

METODOLOGÍAS DE PROGRAMACIÓN I

Patrón de comportamiento *Template method*

Situación de ejemplo

- Una empresa organizadora de eventos se dedica a planificar y ejecutar cumpleaños de 15, casamientos, despedidas de año, etc.
- Para cualquier evento esta empresa es la encargada de proveer la vestimenta, enviar las invitaciones, preparar el salón, la música, la comida, las fotos, entrega de suvenires y de la limpieza al finalizar la fiesta.

Situación de ejemplo

Así, por ejemplo cuenta con una clase para los cumpleaños de 15.

```
class CumpleDe15
    void hacerFiesta()
        buscarVestido()
        enviarInvitacionesDeUnicornio()
        prepararSalonParaJovenes()
        ponerMusicaParaJovenes()
        servirPizzasYGaseosa()
        sacarFotos()
        entregarUnaFotoComoRecuerdo()
        limpiarSalon()
```

Situación de ejemplo

Y otra clase para los casamientos.

```
class Casamiento
    void hacerFiesta()
        buscarVestidoYTraje()
        enviarInvitacionesDeAdultos()
        prepararSalonParaAdultos()
        ponerMusicaDeAntes()
        servirLomoALaMostazaConVino()
        sacarFotos()
        entregarUnArbolComoRecuerdo()
        limpiarSalon()
```

Situación de ejemplo

Y otra clase para los casamientos

```
class Casamiento
    void hacerFiesta()
        buscarVestidoYTraje()
        enviarInvitacionesDeAdultos()
        prepararSalonParaAdultos()
        ponerMusicaDeAntes()
        servirLomoALaMostazaConVino()
        sacarFotos()
        entregarUnArbolComoRecuerdo()
        limpiarSalon()
```

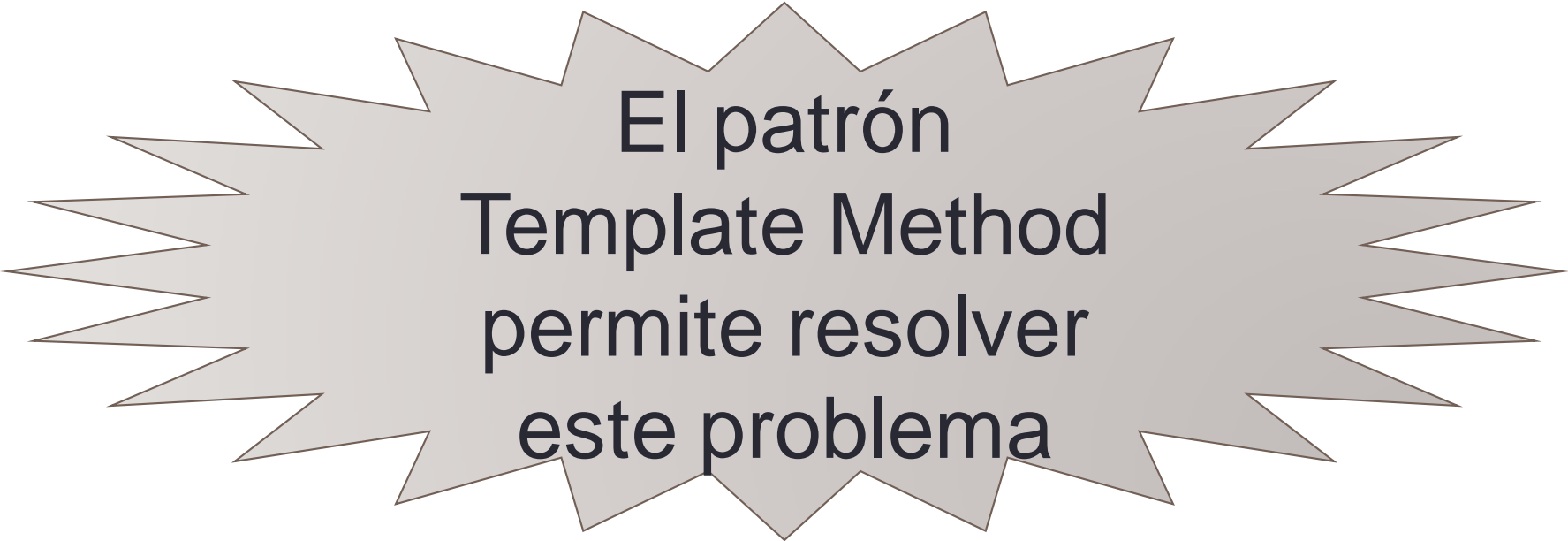
Y de la misma manera, hay una clase por cada tipo de fiesta

Problema

- ¿Qué sucede con las clases mencionadas si la empresa ya no se dedica a vestir a los agasajados, o a la comida o a la limpieza?
- ¿Qué cambios hay que realizar si se sirve la comida dos veces?
- ¿Qué cambiaría si ahora la limpieza la hace solo si los invitados se portaron bien?

