2

(IX)

Template Method

(Patrones de diseño)

Diseño del Software

Grado en Ingeniería Informática del Software

Curso 2017-2018

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

Template Method (Método de plantilla)

- **Patrón de comportamiento, de clases**
- **Propósito:**

> *Define el esqueleto de un algoritmo en una operación, defiriendo 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 primera es la responsable de abrir los documentos almacenados en ficheros
  - La segunda representa la información en sí del documento, una vez que ya ha sido leído del fichero
- **Las aplicaciones construidas con el framework redefinirán ambas clases para adaptarlas a sus necesidades concretas**
  - (no será lo mismo la clase que represente documentos de dibujo que de una hoja de cálculo, por ejemplo)

Motivación

- ¿Cómo implementar, de manera genérica, el método `openDocument` de la clase `Application`?

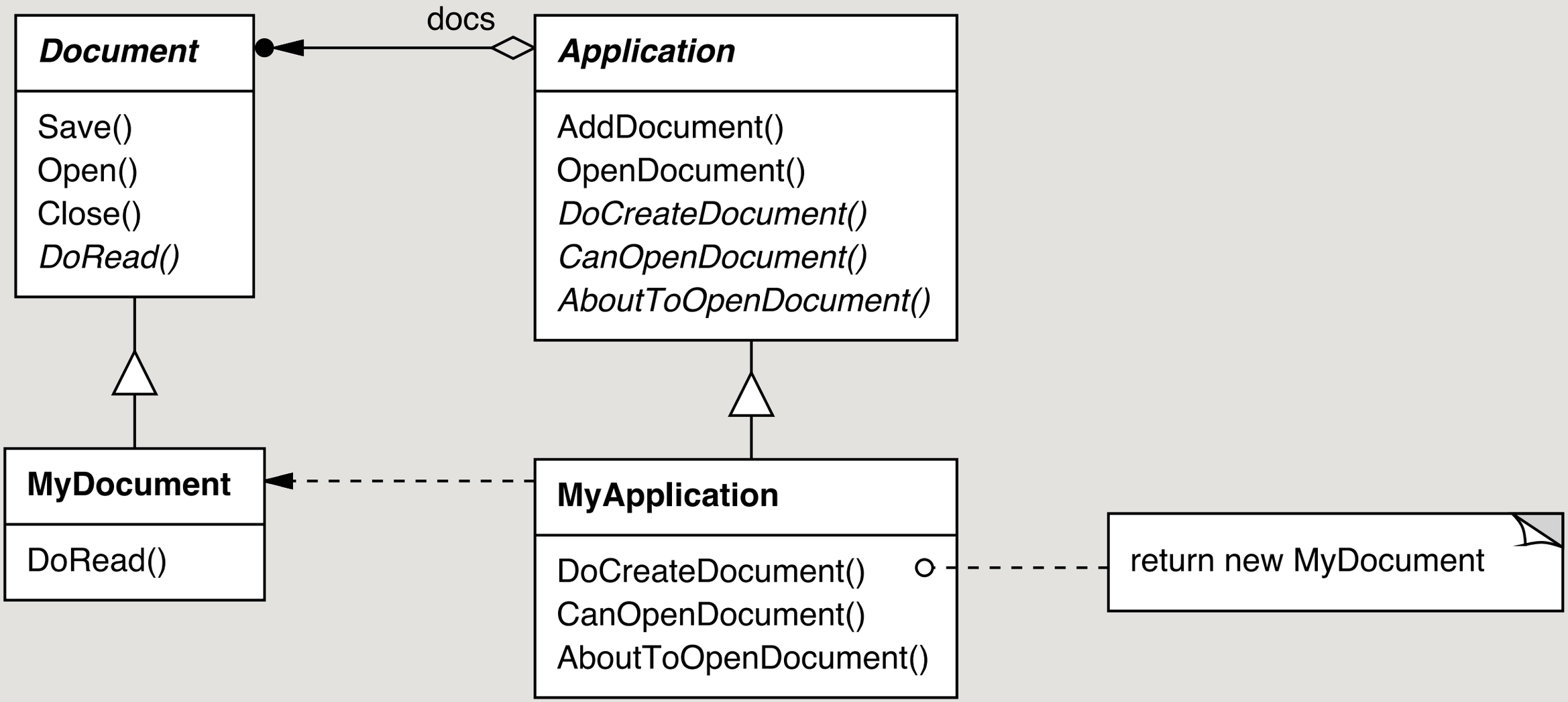
Motivación

```
void Application::OpenDocument (const char* name)
{
    if (!CanOpenDocument(name))
        return;

    Document* doc = DoCreateDocument();

    if (doc) {
        _docs->AddDocument(doc);
        AboutToOpenDocument(doc);
        doc->Open(); doc->DoRead();
    }
}
```

