Template method pattern

In software engineering, the **template method pattern** is a behavioral design pattern that defines the program skeleton of an algorithm in an operation, deferring some steps to subclasses.[1] It lets one redefine certain steps of an algorithm without changing the algorithm's structure![2]

Contents

1 **Overview**

2 **Introduction**

3 **Structure**
- 3.1 UML class diagram
- 3.2 Class diagram

4 **Usage**
- 4.1 Usage with code generators

5 **See also**

6 **References**

7 **External links**

Overview

The Template Method [3] design pattern is one of the twenty-three well-known *GoF design patterns* that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse.

What problems can the Template Method design pattern solve? [4]

- The invariant parts of a behavior should be implemented only once so that subclasses can implement the variant parts.
- Subclasses should redefine only certain parts of a behavior without changing the other parts.

Usually, subclasses control how the behavior of a parent class is redefined, and they aren't restricted to redefine only certain parts of a behavior.

What solution does the Template Method design pattern describe?

Define abstract operations (*primitives*) for the variant parts of a behavior
Define a *template method* that

- implements the invariant parts of a behavior and
- calls abstract operations (*primitives*) that subclasses implement.

The template method controls how subclasses redefine a behavior
This is also referred to as *inversion of control* because subclasses do no longer control how the behavior of a parent class is redefined.

See also the UML class diagram below

Introduction

In the template method of this design pattern, one or more algorithm steps can be overridden by subclasses to allow differing behaviors while ensuring that the overarching algorithm is still followed.[1]

In object-oriented programming, a concrete class is created that provides the steps of an algorithm design. Steps that are considered invariant are implemented inside the base class. The steps that are considered to be variant, are given a default implementation or none at all. These variant steps must be supplied by concrete derived subclasses.[5] Thus the general algorithm is saved in one place but the concrete steps may be changed by the subclasses.

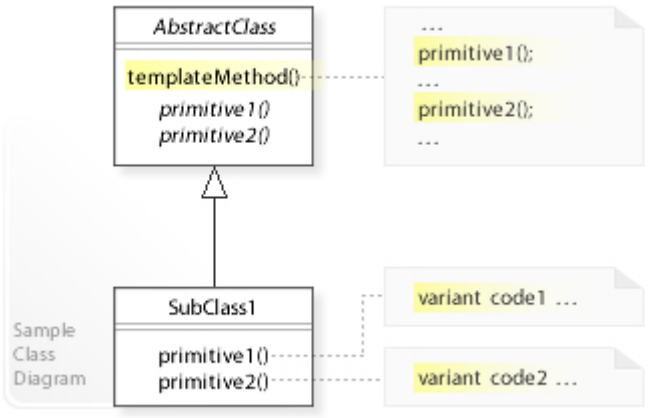
The template method pattern thus manages the larger picture of task semantics, and more refined implementation details of selection and sequence of methods. This larger picture calls abstract and non-abstract methods for the task at hand. The non-abstract methods are completely controlled by the template method, but the abstract methods, implemented in subclasses, provide the pattern's expressive power and degree of freedom. Template method's abstract class may also define hook methods that may be overridden by subclasses.[2]

Some or all of the abstract methods can be specialized in a subclass, allowing the writer of the subclass to provide particular behavior with minimal modifications to the larger semantics. The template method (that is non-abstract) remains unchanged in this pattern, ensuring that the subordinate non-abstract methods and abstract methods are called in the originally intended sequence.

The template method pattern occurs frequently, at least in its simplest case, where a method calls only one abstract method when using object oriented languages. If a software writer uses a polymorphic method at all, this design pattern may be a rather natural consequence. This is because a method calling an abstract or polymorphic function is simply the reason for being of the abstract or polymorphic method. The template method pattern may be used to add immediate present value to the software or with a vision to enhancements in the future.

Structure

UML class diagram

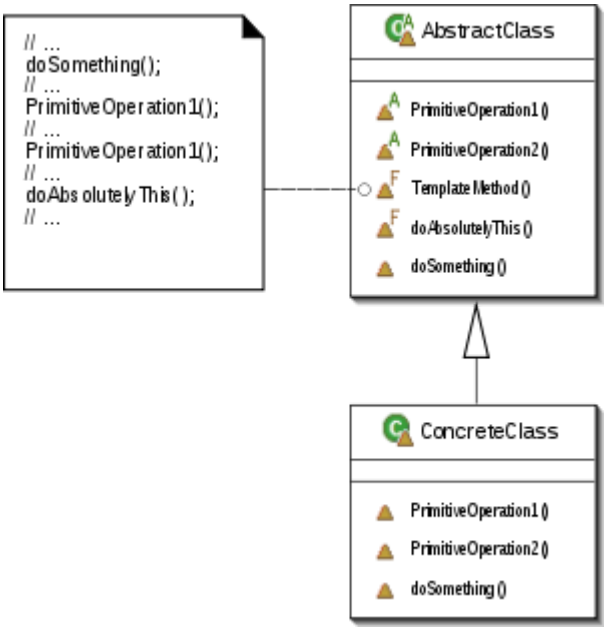


A sample UML class diagram for the Template Method design pattern.[6]

