**When to use Strategy Design Pattern in Java?**

(This gives greater flexibility in design and is in harmony with the [Open/closed principle](http://en.wikipedia.org/wiki/Open/closed_principle) (OCP) that states that classes should be open for extension but closed for modification.)

Strategy pattern in quite useful for implementing set of related algorithms e.g. compression algorithms, filtering strategies etc. Strategy design pattern allows you to create Context classes, which uses Strategy implementation classes for applying business rules. This pattern follow open closed design principle and quite useful in Java. **One example of Strategy pattern from JDK itself is a Collections.sort**() method and [Comparator interface](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html), which is a strategy interface and defines strategy for comparing objects. Because of this pattern, we don't need to modify sort() method (closed for modification) to compare any object, at same time we can implement Comparator interface to define new comparing strategy (open for extension).

http://upload.wikimedia.org/wikipedia/commons/3/39/Strategy\_Pattern\_in\_UML.png



The following example is in [Java](http://en.wikipedia.org/wiki/Java_(programming_language)).

***/\*\* The classes that implement a concrete strategy should implement this.***

***\* The Context class uses this to call the concrete strategy. \*/***

**interface** Strategy {

**int** execute(**int** a, **int** b);

}

***/\*\* Implements the algorithm using the strategy interface \*/***

**class** Add **implements** Strategy {

**public** **int** execute(**int** a, **int** b) {

System.out.println("Called Add's execute()");

**return** a + b; *// Do an addition with a and b*

}

}

**class** Subtract **implements** Strategy {

**public** **int** execute(**int** a, **int** b) {

System.out.println("Called Subtract's execute()");

**return** a - b; *// Do a subtraction with a and b*

}

}

**class** Multiply **implements** Strategy {

**public** **int** execute(**int** a, **int** b) {

System.out.println("Called Multiply's execute()");

**return** a \* b; *// Do a multiplication with a and b*

}

}

*// Configured with a ConcreteStrategy object and maintains*

*// a reference to a Strategy object*

**class** Context {

**private** Strategy strategy;

**public** Context(Strategy strategy) {

**this**.strategy = strategy;

}

**public** **int** executeStrategy(**int** a, **int** b) {

**return** **this**.strategy.execute(a, b);

}

}

***/\*\* Tests the pattern \*/***

**class** StrategyExample {

**public** **static** **void** main(String[] args) {

Context context;

*// Three contexts following different strategies*

context = **new** Context(**new** Add());

**int** resultA = context.executeStrategy(3,4);

context = **new** Context(**new** Subtract());

**int** resultB = context.executeStrategy(3,4);

context = **new** Context(**new** Multiply());

**int** resultC = context.executeStrategy(3,4);

System.out.println("Result A : " + resultA );

System.out.println("Result B : " + resultB );

System.out.println("Result C : " + resultC );

}

}