Motivación

- **El método abstracto anterior define todos los pasos necesarios para abrir un documento:**
  - Comprueba si se puede abrir, crea un objeto **`Document`** específico de la aplicación concreta, lo añade al conjunto de documentos activos y, finalmente, lo lee del fichero
  - Lo hace en términos de operaciones abstractas
    - Que las subclases redefinirán convenientemente

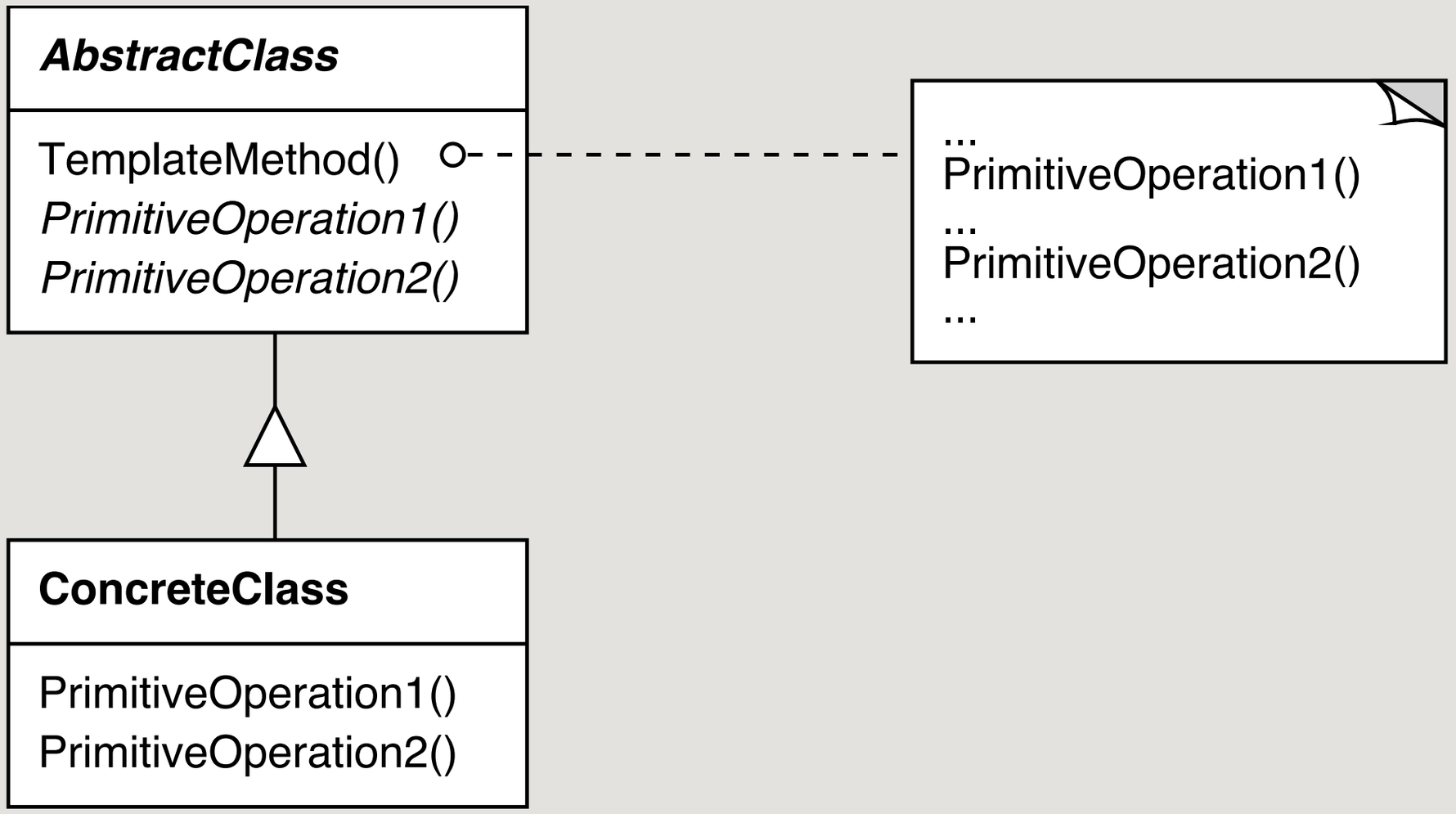
Motivación


docs
Document
Save()
Open()
Close()
DoRead()
Application
AddDocument()
OpenDocument()
DoCreateDocument()
CanOpenDocument()
AboutToOpenDocument()
MyDocument
DoRead()
MyApplication
DoCreateDocument()
CanOpenDocument()
AboutToOpenDocument()
return new MyDocument

Aplicabilidad

- **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**
- **Para controlar el modo en que las subclases extienden la clase base**
  - (Dejando que sea sólo a través de unos métodos de plantilla dados)

Estructura



La estructura del patrón *Template Method*, tal como aparece en el GoF

Participantes

- ***AbstractClass*** **(`Application`)**
  - Define las operaciones primitivas abstractas que redefinirán las subclases
  - Implementa un método de plantilla con el esqueleto del algoritmo
- ***AbstractClass*** **(`MyApplication`)**
  - Implementa las operaciones primitivas

Consecuencias

- **Los métodos de plantilla son una técnica fundamental para la reutilización de código**
- **Inversión de control**
  - Es la clase padre quien llama a operaciones en los hijos

http://martinfowler.com/bliki/InversionOfControl.html

Consecuencias

- **Los métodos de plantilla pueden llamar a los siguientes tipos de operaciones:**
  - Operaciones concretas de las subclases o de otras clases
  - Operaciones concretas en la propia clase base abstracta
  - Operaciones primitivas (es decir, abstractas)
  - Métodos de fabricación
  - Operaciones de enganche (*hook*)