Motivación

¿Es posible tener un único método que se pueda modificar, y qué al modificarlo afecte a todas las fiestas?



El patrón
Template Method
permite resolver
este problema

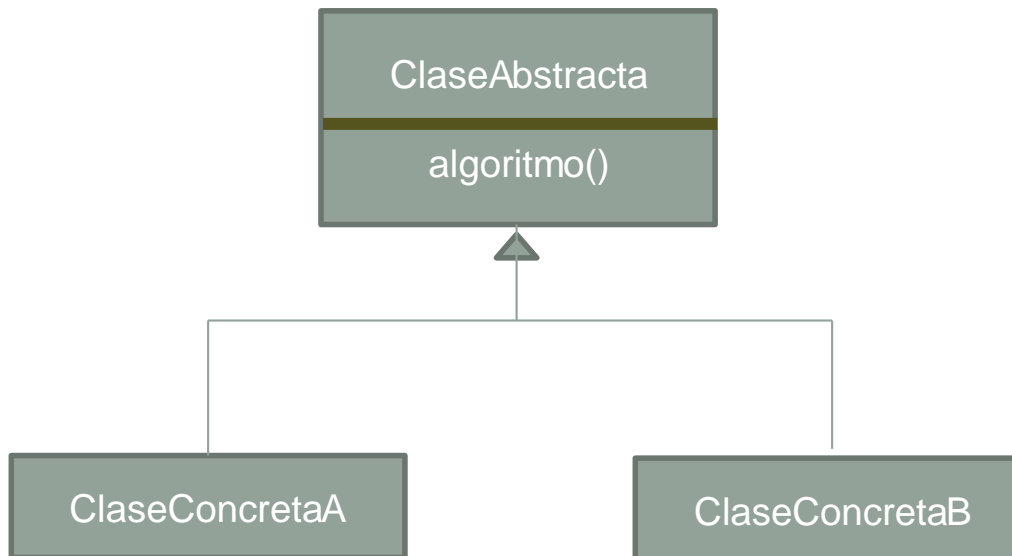
Template method

Propósito: define en una operación el esqueleto de un algoritmo, delegando en sus subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura

