Design patterns. Behavioural software design pattern

Strategy pattern

1. Design pattern description

The strategy pattern is a behavioural software design pattern that enables selecting an algorithm at runtime. Instead of implementing a single algorithm directly, code receives run-time instructions as to which in a family of algorithms to use.

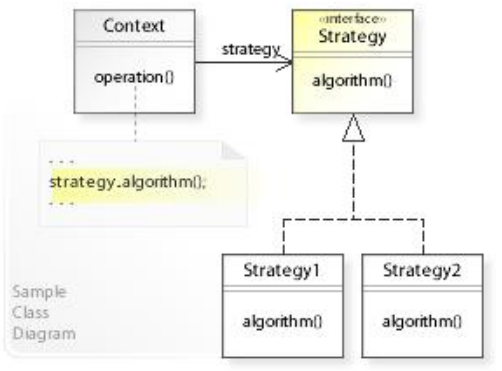
Problems that the strategy pattern solves

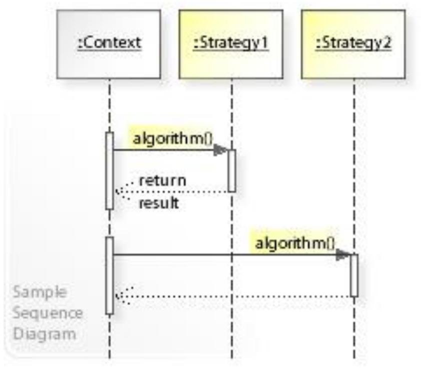
- An inflexible way is to implement (hard-wire) an algorithm directly within the class (Context) that requires (uses) the algorithm. Conditional statements (switch(…)) are needed to switch between different algorithms.
- This commits (couples) the class to particular algorithms at compile-time and makes it impossible to change an algorithm later independently from (**without having to change**) the class. It makes the class more complex, especially if multiple algorithms are needed, and stops the class from being reusable if other algorithms are required.

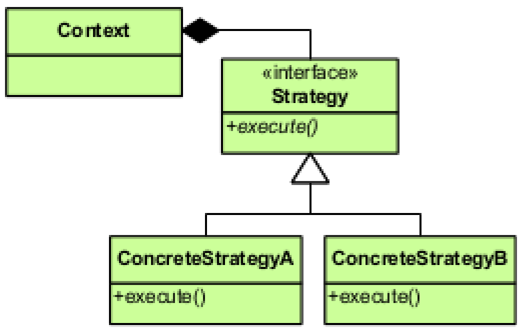
"Hard-wiring all such algorithms into the classes that require them isn't desirable for several reasons:" "Algorithms are often extended, optimized, and replaced during development and reuse."

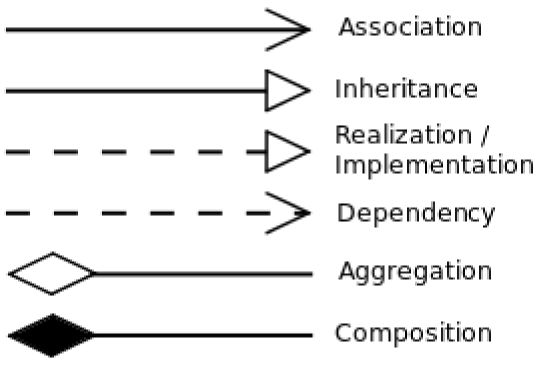
- Strategy lets the algorithm vary independently from clients that use it. Deferring the decision about which algorithm to use until runtime allows the calling code to be more flexible and reusable.

For instance, a class that performs validation on incoming data may use the strategy pattern to select a validation algorithm depending on the type of data, the source of the data, user choice, or other discriminating factors. These factors are not known until run-time and may require radically different validation to be performed. The validation algorithms (strategies), encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.









