10

Template Method

Diseño del Software

Grado en Ingeniería Informática del Software

2024-2025

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

Patrón de comportamiento, de clases

Define el esqueleto de un algoritmo en una operación, difiriendo algunos pasos hasta las subclases. Permite que éstas redefinan ciertos pasos del algoritmo sin cambiar la estructura del algoritmo en sí.

Motivación

Sea un framework de aplicaciones que proporciona unas clases Application y Document.

La clase Application se encarga de abrir los documentos existentes almacenados en un formato externo, como un fichero.

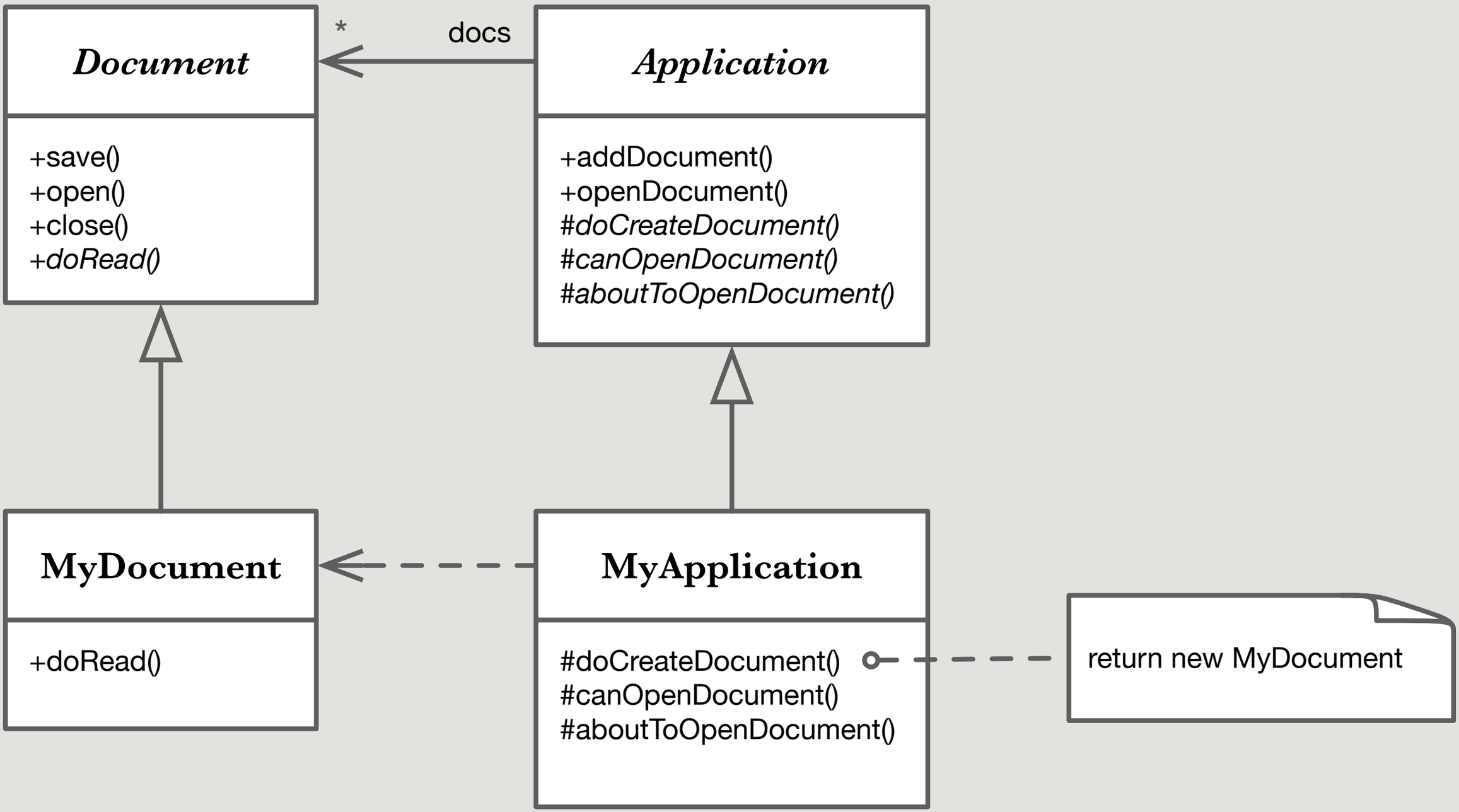
Un objeto Document representa la información de un documento una vez leída del archivo.