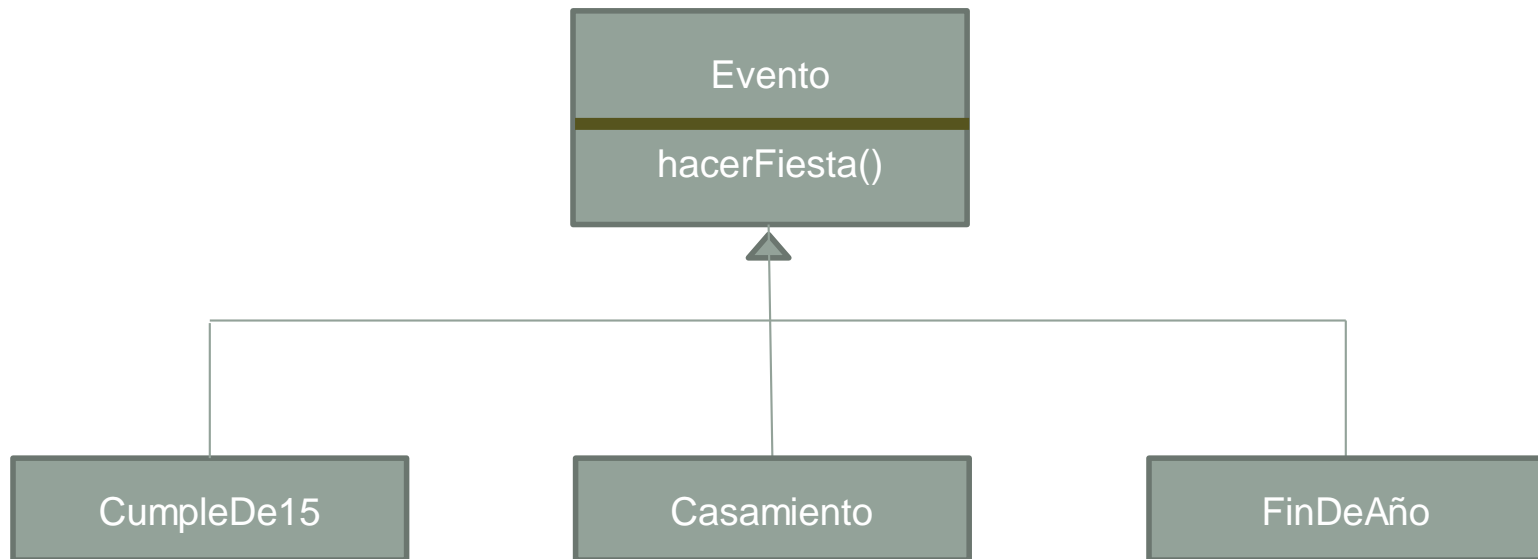
Aplicabilidad: usarlo cuando

- Quiera implementar las partes de un algoritmo que no cambian y dejar que sean las subclases quienes implementan el comportamiento que puede variar.
- El comportamiento repetitivo de varias subclases debería factorizarse y ser localizado en una clase en común para evitar tener código duplicado.
- Quiera controlar las extensiones de las subclases y se necesite definir un método plantilla que llame a operaciones particulares en determinados puntos, permitiendo así las extensiones en esos puntos.

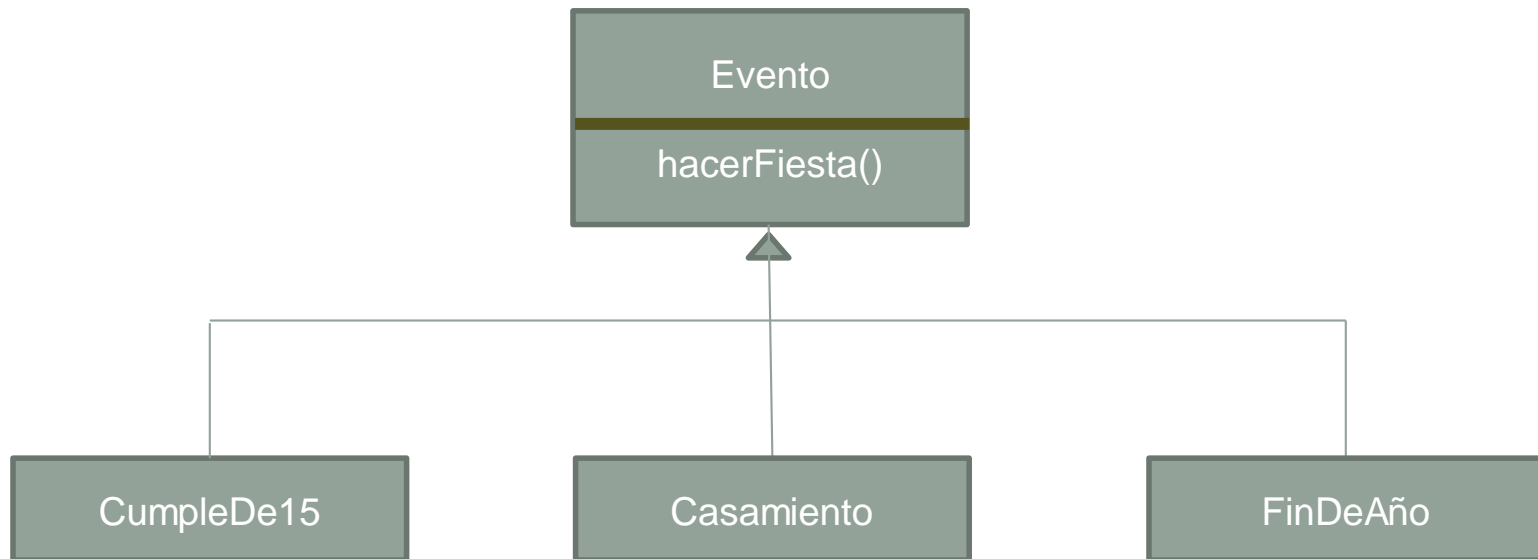
Template method - Estructura



Template method - Estructura



Template method - Estructura



En la superclase se define el esqueleto del algoritmo, las diferentes subclases implementan los detalles

Template method - Implementación

```
abstract class Evento
```

```
    void hacerFiesta()  
        vestirAAgasajado()  
        enviarInvitaciones()  
        prepararSalon()  
        ponerMusica()  
        servirComida()  
        sacarFotos()  
        entregarRecuerdo()  
        limpiarSalon()
```

Template method - Implementación

```
abstract class Evento
```

```
    . . .
```

```
    abstract void vestirAAgasajado()
```

```
    abstract void enviarInvitaciones()
```

```
    abstract void prepararSalon()
```

```
    abstract void ponerMusica()
```

```
    abstract void servirComida()
```

```
    abstract void entregarRecuerdo()
```

Esta clase también tiene
que definir los métodos
"pasos" como abstractos

Template method - Implementación

```
abstract class Evento
```

```
    . . .
```

```
    void sacarFotos()
```

```
        . . .
```

```
    void limpiarSalon()
```

```
        . . .
```

En algunos problemas puede implementar algún comportamiento por defecto que las subclases pueden re-escribir.

