

# Patrón Template



**UCAM**

UNIVERSIDAD CATÓLICA  
DE MURCIA

**Realizado por:**

**Jose Juan Giménez Martínez**

**DNI: 77511404R**

**Grado de Ingeniería Informática**

# Contenidos

- ¿Qué es un patrón y para que se utiliza?
- Patrón Template
  - Propósito
  - Motivación
  - Estructura
  - Aplicabilidad
  - Consecuencias

# ¿Qué es un patrón y para que se utiliza?

- Un patrón es una solución probada a un problema de diseño. Para que nuestra solución podamos considerarla un patrón debe tener unas características. Podríamos decir que una de ellas es comprobar su eficacia resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que nos permite aplicarlos a distintos problemas de diseño en circunstancia distintas.

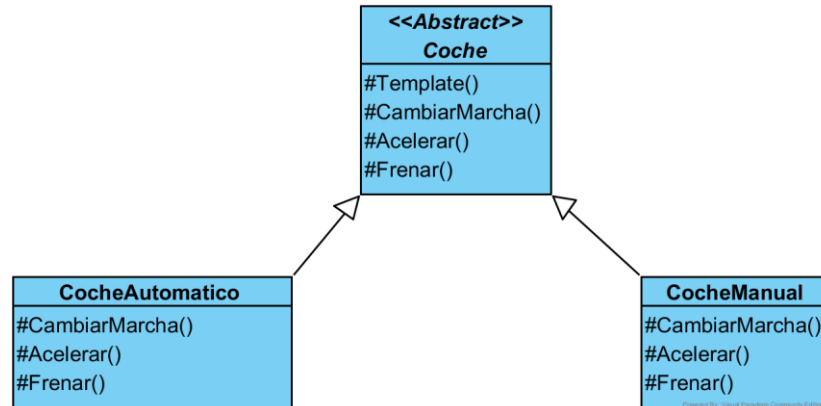
# Propósito

- *“Este método permite que todos los objetos compartan un solo algoritmo que definimos en una SuperClase. Esta superclase tiene un algoritmo compartido por los objetos, pero los objetos que heredan de esta superclase pueden hacer sobrecarga de los métodos heredados sin cambiar la estructura del algoritmo”.*

# Motivación

- *Deseamos implementar las partes de un algoritmo que no cambian y dejar que las subclases implementen aquellas otras que puedan variar. (Clase aplicación que maneja objetos de la clase Documento: método OpenDocument)*
- *Por motivo de factorizar código, cuando movemos cierto código a una clase base común evitar código duplicado.*
- *Para escribir código en un Framework.*

# Estructura



Este método permite que todos los objetos compartan un solo algoritmo que definimos en una SuperClase.

Esta superclase tiene un algoritmo compartido por los objetos, pero los objetos que heredan de esta superclase pueden hacer sobrecarga de los métodos heredados sin cambiar la estructura del algoritmo.

# Aplicabilidad

- Para factorizar el comportamiento común entre varias clases.
- Implementamos las partes fijas en un algoritmo y dejamos que las subclases implementen el comportamiento que puede variar.
- También controlaremos extensiones de las subclases: son algoritmos con puntos de extensión.

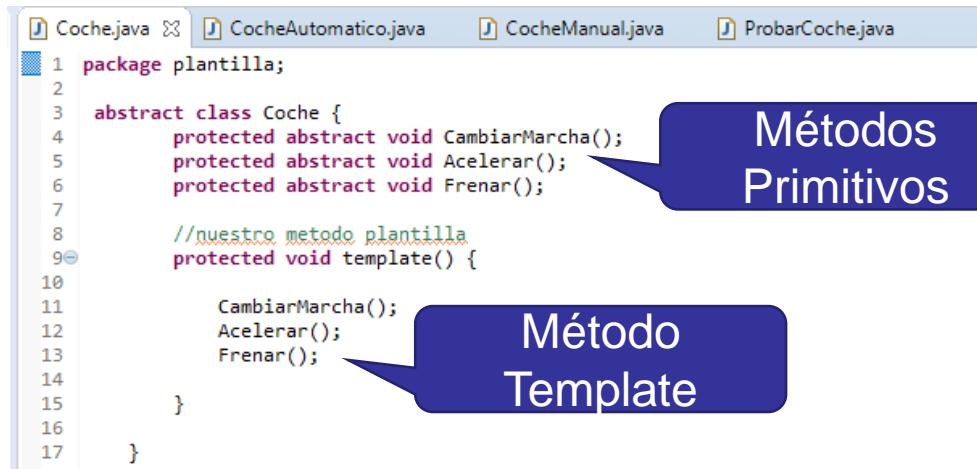
# Consecuencias

- La principal consecuencia que tenemos al utilizar el patrón plantilla es que nos sirve para reutilizar código.
- Inversión de control siendo la clase padre la que llama a los métodos de los hijos.
- Los métodos template pueden llamar a las siguientes tipos de operaciones:
  - Operaciones concretas de las subclases.
  - Operaciones de la propia clase abstracta.
  - Operaciones primitivas es decir abstractas (métodos factorías y operaciones de enganche).