Las aplicaciones creadas con el framework pueden heredar de **Application** y **Document** para satisfacer necesidades específicas. Por ejemplo, una aplicación de dibujo definiría las subclases **DrawApplication** y **DrawDocument,** mientras que una aplicación de hoja de cálculo definiría las subclases **SpreadsheetApplication** y **SpreadsheetDocument**.

El método openDocument debe definir cada paso para abrir un documento.

Comprueba si el documento puede abrirse, crea el objeto documento específico de la aplicación, lo añade a su lista de documentos abiertos y lee el contenido desde un fichero.

¿Cómo implementar, de forma genérica, el método openDocument de la clase Application?


Document
+save()
+open()
+close()
+doRead()
*
docs
Application
+addDocument()
+openDocument()
#doCreateDocument()
#canOpenDocument()
#aboutToOpenDocument()
MyDocument
+doRead()
MyApplication
#doCreateDocument()
#canOpenDocument()
#aboutToOpenDocument()
return new MyDocument

Application.java

```
void openDocument(String name) {

    if (!canOpenDocument(name))
        return;

    Document doc = createDocument();

    docs.add(doc);
    aboutToOpenDocument(doc);
    doc.open();
    doc.read();
}
```

openDocument es un template method.

Un método plantilla define un algoritmo en términos de operaciones abstractas que las subclases redefinen para proporcionar un comportamiento concreto.

Las subclases Application definen los pasos del algoritmo que comprueban si el documento se puede abrir (canOpenDocument) y que crean el Documento (createDocument).

Las clases Document definen el paso que lee el documento (read).

El método de plantilla también define una operación que permite a las subclases de Aplicación saber cuándo el documento está a punto de ser abierto (aboutToOpenDocument), en caso de que les interese.

Al definir algunos de los pasos de un algoritmo mediante operaciones abstractas, el método de plantilla **fija su orden**, pero permite a las subclases **variar esos pasos** para adaptarlos a sus necesidades.

Aplicabilidad

Se debería usar el patrón Template Method

Para implementar las partes de un algoritmo que no cambian y dejar que las subclases implementen aquéllas otras que pueden variar.

Como motivo de factorizar código, cuando movemos cierto código a una clase base común para evitar código duplicado.

Este es un buen ejemplo de «refactorizar para generalizar».

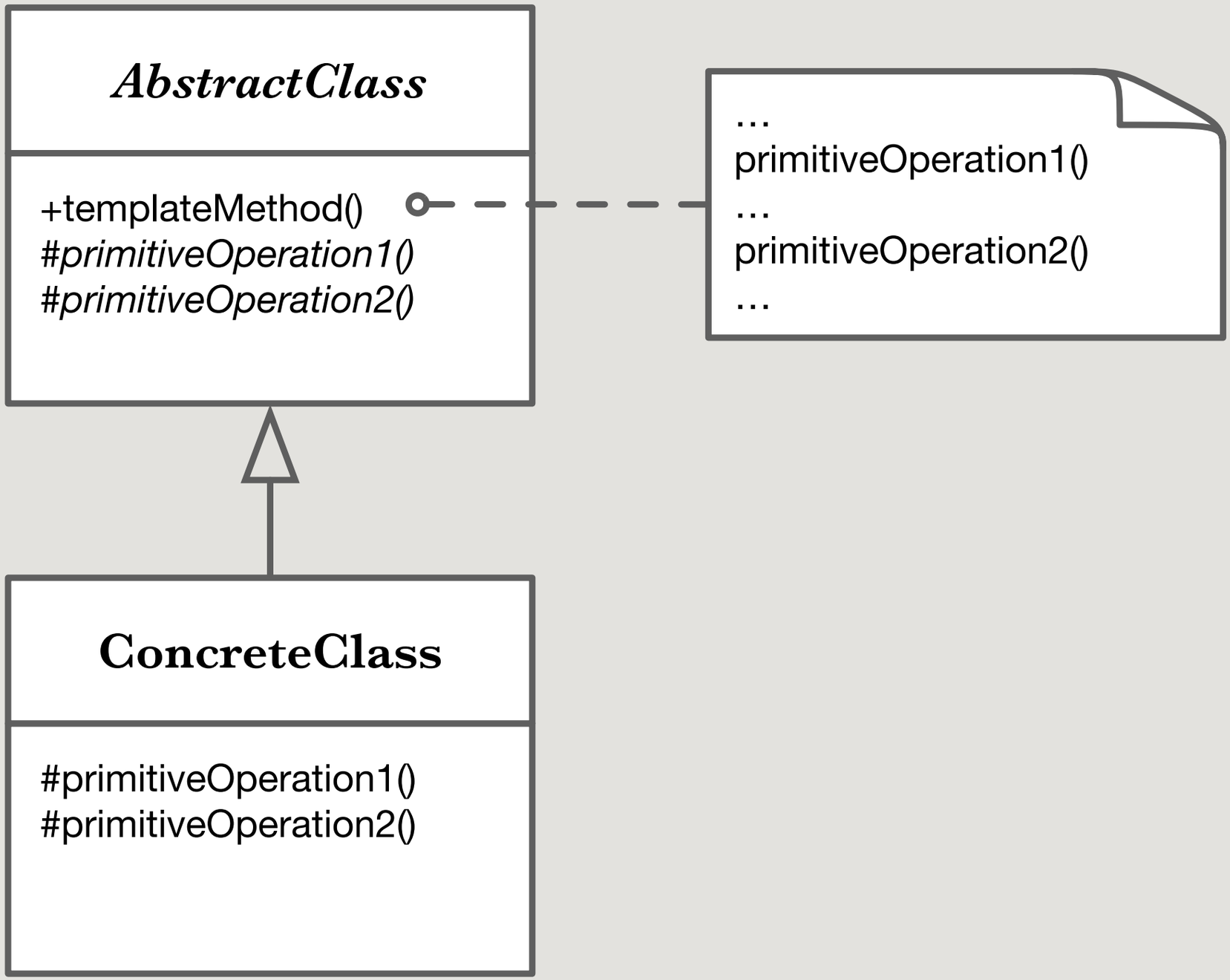
Primero identificamos las **diferencias** en el código existente y luego separamos las diferencias en **nuevas operaciones**. Por último, sustituimos el código diferente por un **método de plantilla que llama** a dichas operaciones.

