Patrón Flyweight

Noelia Moreno Cano

77723786-R

nmoreno2@alu.ucam.edu

Patrones de diseño

¿Qué son los patrones de diseño?

Son soluciones para problemas típicos y recurrentes que nos podemos encontrar a la hora de desarrollar una aplicación. Un patrón debe ser efectivo y reutilizable.

¿Por qué son útiles los patrones de diseño?

- Te ahorran tiempo → te permiten solucionar algunos problemas de forma directa.
- Te aseguran la validez de tu código → los patrones de diseño son estructuras probadas por millones de desarrolladores a lo largo de muchos años, por lo que si elegimos el patrón válido podemos asegurar que será una solución correcta.
- Establecen un lenguaje común → la resolución de un problema de un código modelado con un patrón de diseño será fácil de explicar a otras personas, conozcan el código o no.

Patrón Flyweight

El patrón **Flyweight** (también llamado “objeto ligero”) se utiliza para eliminar o intentar reducir lo máximo posible la redundancia cuando tenemos objetos con información similar.

Se trata de un **patrón estructural**. Los patrones estructurales nos facilitan la modelización de nuestro software especificando la forma en la que unas clases se relacionan con otras.

Aplicaciones

- Cuando se necesita eliminar o reducir la redundancia cuando se tiene gran cantidad de objetos que comparten bastante información.
- El soporte tiene memoria limitada y se necesita que sea aprovechada óptimamente.
- La identidad propia de los objetos es irrelevante.

Ventajas y desventajas

+ Produce ahorro de la capacidad de almacenamiento.

+ Reduce el número total de objetos.

+Reduce en gran cantidad el peso de los datos en un servidor.

-Consume un poco más de tiempo para realizar las búsquedas.

-Aumenta la complejidad de los objetos

-Aumenta el número de clases del sistema

Tipos de datos

En un patrón **Flyweight** distinguiremos entre dos tipos de datos:

- **Intrínsecos**: son los datos compartidos por todos los objetos de la clase. Normalmente estos objetos no cambian a lo largo del tiempo, y si cambian también modificarán el estado de todos los objetos que hagan uso de ellos.
- **Extrínsecos**: es la información que se puede quitar de una clase y se almacena externamente. La idea de los datos extrínsecos es que ocupen una cantidad de memoria mínima.

Participantes

Cliente: Trabaja con una referencia a un Flyweight. Establece los estados extrínsecos de los objetos.

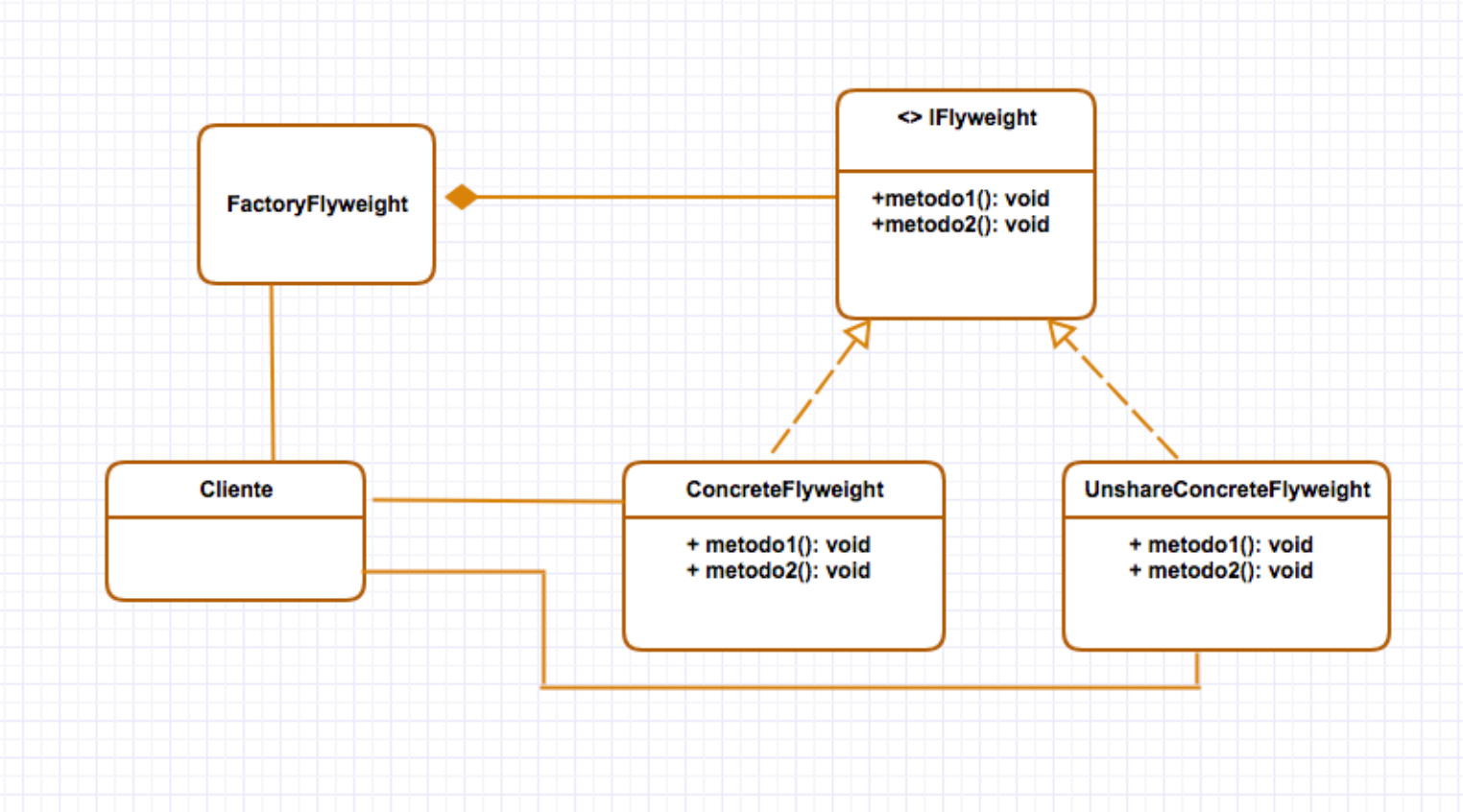
IFlyweight: Declara una interfaz a través de la cual los flyweight pueden recibir y actuar sobre los estados no compartidos.

