**Strategy(Policy) Design Pattern**

Strategy [design pattern](https://www.journaldev.com/1827/java-design-patterns-example-tutorial) 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.

PaymentStrategy.java

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

CreditCardStrategy.java

PaypalStrategy.java

Now our strategy pattern example algorithms are ready. We can implement Shopping Cart and payment method will require input as Payment strategy.

Item.java

ShoppingCart.java

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.

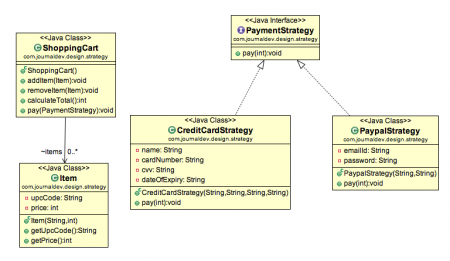
ShoppingCartTest.java

Output of above program is:

50 paid using Paypal.

50 paid with credit/debit card

### Strategy Design Pattern Class Diagram

[](https://cdn.journaldev.com/wp-content/uploads/2013/07/Strategy-Pattern.png)

### 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**](https://www.journaldev.com/1751/state-design-pattern-java). 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.