Para controlar el modo en que las subclases extienden la clase base.

Definiendo un método de plantilla que llame a operaciones «gancho» en puntos específicos, permitiendo así ampliaciones sólo en esos puntos.

Lo veremos en la sección de consecuencias.

Estructura



La estructura del patrón Template Method

Participantes

AbstractClass

(Application)

Define las operaciones primitivas abstractas que redefinirán las subclases para implementar los distintos pasos de la operación.

Implementa un método de plantilla con el esqueleto del algoritmo.

El método de plantilla llama a operaciones primitivas, así como a operaciones definidas en AbstractClass o a las de otros objetos.

ConcreteClass

(MyApplication)

Implementa las operaciones primitivas para llevar a cabo los pasos específicos de la subclase del algoritmo.

Colaboraciones

ConcreteClass se apoya en AbstractClass para implementar los pasos invariantes del algoritmo.

Consecuencias

Son una técnica fundamental para la reutilización de código.

Factorizar código común para evitar la lógica duplicada.

DON'T CALL US

WE'LL CALL YOU

Principio de Hollywood

Dan lugar a una estructura de inversión de control que se conoce como «el principio de Hollywood».

Es la clase padre quien llama a las operaciones de una subclase y no al revés.

Inversion Of Control

26 June 2005

Martin Fowler

APPLICATION ARCHITECTURE
API DESIGN
OBJECT COLLABORATION DESIGN

Inversion of Control is a common phenomenon that you come across when extending frameworks. Indeed it's often seen as a defining characteristic of a framework.

Let's consider a simple example. Imagine I'm writing a program to get some information from a user and I'm using a command line enquiry. I might do it something like this

```
#ruby
puts 'What is your name?'
name = gets
process_name(name)
puts 'What is your quest?'
quest = gets
process_quest(quest)
```

In this interaction, my code is in control: it decides when to ask questions, when to read responses, and when to process those results.

One important characteristic of a framework is that the methods defined by the user to tailor the framework will often be called from within the framework itself, rather than from the user's application code. The framework often plays the role of the main program in coordinating and sequencing application activity. This inversion of control gives frameworks the power to serve as extensible skeletons. The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application.