    - Proporcionan comportamiento predeterminado que las subclases pueden redefinir si es necesario. Normalmente, la implementación predeterminada no hace nada.

Operaciones «de enganche»

- **Una subclase puede extender una operación de la clase padre llamando explícitamente a dicha operación:**

```
void operation()
{
    super.operation();
    // DerivedClass extended behavior
}
```

- **Problema**
  - Que nos podemos olvidar de llamar a esa operación del padre

Operaciones «de enganche»

- Podemos hacerlo a la inversa: un método de plantilla en la clase padre que llame a una operación en la subclase:

```
void operation()
{
    // ParentClass behavior
    hookOperation();
}

void hookOperation()
{
}
```

Implementación

- **Hacer a las operaciones primitivas llamadas por el método de plantilla `protected`**
  - Y, a aquéllas que deban ser obligatoriamente redefinidas, abstractas
- **Minimizar el número de operaciones primitivas abstractas**
- **Convenios de nombrado**
  - Es conveniente identificar a las operaciones que deben ser redefinidas anteponiendo un prefijo al nombre (por ejemplo, `Do-`)

Código de ejemplo

- En NeXT AppKit, las clases `View` son las que pueden dibujarse en pantalla, pero sólo después de haberse realizado una serie de operaciones de inicialización de fuentes y colores cuando recibe el *foco*
- Para garantizar que las subclases cumplan esta invariante, podemos definir en `View` `Display` como un método de plantilla:

Código de ejemplo

- En `View`:

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

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

  - Las subclases redefinirán **`DoDisplay`** para dibujarse a sí mismas como corresponda

Patrones relacionados

- *Factory Method*
  - Muchas veces los métodos de fabricación son llamados desde métodos de plantilla
- *Strategy*
  - El patrón *Template Method* usa la herencia para modificar parte de un algoritmo; el *Strategy* usa delegación para cambiar el algoritmo entero