ConcreteFlyweight: Implementa la interfaz Flyweight y almacena los estados compartidos, si los hay. Un objeto ConcreteFlyweight debe ser compartible. Cualquier estado que almacene debe ser intrínseco; es decir, debe ser independiente de su contexto.

UnshareConcreteFlyweight: No todas las subclases de flyweight tiene que ser compartidas. La interfaz IFlyweight permite que se comparta, pero no es obligatorio. Es común que los objetos de esta clase tengan hijos de la clase ConcreteFlyweight en algún nivel de su estructura.

FactoryFlyweight: Crea y gestiona los objetos flyweight. Garantiza que los objetos flyweight se comparten de forma apropiada. Cuando un cliente solicita un flyweight, el objeto de la clase FactoryFlyweight proporciona una instancia existente, o crea una.

Diagrama UML

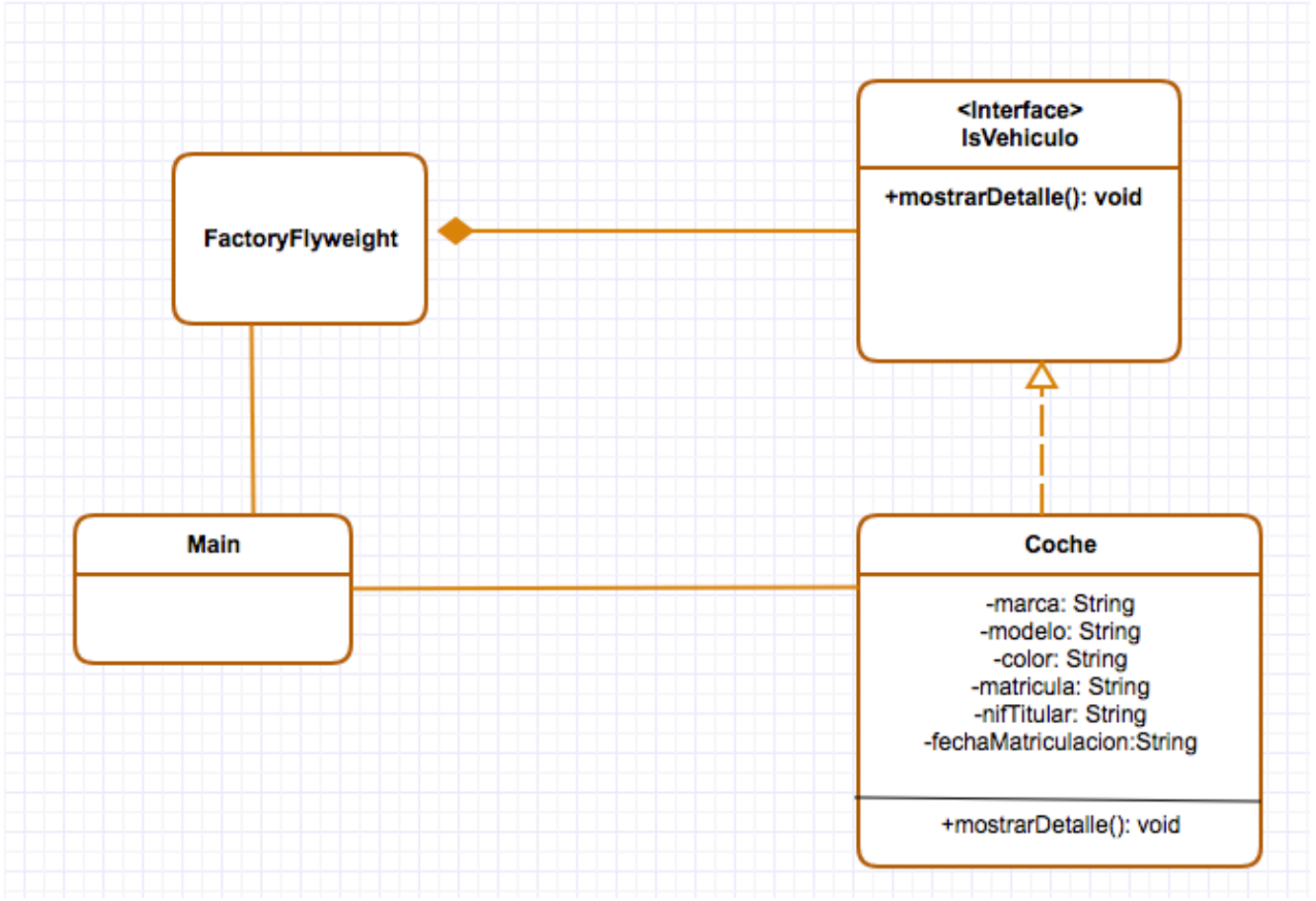


Ejemplo

- La DGT nos pide una aplicación que se de encargue de gestionar todos los vehículos de España y necesitamos almacenar toda la flota de vehículos. Debemos almacenar datos como la marca, modelo, color, matrícula, fecha de matriculación y NIF titular.

Ejemplo

- Aplicando al ejemplo:



Ejemplo

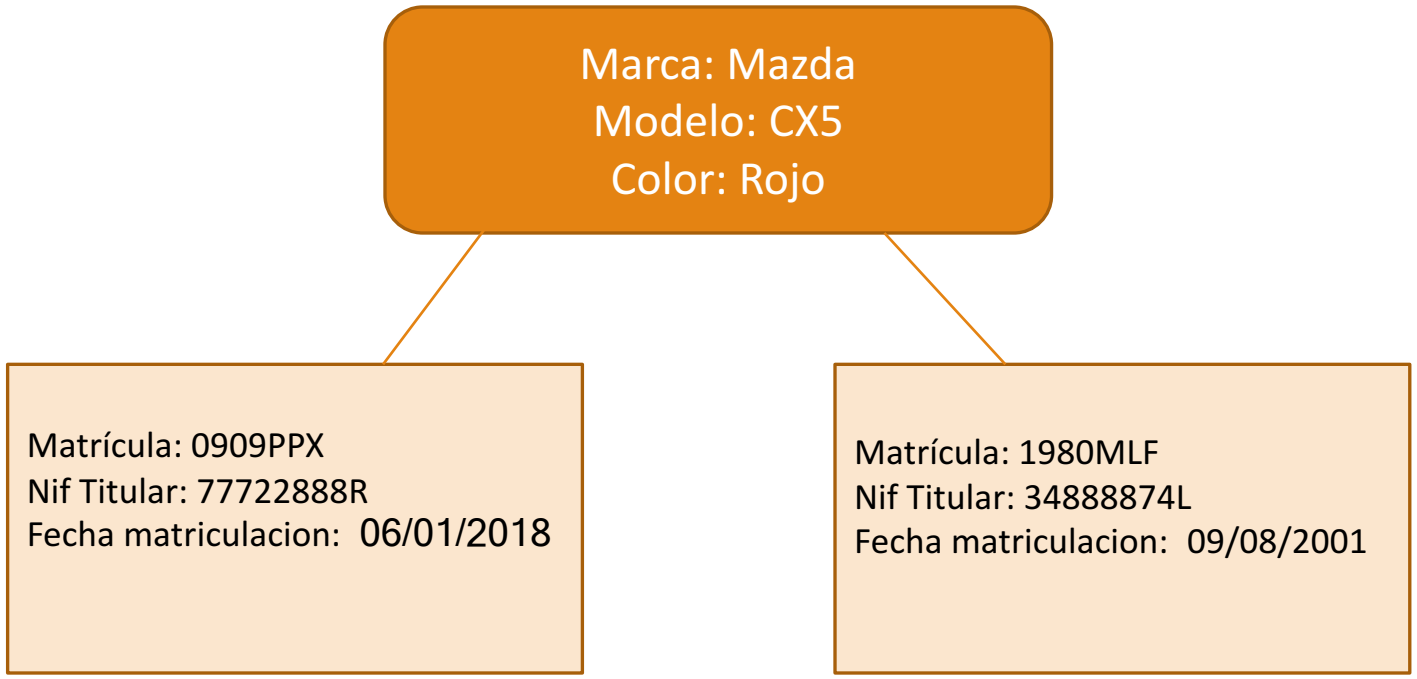
- Sin aplicar el patrón Flyweight:

