Template Method

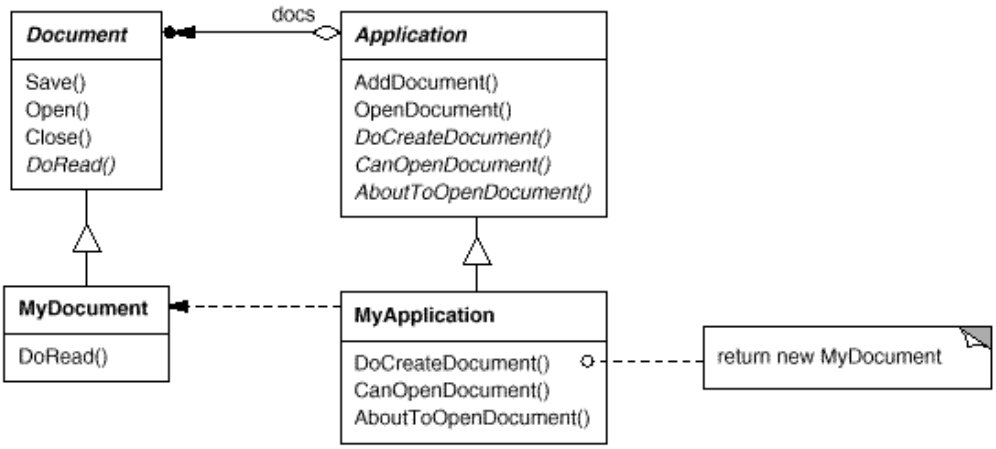
Intent

Define the skeleton of an algorithm in an operation, deferring somesteps to subclasses. Template Method lets subclasses redefinecertain steps of an algorithm without changing the algorithm'sstructure.

Motivation

Consider an application framework that provides Application andDocument classes. The Application class is responsible for openingexisting documents stored in an external format, such as a file. ADocument object represents the information in a document once it'sread from the file.

Applications built with the framework can subclass Application andDocument to suit specific needs. For example, a drawing applicationdefines DrawApplication and DrawDocument subclasses; a spreadsheetapplication defines SpreadsheetApplication and SpreadsheetDocumentsubclasses.



The abstract Application class defines the algorithm for opening andreading a document in its OpenDocument operation:

```
void Application::OpenDocument (const char* name) {
        if (!CanOpenDocument(name)) {
                // cannot handle this document
                return;
        }
```

```
        Document* doc = DoCreateDocument();
        if (doc) {
                _docs->AddDocument(doc);
AboutToOpenDocument(doc);
doc->Open();
doc->DoRead();
        }
}
```

OpenDocument defines each step for opening a document. It checks ifthe document can be opened, creates the application-specific Documentobject, adds it to its set of documents, and reads the Document from afile.

We call OpenDocument a **template method**. A template methoddefines an algorithm in terms of abstract operations that subclassesoverride to provide concrete behavior. Application subclasses definethe steps of the algorithm that check if the document can be opened(CanOpenDocument) and that create the Document (DoCreateDocument).Document classes define the step that reads the document (DoRead).The template method also defines an operation that lets Applicationsubclasses know when the document is about to be opened(AboutToOpenDocument), in case they care.

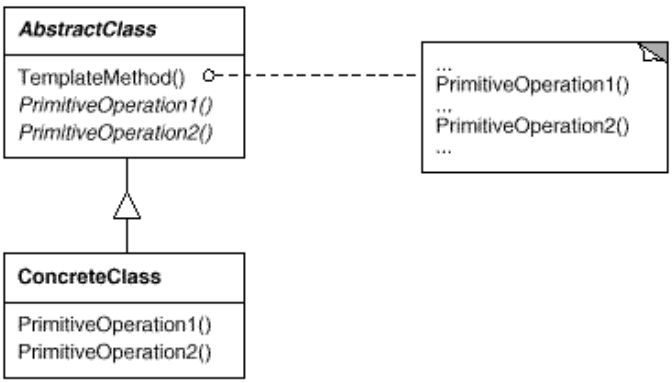
By defining some of the steps of an algorithm using abstractoperations, the template method fixes their ordering, but it letsApplication and Document subclasses vary those steps to suit theirneeds.

Applicability

The Template Method pattern should be used

- to implement the invariant parts of an algorithm once and leave it upto subclasses to implement the behavior that can vary.
- when common behavior among subclasses should be factored and localizedin a common class to avoid code duplication. This is a good example of"refactoring to generalize" as described by Opdyke andJohnson [OJ93]. You first identify thedifferences in the existing code and then separate the differencesinto new operations. Finally, you replace the differing code with atemplate method that calls one of these new operations.
- to control subclasses extensions. You can define a template methodthat calls "hook" operations (see Consequences) at specific points,thereby permitting extensions only at those points.

