WIKIPEDIA

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] The template method is one of the twenty-three well-known patterns described in the "Gang of Four" book *Design Patterns*.

Contents

Overview

This pattern has two main parts, and typically uses object-oriented programming:

- The "template method", generally implemented as a base class (possibly an abstract class), which contains shared code and parts of the overall algorithm which are invariant. The template ensures that the overarching algorithm is always followed.[1] In this class, "variant" portions are given a default implementation, or none at all.
- Concrete implementations of the abstract class, which fill in the empty or "variant" parts of the "template" with specific algorithms that vary from implementation to implementation.[3]

At run-time, a concrete class is instantiated. A main method inherited from the base class is called, which then may call other methods defined by both the base class and subclasses. This performs the overall algorithm in the same steps every time, but the details of some steps depend on which subclass was instantiated.

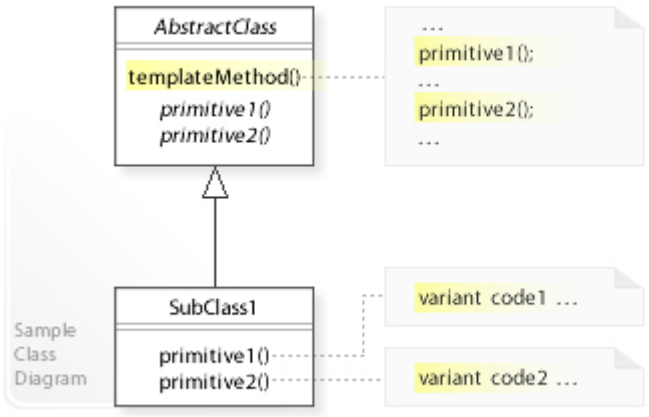
This pattern is an example of inversion of control because the high-level code no longer determines what algorithms to run; a lower-level algorithm is instead selected at run-time.

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] These have a no-op implementation in the abstract class, but provide a "hook" on which to "hang" implementations.

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

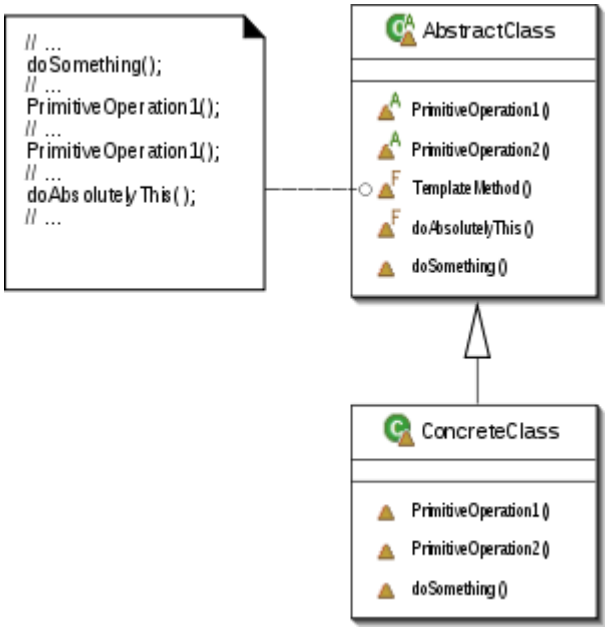


A sample UML class diagram for the Template Method design pattern.[4]

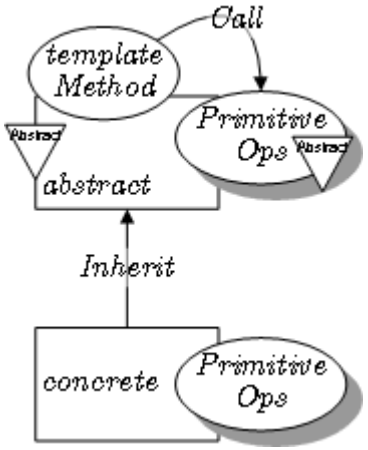
In the above UML class diagram, the `AbstractClass` defines a `templateMethod()` operation that defines the skeleton (template) of a behavior by

- 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.[5]

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:[3]

- Let subclasses implement varying behavior (through method overriding).[6]
- 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.[6]
- 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.[6]

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.[7]

See also

- Inheritance (computer science)
- Method overriding (programming)
- GRASP (object-oriented design)
- Adapter pattern
- Strategy 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, eds. *Head First Design Patterns* (http://shop.oreilly.com/product/9780596007126.do) (paperback). **1**. O'REILLY. pp. 289, 311. ISBN 978-0-596-00712-6. Retrieved 2012-09-12.
3. "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."
4. "The Template Method design pattern - Structure" (http://w3sdesign.com/?gr=b10&ugr=struct). *w3sDesign.com*. Retrieved 2017-08-12.
5. LePUS3 legend. Retrieved from http://lepus.org.uk/ref/legend/legend.xml.
6. Chung, Carlo (2011). *Pro Objective-C Design Patterns for iOS*. Berkely, CA: Apress. p. 266. ISBN 978-1-4302-3331-2.
7. 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

- Six common uses of the template pattern (https://www.codeproject.com/Articles/307452/common-use-of-Template-Design-pattern-Design-pat)
- Working with Template Classes in PHP 5 (http://www.devshed.com/c/a/PHP/Working-with-Template-Classes-in-PHP-5/)
- Template Method Design Pattern (http://sourcemaking.com/design_patterns/template_method)