Strategy Design Pattern in Java – Example Tutorial

journaldev.com/1754/strategy-design-pattern-in-java-example-tutorial

July 30, 2013

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

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.

`PaymentStrategy.java`

```
package com.journaldev.design.strategy;

public interface PaymentStrategy {

 public void pay(int amount);
}
```

Now we will have to create concrete implementation of algorithms for payment using credit/debit card or through paypal.

CreditCardStrategy.java

```
package com.journaldev.design.strategy;

public class CreditCardStrategy implements PaymentStrategy {

 private String name;
 private String cardNumber;
 private String cvv;
 private String dateOfExpiry;

 public CreditCardStrategy(String nm, String ccNum, String cvv, String expiryDate){
  this.name=nm;
  this.cardNumber=ccNum;
  this.cvv=cvv;
  this.dateOfExpiry=expiryDate;
 }
 @Override
 public void pay(int amount) {
  System.out.println(amount +" paid with credit/debit card");
 }

}
```

PaypalStrategy.java

```
package com.journaldev.design.strategy;

public class PaypalStrategy implements PaymentStrategy {

 private String emailId;
 private String password;

 public PaypalStrategy(String email, String pwd){
  this.emailId=email;
  this.password=pwd;
 }

 @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.

Item.java

```
package com.journaldev.design.strategy;

public 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;
 }

}
```

ShoppingCart.java

```
package com.journaldev.design.strategy;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

public 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 instance variable.

Let's test our strategy pattern example setup with a simple program.

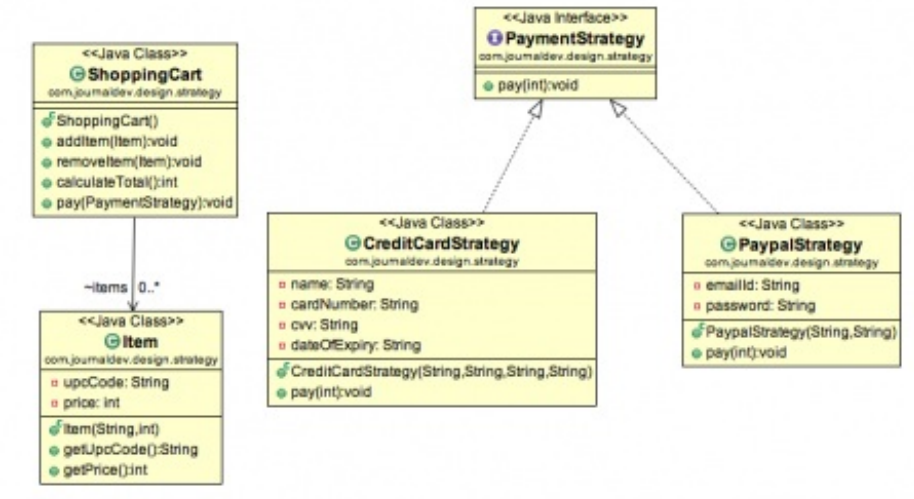
ShoppingCartTest.java

```
package com.journaldev.design.strategy;

public class ShoppingCartTest {

 public static void main(String[] args) {
  ShoppingCart cart = new ShoppingCart();

  Item item1 = new Item("1234",10);
  Item item2 = new Item("5678",40);

  cart.addItem(item1);
  cart.addItem(item2);

  //pay by paypal
  cart.pay(new PaypalStrategy("myemail@example.com", "mypwd"));

  //pay by credit card
  cart.pay(new CreditCardStrategy("Pankaj Kumar", "1234567890123456", "786", "12/15"));
 }

}
```

Output of above program is:

```
50 paid using Paypal.
50 paid with credit/debit card
```

Strategy Design Pattern Class Diagram



Strategy Design Pattern Important Points

- We could have used composition to create instance variable for strategies but we should avoid that as we want the specific strategy to be applied for a particular task. Same is followed in Collections.sort() and Arrays.sort() method that take comparator as argument.

- Strategy Pattern is very similar to **State Pattern**. One of the difference is that Context contains state as instance variable and there can be multiple tasks whose implementation can be dependent on the state whereas in strategy pattern strategy is passed as argument to the method and context object doesn't have any variable to store it.
- Strategy pattern is useful when we have multiple algorithms for specific task and we want our application to be flexible to chose any of the algorithm at runtime for specific task.

That's all for Strategy Pattern in java, I hope you liked it.