Template

Design Pattern

Branga Andrei - 10 June 2018

Intent

- Define the skeleton of an algorithm in an operation, deferring some steps to client subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
- Base class declares algorithm 'placeholders', and derived classes implement the placeholders.

In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. This pattern comes under behaviour pattern category.

Explanation

Problem

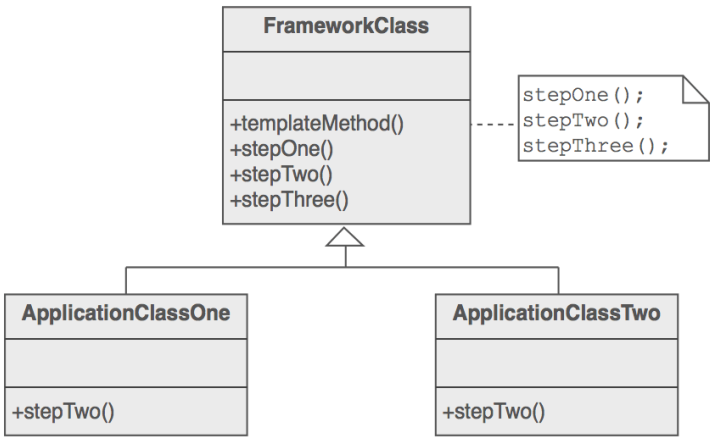
Two different components have significant similarities, but demonstrate no reuse of common interface or implementation. If a change common to both components becomes necessary, duplicate effort must be expended.

Solution / Discussion

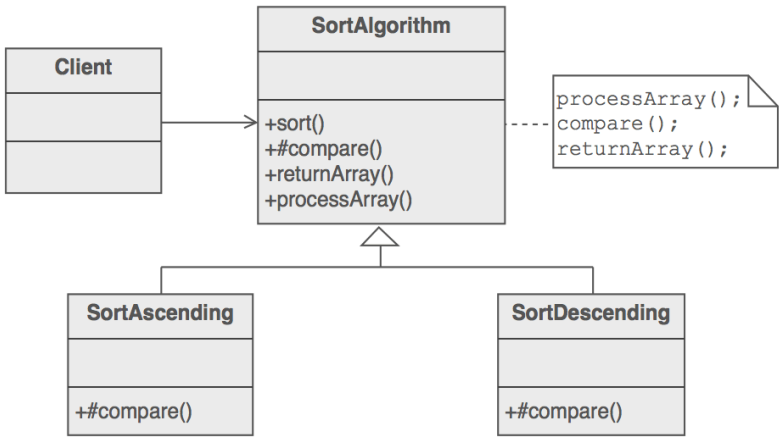
The component designer decides which steps of an algorithm are invariant (or standard), and which are variant (or customisable). The invariant steps are implemented in an abstract base class, while the variant steps are either given a default implementation, or no implementation at all. The variant steps represent "hooks", or "placeholders", that can, or must, be supplied by the component's client in a concrete derived class.

The component designer mandates the required steps of an algorithm, and the ordering of the steps, but allows the component client to extend or replace some number of these steps. Template Method is used prominently in frameworks. Each framework implements the invariant pieces of a domain's architecture, and defines "placeholders" for all necessary or interesting client customisation options. In so doing, the framework becomes the “ centre of

the universe", and the client customisations are simply "the third rock from the sun". This inverted control structure has been affectionately labelled "the Hollywood principle" - "don't call us, we'll call you".



The implementation of `template_method()` is: call `step_one()`, call `step_two()`, and call `step_three()`. `step_two()` is a "hook" method – a placeholder. It is declared in the base class, and then defined in derived classes. Frameworks (large scale reuse infrastructures) use Template Method a lot. All reusable code is defined in the framework's base classes, and then clients of the framework are free to define customizations by creating derived classes as needed.



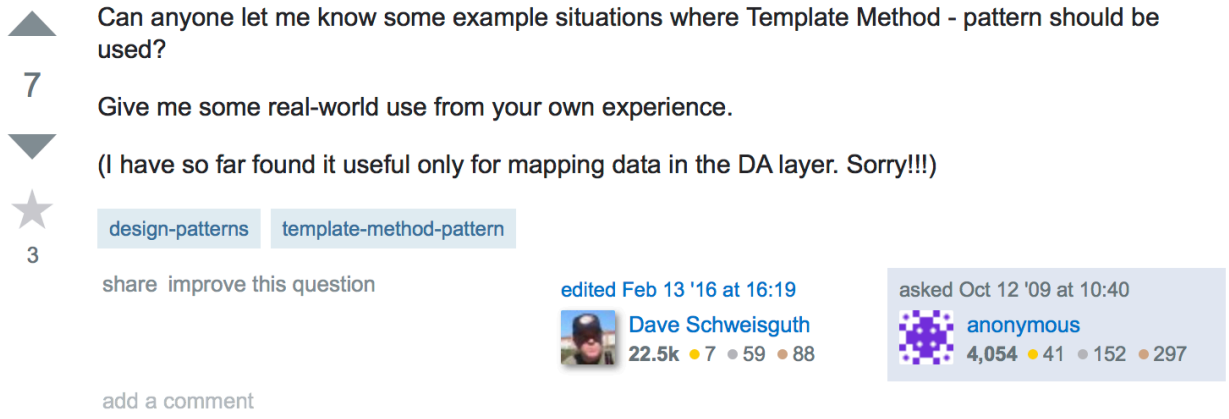
https://sourcemaking.com/design_patterns/template_method - SourceMaking
https://www.tutorialspoint.com/design_pattern/template_pattern.htm - TutorialsPoint

Related Patterns

- Strategy is like Template Method except in its granularity.
- Template Method uses inheritance to vary part of an algorithm. Strategy uses delegation to vary the entire algorithm.
- Strategy modifies the logic of individual objects. Template Method modifies the logic of an entire class.
- Factory Method is a specialisation of Template Method.