# Código

- Coche.java
- Nuestra primera clase abstracta es Coche, esta nos va a proporcionar los métodos primitivos abstractos que podremos sobrecargar en las subclases hijas CocheAutomatico y CocheManual.



```
1 package plantilla;
2
3 abstract class Coche {
4     protected abstract void CambiarMarcha();
5     protected abstract void Acelerar();
6     protected abstract void Frenar();
7
8     //nuestro metodo plantilla
9     protected void template() {
10
11         CambiarMarcha();
12         Acelerar();
13         Frenar();
14
15     }
16
17 }
```

Métodos Primitivos

Método Template

- La clase Coche también implementa un método “template” desde el que podremos invocar las operaciones primitivas de nuestra clase.
- Este es el método que va a actuar como plantilla, de donde viene el nombre de nuestro patrón, el cual define la secuencia de operaciones de nuestro algoritmo.

# Código

- En las clases hijas CocheAutomatico y CocheManual, vamos a implementar los métodos primitivos heredados de la clase abstracta Coche, redefiniendo de esta forma el comportamiento específico del algoritmo que hemos definido en el método “template”, para las subclases CocheAutomatico y CocheManual.

```
1 package plantilla;
2
3 public class CocheAutomatico extends Coche {
4
5     public CocheAutomatico() {
6         // TODO Auto-generated constructor stub
7     }
8
9     @Override
10    protected void CambiarMarcha() {
11        // TODO Auto-generated method stub
12        System.out.println("Cambio de Marcha Coche Automatico");
13    }
14
15    @Override
16    protected void Acelerar() {
17        System.out.println("Acelero Coche Automatico");
18        // TODO Auto-generated method stub
19    }
20
21    @Override
22    protected void Frenar() {
23        System.out.println("Freno Coche Automatico");
24        // TODO Auto-generated method stub
25    }
26 }
```

```
1 package plantilla;
2
3 public class CocheManual extends Coche {
4
5     public CocheManual() {
6         // TODO Auto-generated constructor stub
7     }
8
9     @Override
10    protected void CambiarMarcha() {
11        System.out.println("Cambio de Marcha Coche Manual");
12        // TODO Auto-generated method stub
13    }
14
15    @Override
16    protected void Acelerar() {
17        // TODO Auto-generated method stub
18        System.out.println("Acelero Coche Manual");
19    }
20
21    @Override
22    protected void Frenar() {
23        // TODO Auto-generated method stub
24        System.out.println("Freno Coche Manual");
25    }
26 }
```

- Debemos tener en cuenta al implementar este patrón unos detalles:
  - Declarar las operaciones primitivas como protected de forma que solo puedan ser llamadas por el método plantilla.
  - Debemos reducir las operaciones primitivas que serán invocadas por “template” al máximo, de esta forma se reduce la complejidad de las subclases y es menos compleja su implementación.

# Código

- Por ultimo tenemos la clase ProbarCoche, con la que vamos a crear dos coches uno automático y otro manual y vamos a llamar al método plantilla que se encargara de escribir los métodos dependiendo desde donde lo llamemos.

```
Coche.java  *CocheAutomatico.java  *CocheManual.java  *ProbarCoche.java  ⌵
1 package plantilla;
2
3 class ProbarCoche {
4
5     public static void main(String[] args) {
6
7         //Creamos un objeto Coche Automatico
8         CocheAutomatico ca = new CocheAutomatico();
9         System.out.println("Metodos de nuestro Coche Automatico");
10        //llamada al metodo template
11        ca.template();
12        System.out.println("\n");
13
14        //Creamos un objeto Coche manual
15        CocheManual cm = new CocheManual();
16        System.out.println("Metodos de nuestro Coche Manual");
17        //llamada al metodo template
18        cm.template();
19    }
20 }
```

```
<terminated> ProbarCoche [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (12 may. 2018 8:42:49)
Metodos de nuestro Coche Automatico
Cambio de Marcha Coche Automatico
Acelero Coche Automatico
Freno Coche Automatico

Metodos de nuestro Coche Manual
Cambio de Marcha Coche Manual
Acelero Coche Manual
Freno Coche Manual
```

# Bibliografía

- Apuntes Tema 3 asignatura.
- Design Patterns Elements of Reusable Object-Oriented Software.(Grady Booch)
- Introducción a los patrones de diseño, un enfoque practico (Oscar Javier Blancarte Iturralde)
- [https://danielggarcia.wordpress.com/2014/05/05/patrones-de-comportamiento-iii-template-method/?utm\\_source=feedburner&utm\\_medium=feed&utm\\_campaign=Feed%3A+danigarcia+%28Let%27s+code+something+up%21%29](https://danielggarcia.wordpress.com/2014/05/05/patrones-de-comportamiento-iii-template-method/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+danigarcia+%28Let%27s+code+something+up%21%29)
- El código de Ejemplo
- <https://es.slideshare.net/An3s/patrn-template>