2. Design pattern example

```
public class StrategyPatternWiki
{
    public static void Main(String[] args)
    {
        // Prepare strategies
        IBillingStrategy normalStrategy    = new NormalStrategy();
        IBillingStrategy happyHourStrategy = new HappyHourStrategy();

        Customer firstCustomer = new Customer(normalStrategy);

        // Normal billing
        firstCustomer.Add(1.0, 1);

        // Start Happy Hour
        firstCustomer.Strategy = happyHourStrategy;
        firstCustomer.Add(1.0, 2);

        // New Customer
        Customer secondCustomer = new Customer(happyHourStrategy);
        secondCustomer.Add(0.8, 1);
        // The Customer pays
        firstCustomer.PrintBill();

        // End Happy Hour
        secondCustomer.Strategy = normalStrategy;
        secondCustomer.Add(1.3, 2);
        secondCustomer.Add(2.5, 1);
        secondCustomer.PrintBill();
    }
}
```

```
class Customer
{
    private IList<double> drinks;

    // Get/Set Strategy
    public IBillingStrategy Strategy { get; set; }

    public Customer(IBillingStrategy strategy)
    {
        this.drinks = new List<double>();
        this.Strategy = strategy;
    }

    public void Add(double price, int quantity)
    {
        drinks.Add(Strategy.GetActPrice(price * quantity));
    }

    // Payment of bill
    public void PrintBill()
    {
        double sum = 0;
        foreach (double i in drinks)
        {
            sum += i;
        }
        Console.WriteLine("Total due: " + sum);
        drinks.Clear();
    }
}

interface IBillingStrategy
{
    double GetActPrice(double rawPrice);
}

// Normal billing strategy (unchanged price)
class NormalStrategy : IBillingStrategy
{
    public double GetActPrice(double rawPrice)
    {
        return rawPrice;
    }
}

// Strategy for Happy hour (50% discount)
class HappyHourStrategy : IBillingStrategy
{
    public double GetActPrice(double rawPrice)
    {
        return rawPrice * 0.5;
    }
}
```

3. Existing pattern in FEM-MAT-OO?

Some parts of the code such as the couples:

-Sh. Func → Filters

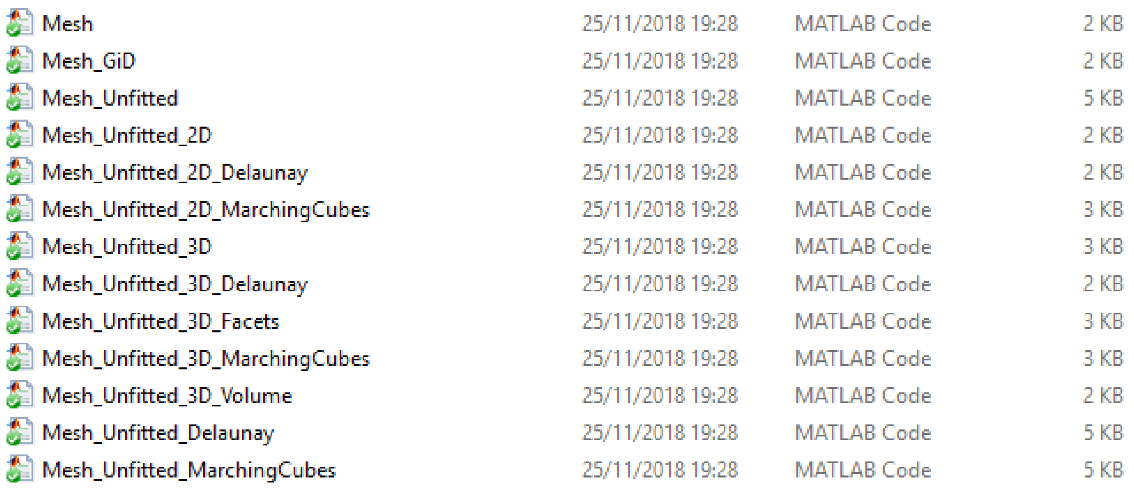
-Optimizer Constrained → Opt. Unconstrained

Have to some extent the general idea of the strategy pattern. However, a pure redefinition on the fly of the filter or opt. Unconstrained in the code would not work as initialization of the objects is needed.