—Ralph Jhonson and Brian Foote

Ralph Jhonson y Brian Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, June/Jule 1988, Vol. 1, n. 2, pp. 22-35

Los métodos de plantilla pueden llamar a los siguientes tipos de operaciones

Operaciones concretas de otras clases o llamadas a métodos privados de la propia clase

Como cualquier otro método. No requieren mucha explicación y no son esenciales para el patrón. En el ejemplo de motivación, addDocument y read, de Document, son ejemplos de este tipo de métodos, como lo sería cualquier llamada a un método privado de la clase abstracta donde está definido el método del patrón.

Operaciones concretas en la propia clase base abstracta

Permiten proporcionar una implementación predeterminada para ciertas operaciones de la clase abstracta, pero que las subclases pueden redefinir si lo necesitan (siempre y cuando se definan como públicas o protegidas).

Por ejemplo, canOpenDocument se podría implementar en la case abstracta para que devolviera siempre verdadero, y que las clases específicas de aplicación que lo necesiten hagan las comprobaciones necesarias.

Application.java

```
protected boolean canOpenDocument(String name) {
    return true;
}
```

Implementación por omisión de un paso del algoritmo

Operaciones abstractas

Que el libro denomina “primitivas”

Las partes del algoritmo que las clases concretas deben implementar obligatoriamente.

Por ejemplo, createDocument.

Factory methods

Un caso especial de operación abstract

En su forma típica, los métodos de fábrica siempre se llaman desde dentro de los métodos de plantilla.

Por ejemplo, createDocument.

Application.java

```
protected abstract Document createDocument();
```

Los métodos de fabricación se llaman desde un método de plantilla

Operaciones «de enganche»

Una especie de paso opcional

Pueden considerarse una opción intermedia entre las operaciones concretas y las primitivas. En lugar de proporcionar una implementación por omisión, estos métodos se dejan vacíos. De esta forma, las subclases concretas pueden redefinirlos, pero no tienen por qué hacerlo.

Es como si las subclases inyectaran código en puntos específicos del método de plantilla de la clase base.

Por ejemplo, aboutToOpenDocument permitiría a las distintas implementaciones de un framework hacer algo en ese punto concreto, en caso de que lo necesitaran.

Viene del inglés «hook». Probablemente una mejor forma de referirse a ellos en español, aunque menos literal, sería como métodos de extensión.

Application.java

```
protected void aboutToOpenDocument(Document doc) {
    // hook method
}
```

Los métodos gancho son una forma de permitir a las subclases realizar pasos opcionales en puntos específicos del algoritmo

Implementación

Aprovechar los mecanismos de control de acceso del lenguaje de programación

Los métodos abstractos deberíamos declararlos **protected**.

Si quisiéramos reforzar el hecho de que el método plantilla en sí (o cualquier otro método concreto) no se puede redefinir, podríamos declararlos **final**.

Minimizar las operaciones primitivas

Cuantas más operaciones haya que redefinir, más tedioso será para los clientes.

Convenios de nombrado

A veces puede ser útil identificar las operaciones que deben anularse añadiéndoles un prefijo.

Por ejemplo, el framework MacApp para aplicaciones Macintosh prefijaba los nombres de los métodos de las plantillas con **Do-**: DoCreateDocument, DoRead, etcétera.

Código de ejemplo

El ejemplo es de la biblioteca de aplicaciones NeXT.

Consideremos una clase **View** que permite dibujar en la pantalla.

Las subclases de View pueden dibujar en una vista **solo después de obtener el foco**, lo que requiere inicializar el contexto de dibujo, como colores y fuentes.

Del mismo modo, cuando deja de estar activa, es necesario hacer cierta **limpieza** de dicho contexto.

View.cpp

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

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

Las vistas en NeXT aplican estas invariantes a través de un método de plantilla

Usos conocidos

En casi cualquier clase abstracta.

Desempeñan un papel esencial en el diseño de frameworks.

Los métodos de plantillas permiten a los desarrolladores ampliar y personalizar el comportamiento de un framework sin modificar el código del propio framework.

Por otro lado, al controlar el flujo de trabajo general, los frameworks mantienen su robustez a la vez que son lo suficientemente flexibles como para adaptarse a una amplia gama de aplicaciones.

Patrones relacionados

Factory Method

Los métodos de fabricación se llaman la mayoría de las veces desde un Template Method.

En el ejemplo de motivación, el factory method **createDocument** se llama desde el template method **openDocument**.

Strategy

Los métodos de plantilla utilizan la **herencia** para cambiar parte un algoritmo. Las estrategias usan **delegation** para cambiar el algoritmo entero.