Common situations of use

Where should we use Template Method - pattern?



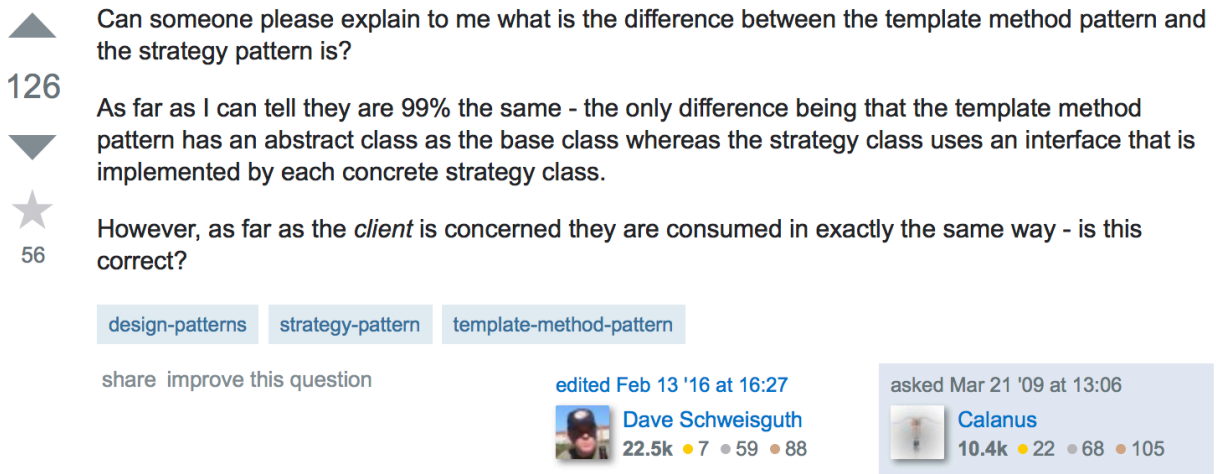
- **When you want your program be "Open For Extension" and also "Closed for Modification".** This means that the behavior of the module can be extended, such that we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications. However, The source code of such a module is inviolate. No one is allowed to make source code changes to it. In following example, you can add new manner of salary calculation (such as Remotely class) without changing the previous codes.

- **When you face many same line of codes that are duplicated through deferring just some steps of your algorithm.** When you are implementing content of a method or function you can find some section of your code that vary from one type to another type. The feature of this sections are that one can redefine or modify these sections of an method or function without changing the algorithm's (method or function) main structure. For example if you want to solve this problem without this pattern you will face this sample:

- **When you as a designer of a framework, want that your clients just to use any executable code that is passed as an argument to your framework, which is expected to call back (execute) that argument at a given time.** This execution may be immediate as in a synchronous callback, or it might happen at a later time as in an asynchronous callback. Lets consider one of famous ones.

https://stackoverflow.com/questions/1553856/where-should-we-use-template-method-pattern - StackOverflow

Confusions



126

56

Can someone please explain to me what is the difference between the template method pattern and the strategy pattern is?

As far as I can tell they are 99% the same - the only difference being that the template method pattern has an abstract class as the base class whereas the strategy class uses an interface that is implemented by each concrete strategy class.

However, as far as the *client* is concerned they are consumed in exactly the same way - is this correct?

design-patterns strategy-pattern template-method-pattern

share improve this question

edited Feb 13 '16 at 16:27
Dave Schweisguth
22.5k ● 7 ● 59 ● 88

asked Mar 21 '09 at 13:06
Calanus
10.4k ● 22 ● 68 ● 105

99

The main difference between the two is when the concrete algorithm is chosen.

With the **Template method pattern** this happens at *compile-time* by *subclassing* the template. Each subclass provides a different concrete algorithm by implementing the template's abstract methods. When a client invokes methods of the template's external interface the template calls its abstract methods (its internal interface) as required to invoke the algorithm.

```
class ConcreteAlgorithm : AbstractTemplate
{
    void DoAlgorithm(int datum) {...}
}

class AbstractTemplate
{
    void run(int datum) { DoAlgorithm(datum); }

    virtual void DoAlgorithm() = 0; // abstract
}
```

In contrast, the **Strategy pattern** allows an algorithm to be chosen at *runtime* by *containment*. The concrete algorithms are implemented by separate classes or functions which are passed to the strategy as a parameter to its constructor or to a setter method. Which algorithm is chosen for this parameter can vary dynamically based on the program's state or inputs.

```
class ConcreteAlgorithm : IAlgorithm
{
    void DoAlgorithm(int datum) {...}
}

class Strategy
{
    Strategy(IAlgorithm algo) {...}

    void run(int datum) { this->algo.DoAlgorithm(datum); }
}
```

In summary:

- Template method pattern: **compile-time** algorithm selection by **subclassing**
- Strategy pattern: **run-time algorithm** selection by **containment**