Current FEM-MAT-OO skeleton is not thought as a run-time variant code. This is in part due to the fact that the initial settings define the tools and algorithms that are used in an optimization run.

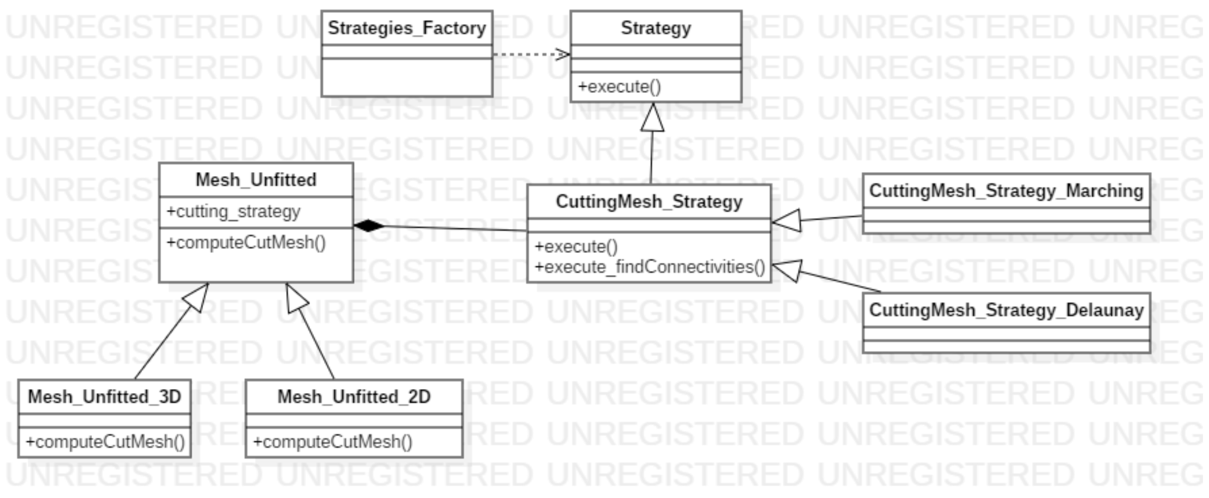
4. Design proposal in FEM-MAT-OO

Current master



Mesh	25/11/2018 19:28	MATLAB Code	2 KB
Mesh_GiD	25/11/2018 19:28	MATLAB Code	2 KB
Mesh_Unfitted	25/11/2018 19:28	MATLAB Code	5 KB
Mesh_Unfitted_2D	25/11/2018 19:28	MATLAB Code	2 KB
Mesh_Unfitted_2D_Delaunay	25/11/2018 19:28	MATLAB Code	2 KB
Mesh_Unfitted_2D_MarchingCubes	25/11/2018 19:28	MATLAB Code	3 KB
Mesh_Unfitted_3D	25/11/2018 19:28	MATLAB Code	3 KB
Mesh_Unfitted_3D_Delaunay	25/11/2018 19:28	MATLAB Code	2 KB
Mesh_Unfitted_3D_Facets	25/11/2018 19:28	MATLAB Code	3 KB
Mesh_Unfitted_3D_MarchingCubes	25/11/2018 19:28	MATLAB Code	3 KB
Mesh_Unfitted_3D_Volume	25/11/2018 19:28	MATLAB Code	2 KB
Mesh_Unfitted_Delaunay	25/11/2018 19:28	MATLAB Code	5 KB
Mesh_Unfitted_MarchingCubes	25/11/2018 19:28	MATLAB Code	5 KB

Proposal


Strategies_Factory
Strategy
+execute()
Mesh_Unfitted
+cutting_strategy
+computeCutMesh()
CuttingMesh_Strategy
+execute()
+execute_findConnectivities()
CuttingMesh_Strategy_Marching
CuttingMesh_Strategy_Delaunay
Mesh_Unfitted_3D
+computeCutMesh()
Mesh_Unfitted_2D
+computeCutMesh()