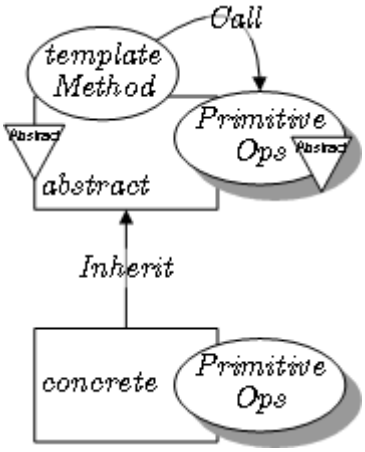
In the above UML class diagram, the `AbstractClass` defines a `templateMethod()` operation that defines the skeleton (template) of a behavior by

- implementing the invariant parts of the behavior and
- calling abstract `primitive1()` and `primitive2()` operations to defer implementing the variant parts to `SubClass1`.

Class diagram



Template method: UML class diagram.



Template Method in LePUS3.[7]

Usage

The template method is used in frameworks, where each implements the invariant parts of a domain's architecture, leaving "placeholders" for customization options. This is an example of inversion of control. The template method is used for the following reasons:[5]

- Let subclasses implement varying behavior (through method overriding).[8]
- Avoid duplication in the code: the general workflow structure is implemented once in the abstract class's algorithm, and necessary variations are implemented in the subclasses.[8]
- Control at what point(s) subclassing is allowed. As opposed to a simple polymorphic override, where the base method would be entirely rewritten allowing radical change to the workflow, only the specific details of the workflow are allowed to change.[8]

Usage with code generators

The template pattern is useful working with auto-generated code. The challenge of working with generated code is that any refinement of the source material will lead to changes in the generated code, which could overwrite hand-written modifications. This may be solved using the Template pattern, by generating abstract code, and making hand-written modifications to a concrete subclass

or implementation class. The abstract code may be in the form of an abstract class in C++, or an interface in Java or C#. The hand-written code would go into a subclass in C++, and an implementing class in Java or C#. When used with code generation, this pattern is sometimes referred to as the Generation Gap pattern.[9]

See also

- Inheritance (computer science)
- Method overriding (programming)
- GRASP (object-oriented design)
- Adapter pattern

References

1. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1994). "Template Method". *Design Patterns*. Addison-Wesley. pp. 325–330. ISBN 0-201-63361-2
2. Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). Hendrickson, Mike; Loukides, Mike, ed. *Head First Design Patterns* (http://shop.oreilly.com/product/9780596007126.do) (paperback). 1. O'REILLY. p. 289, 311. ISBN 978-0-596-00712-6. Retrieved 2012-09-12.
3. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley. pp. 325ff. ISBN 0-201-63361-2
4. "The Template Method design pattern - Problem, Solution, and Applicability" (http://w3sdesign.com/?gr=b10&ugr=proble). *w3sDesign.com*. Retrieved 2017-08-12.
5. "Template Method Design Pattern" (http://sourcemaking.com/design_patterns/template_method). Source Making - teaching IT professional. Retrieved 2012-09-12. "Template Method is used prominently in frameworks."
6. "The Template Method design pattern - Structure" (http://w3sdesign.com/?gr=b10&ugr=struct). *w3sDesign.com*. Retrieved 2017-08-12.
7. LePUS3 legend. Retrieved from http://lepus.org.uk/ref/legend/legend.xml
8. Chung, Carlo (2011). *Pro Objective-C Design Patterns for iOS*. Berkely, CA: Apress. p. 266. ISBN 978-1-4302-3331-2.
9. Vlissides, John (1998-06-22). *Pattern Hatching: Design Patterns Applied* (http://www.informit.com/store/pattern-hatching-design-patterns-applied-9780201432930). Addison-Wesley Professional. pp. 85–101. ISBN 978-0201432930.

External links

- Working with Template Classes in PHP 5
- Template method pattern in UML and in LePUS3 (a formal modelling language)
- Template Method Design Pattern
- Template Method Example