Template method - Implementación

```
class CumpleDe15 : Evento

    void vestirAAgasajado()
        print("Buscando un vestido para la
              cumpleaños")

    void enviarInvitaciones()
        print("Enviando invitaciones en tarjetas
              con dibujitos de unicornio")

    void prepararSalon()
        print("Preparando salón para cumple de 15")

    . . .
```

Template method - Implementación

```
class CumpleDe15 : Evento

    void vestirAAgasajado()
        print("Buscando un vestido para la
              cumpleañosera")

    void enviarInvitaciones()
        print("Enviando invitaciones en tarjetas
              con dibujitos de unicornio")

    void prepararSalon()
        print("Preparando salón para cumple de 15")

    . . .
```

Las subclases de *Evento* implementan todos los "pasos" del algoritmo sin importar el orden y la cantidad de veces que son invocados.

Template method - Implementación

```
abstract class Evento
```

```
    void hacerFiesta()  
        vestirAAgasajado()  
        enviarInvitaciones()  
        prepararSalon()  
        while (quedenInvitados())  
            ponerMusica()  
            servirComida()  
            sacarFotos()  
        entregarRecuerdo()  
        if (noHuboProblemas())  
            limpiarSalon()  
        . . .
```

Este algoritmo "esqueleto" permite cambiar fácilmente el orden de ejecución de cada "paso" del algoritmo.

También permite de manera simple invocar a un mismo método más de una vez o usar estructuras de control

Template method - Implementación

```
void metodoCliente()
```

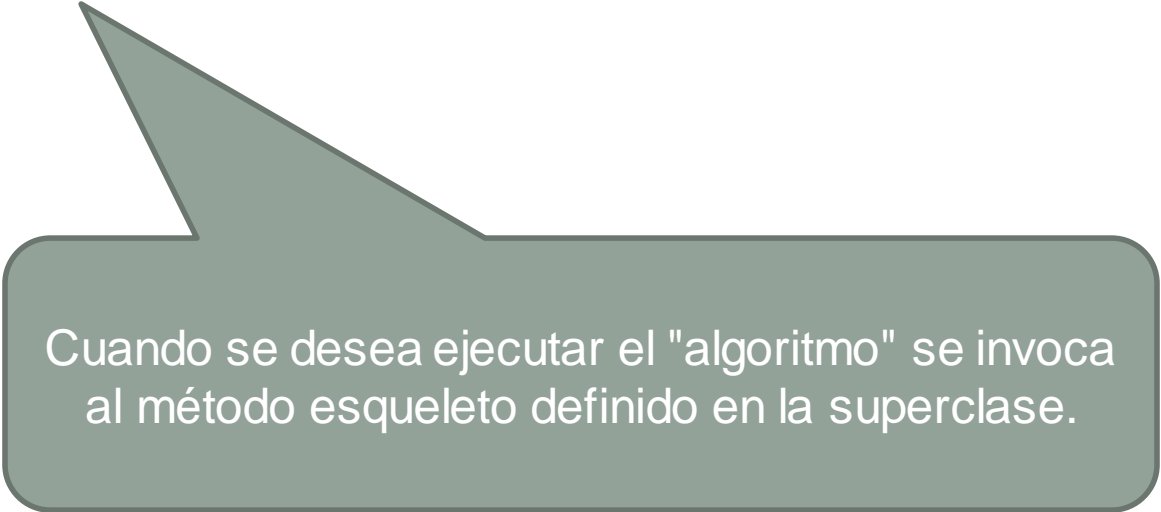
```
    evento = new CumpleDe15()
```

```
    ...
```

```
    evento.hacerFiesta()
```

Template method - Implementación

```
void metodoCliente()  
  
    evento = new CumpleDe15()  
  
    ...  
  
    evento.hacerFiesta()
```



Cuando se desea ejecutar el "algoritmo" se invoca al método esqueleto definido en la superclase.

Template method – Ventajas

- Los métodos "esqueletos" o "plantillas" son una técnica fundamental de reutilización de código.
- Son importantes en la biblioteca de clases ya que son el modo de factorizar y extraer el comportamiento común de las clases de la biblioteca.
- Notar que el patrón Factory Method es un caso particular de este patrón.