to be used as a parameter.

One of the best example of strategy pattern is Collections.sort() method that takes Comparator parameter. Based on the different implementations of Comparator interfaces, the Objects are getting sorted in different ways.

For our example, we will try to implement a simple Shopping Cart where we have two payment strategies – using Credit Card or using PayPal.

First of all we will create the interface for our strategy pattern example, in our case to pay the amount passed as argument.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.ArrayList;

import java.util.List;

interface PaymentStrategy{

public void pay(int amount);

}

/\*

\* Now we will have to create concrete implementations of algorithms for payment useing credit/debit or

\* through paypal

\*/

class CreditCardStrategy implements PaymentStrategy{

private String name;

private String cardNumber;

private String cvv;//card verification value

private String dateOfExpiry;

public CreditCardStrategy(String nm, String cardNumber , String cvv, String dateOfExpiry) {

this.name = nm;

this.cardNumber = cardNumber;

this.cvv = cvv;

this.dateOfExpiry = dateOfExpiry;

}

public void pay(int amount) {

System.out.println(amount + " paid with credit/debit card");

}

}

class PaypalStrategy implements PaymentStrategy{

private String emailId;

private String password;

public PaypalStrategy(String email, String password) {

this.emailId = email;

this.password = password;

}

@Override

public void pay(int amount) {

System.out.println(amount + " paid using paypal");

}

}

/\*

\* Now our strategy pattern example algorithms are ready. we can implement Shopping Cart and payment method will require

\* input as payment Strategy

\*/

class Item{

private String upcCode;

private int price;

public Item(String upc, int cost) {

this.upcCode = upc;

this.price = cost;

}

public String getUpcCode() {

return upcCode;

}

public int getPrice() {

return price;

}

}

class ShoppingCart{

//List of items

List<Item> items;

public ShoppingCart() {

this.items = new ArrayList<Item>();

}

public void addItem(Item item) {

this.items.add(item);

}

public void removeItem(Item item) {

this.items.remove(item);

}

public int calculateTotal() {

int sum = 0;

for(Item item : items) {

sum += item.getPrice();

}

return sum;

}

public void pay(PaymentStrategy paymentMethod) {

int amount = calculateTotal();

paymentMethod.pay(amount);

}

}

/\*

\* Notice that payment method of shopping cart requires payment algorithm as argument and doesn't store it

\* anywhere as insntance variable

\* Lets test our strategy pattern example setup with a simple program

\*/

public class TestStrategyDemo {

public static void main(String[] args) {

ShoppingCart shoppingCart = new ShoppingCart();

Item item1 = new Item("1234",10);

Item item2 = new Item("5678",40);

shoppingCart.addItem(item1);

shoppingCart.addItem(item2);

//pay by paypal

shoppingCart.pay(new PaypalStrategy("myemail.example@gmail.com", "mypwd"));

//pay by credit card

shoppingCart.pay(new CreditCardStrategy("rajat kumar", "3453453345", "777", "12/15/1999"));

}

}