POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

26. *Template Method*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

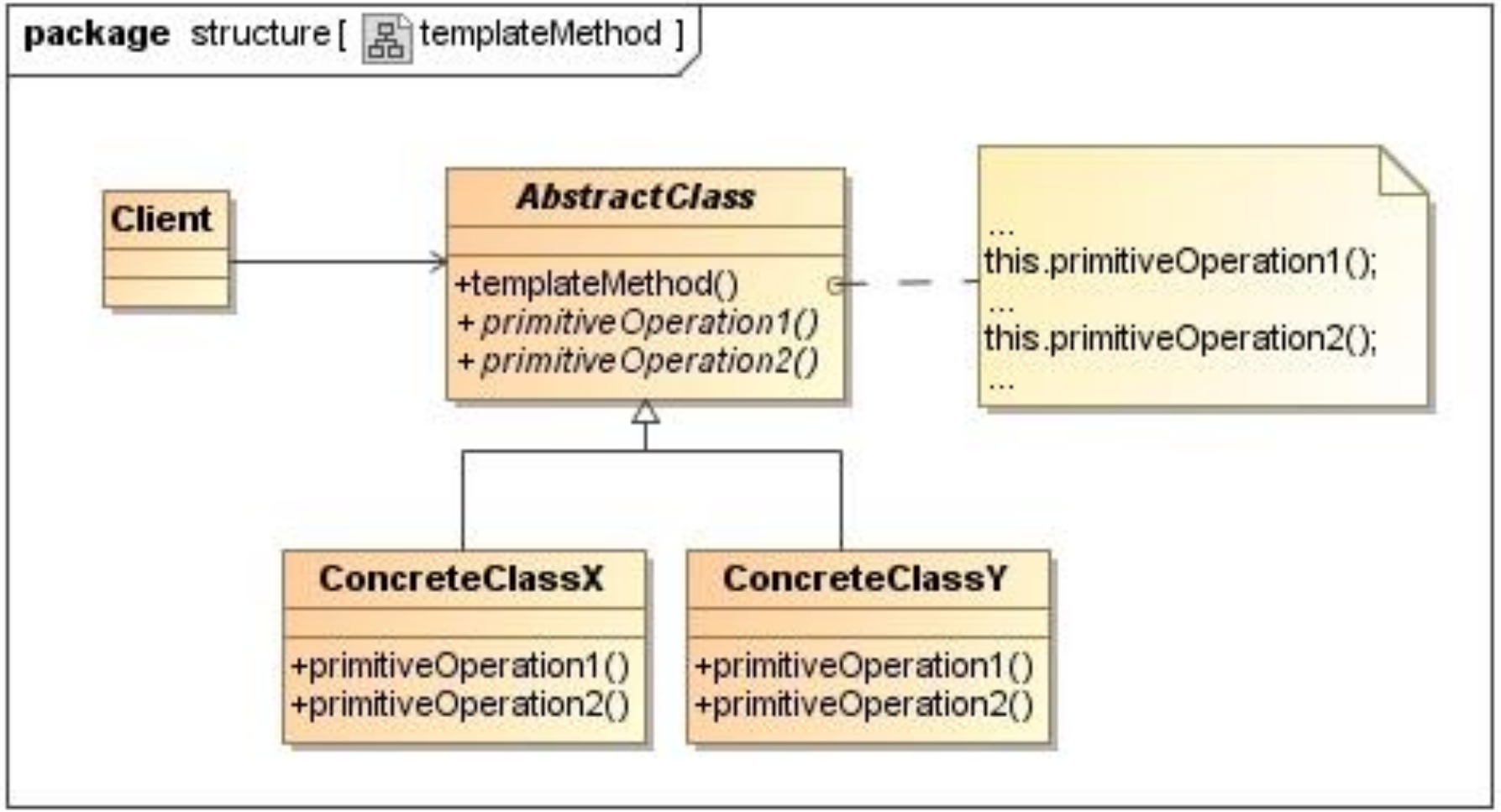
- *La responsabilidad de un profesor se concreta en un algoritmo que contempla la preparación de asignaturas, su impartición y evaluación. Para ello dispone de diversos recursos didácticos (p.e. libros, pizarras, fotocopias, …) disponibles según el contexto (p.e. profesor del MIT o voluntariado en una selva perdida o …). Lo cual provoca que la formación de un profesor sea compleja y tienda a crecer con nuevos recursos didácticos (p.e. juegos didácticos ágiles, internet, …).*
- *Una alternativa es separar la formación de un profesor genérico con el algoritmo de sus tareas sin ceñirse a unos recursos didácticos particulares y, por otro lado, distintas formaciones especializadas en ciertos recursos didácticos. Se simplifica la responsabilidad de un profesor particular sin afectarle con nuevos recursos que no utilizará.*

- *Define el esqueleto de un algoritmo en una operación difiriendo algunos pasos a las subclases que redefinen esos pasos del algoritmo sin cambiar la estructura*

1. Problema

- Debería usarse:
  - Para implementar las partes de un algoritmo que no cambian y dejar que sean las subclases quienes implementen el comportamiento que puede variar
  - Cuando un comportamiento repetido de varias subclases debería factorizarse y ser localizado en una clase común para evitar código duplicado.
  - Controlar la extensión de subclases definiendo una plantilla de método que llama a operaciones vacías en puntos específicos permitiendo extensiones solo en estos puntos

2. Solución


package structure [templateMethod]
Client
AbstractClass
+templateMethod()
+ primitiveOperation1()
+ primitiveOperation2()
...
this.primitiveOperation1();
...
this.primitiveOperation2();
...
ConcreteClassX
+primitiveOperation1()
+primitiveOperation2()
ConcreteClassY
+primitiveOperation1()
+primitiveOperation2()

2. Solución

- Primero identificar las diferencias en el código existente y entonces separar las diferencias en nuevas operaciones. Finalmente, reemplazar las diferencias de código con una plantilla de método que llama a estas nuevas operaciones
  - Implementar partes invariantes de un algoritmo una sola vez y dejar a las subclases implementar el comportamiento que puede variar
- Las operaciones primitivas a las que llama un método plantilla pueden ser declaradas como miembros protegidos. Esto garantiza que solo puedan ser llamadas por el método plantilla. Las operaciones primitivas que deben ser redefinidas se declaran como virtuales puras. El metodo plantilla en sí no debería ser redefinido; por tanto podemos hacer que sea una función miembro no virtual.

2. Solución

- Un objetivo importante para diseñar métodos plantilla es minimizar el número de operaciones primitivas que una subclase debe redefinir para dar cuerpo al algoritmo. Cuantas más operaciones necesiten ser redefinidas, más tediosas se vuelven las cosas para los clientes.
- Se pueden identificar las operaciones que deberían ser redefinidas añadiendo un sufijo a sus nombre. Por ejemplo "do-"

3. Consecuencias

- Los métodos plantilla llaman a los siguientes tipos de operaciones:
  - Operaciones primitivas, abstractas
  - Operaciones concretas, ya sea de la ClaseConcreta, de las clases cliente o de ClaseAbstracta, que suelen ser útiles para las subclases
  - Operaciones de enganche que proporcionan el comportamiento predeterminado que puede ser modificado por las subclases si es necesario. Una operación de enganche normalmente no hace nada por omisión
    - Es importante que los métodos plantilla especifiquen qué operaciones son de enganche y cuáles son operaciones abstractas.

4. Comparativa

- ***Template Method*** *vs* ***Strategy.***
  - Ver *Strategy vs Template Method*