Structure



Participants

- **AbstractClass** (Application)
  - defines abstract **primitive operations** that concretesubclasses define to implement steps of an algorithm.
  - implements a template method defining the skeleton of an algorithm.The template method calls primitive operations as well as operationsdefined in AbstractClass or those of other objects.
- **ConcreteClass** (MyApplication)
  - implements the primitive operations to carry out subclass-specificsteps of the algorithm.

Collaborations

- ConcreteClass relies on AbstractClass to implement the invariant steps of the algorithm.

Consequences

Template methods are a fundamental technique for code reuse. They areparticularly important in class libraries, because they are the meansfor factoring out common behavior in library classes.

Template methods lead to an inverted control structure that'ssometimes referred to as "the Hollywood principle," that is, "Don'tcall us, we'll call you" [Swe85]. This refers tohow a parent class calls the operations of a subclass and not theother way around.

Template methods call the following kinds of operations:

- concrete operations (either on the ConcreteClass or onclient classes);
- concrete AbstractClass operations (i.e., operations that aregenerally useful to subclasses);
- primitive operations (i.e., abstract operations);
- factory methods (see Factory Method (121)); and
- **hook operations**, which provide default behavior thatsubclasses can extend if necessary. A hook operation often doesnothing by default.

It's important for template methods to specify which operations arehooks (*may* be overridden) and which are abstract operations(*must* be overridden). To reuse an abstract class effectively,subclass writers must understand which operations are designed foroverriding.

A subclass can *extend* a parent class operation's behavior byoverriding the operation and calling the parent operation explicitly:

```
void DerivedClass::Operation () {
        // DerivedClass extended behavior
        ParentClass::Operation();
}
```

Unfortunately, it's easy to forget to call the inherited operation.We can transform such an operation into a template method to givethe parent control over how subclasses extend it. The idea is tocall a hook operation from a template method in the parent class.Then subclasses can then override this hook operation:

```
void ParentClass::Operation () {
        // ParentClass behavior
        HookOperation();
}
```

HookOperation does nothing in ParentClass:

```
void ParentClass::HookOperation () { }
```

Subclasses override HookOperation to extend itsbehavior:

```
void DerivedClass::HookOperation () {
        // derived class extension
}
```

Implementation

Three implementation issues are worth noting:

1. *Using C++ access control.*In C++, the primitive operations that a template method calls can bedeclared protected members. This ensures that they are only called bythe template method. Primitive operations that *must* be overridden aredeclared pure virtual. The template method itself should not beoverridden; therefore you can make the template method a nonvirtualmember function.
2. *Minimizing primitive operations.*An important goal in designing template methods is to minimize thenumber of primitive operations that a subclass must override to fleshout the algorithm. The more operations that need overriding, the moretedious things get for clients.
3. *Naming conventions.*You can identify the operations that should be overridden by adding aprefix to their names. For example, the MacApp framework for Macintoshapplications [App89] prefixes template method names with "Do-":"DoCreateDocument", "DoRead", and so forth.

Sample Code

The following C++ example shows how a parent class can enforce aninvariant for its subclasses. The example comes from NeXT'sAppKit [Add94]. Consider a class View that supportsdrawing on the screen. View enforces the invariant that itssubclasses can draw into a view only after it becomes the "focus,"which requires certain drawing state (for example, colors and fonts) tobe set up properly.

We can use a Display template method to set up this state.View defines two concrete operations,SetFocus and ResetFocus, that set up and clean upthe drawing state, respectively. View's DoDisplayhook operation performs the actual drawing. Display callsSetFocus before DoDisplay to set up the drawingstate; Display calls ResetFocus afterwards torelease the drawing state.

```
void View::Display () {
        SetFocus();
        DoDisplay();
        ResetFocus();
}
```

To maintain the invariant, the View's clients always callDisplay, and View subclasses always overrideDoDisplay.

DoDisplay does nothing in View:

```
void View::DoDisplay () { }
```

Subclasses override it to add their specific drawing behavior:

```
void MyView::DoDisplay () {
        // render the view's contents
}
```

Known Uses

Template methods are so fundamental that they can be found in almostevery abstract class. Wirfs-Brock et al. [WBWW90, WBJ90] provide a good overview anddiscussion of template methods.

Related Patterns

Factory Methods (121) are often called by template methods. In the Motivation example,the factory method DoCreateDocument is called by the template methodOpenDocument.

Strategy (349): Template methods use inheritance to vary part of an algorithm.Strategies use delegation to vary the entire algorithm.