Marca: Mazda
Modelo: CX5
Color: Rojo
Matrícula: 0909PPX
Nif Titular: 77722888R
Fecha matriculacion: 06/01/2018

Marca: Mazda
Modelo: CX5
Color: Rojo
Matrícula: 1980MLF
Nif Titular: 34888874L
Fecha matriculacion: 09/08/2001

Ejemplo

- Aplicando el patrón Flyweight:



Clase Coche

```java
package PatronFlyweight;

import java.sql.Date;

public class Coche implements IsVehiculo {
    public String marca;
    public String modelo;
    public String color;
    public String matricula;
    public String nifTitular;
    public String fechaMatriculacion;

    //Constructor Coche, para crear un nuevo coche
    public Coche (String marca, String modelo, String color, String matricula, String nifTitular, String fechaMatriculacion)
    {
        this.marca= marca;
        this.modelo= modelo;
        this.color= color;
        this.matricula= matricula;
        this.nifTitular= nifTitular;
        this.fechaMatriculacion= fechaMatriculacion;
    }

    @Override
    public String mostrarDetalle() {
        // Sobrescribimos el mÈtodo para mostrar los detalles que nos interesan.
        return "\n   Marca: " + this.marca + "\n   Modelo: " + this.modelo + "\n   Color: " + this.color + "\n Matricula: " +this.matricula + "\n NifTitular: "
        + this.nifTitular + "\n Fecha de Matricula: "+ this.fechaMatriculacion +"\n";
    }

}
```

Clase FactoryFlyweight

```
1  package PatronFlyweight;
2
3  import java.sql.Date;
4
5  public class FactoryFlyweight {
6      IVehiculo coche;
7
8      public void listarCoches(String marca[], String modelo[], String color[], String [] matricula, String [] nifTitular, String [] fechaMatric
9      {
0          for (int i=0; i < matricula.length; i++)
1          {
2              coche = new Coche(marca[i], modelo[i], color[i], matricula[i], nifTitular[i], fechaMatriculacion[i]);
3              System.out.println("Datos del coche:\n" + coche.mostrarDetalle());
4          }
5      }
6  }
7
```

Clase IVehiculo

```
1  package PatronFlyweight;
2
3  //Interfaz IsVehiculo.
4  public interface IVehiculo {
5
6      public String mostrarDetalle();
7
8  }
9
10
```

Clase Main

```java
package PatronFlyweight;

import java.sql.Date;

public class main {
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        //Creamos un nuevo FactoryFlyweight.
        FactoryFlyweight fflyweight = new FactoryFlyweight();

        //Listado de marcas
        String marca[] = {"Mazda", "Mazda", "Ford", "Seat", "Mercedes"};

        //Listado de modelos
        String modelo[] = {"CX5", "CX5", "Focus", "Ateca", "Clase A"};

        //Listado de colores
        String color[] = {"Rojo", "Rojo", "Negro", "Blanco", "Gris"};

        //Listado de matriculas
        String matricula[] = {"0909PPX", "1980MLF", "6765BAD", "1188NXS", "9234XCS"};

        //Listado de colores
        String nifTitular[] = {"77722888R", "34888874L", "98989274R", "77723788M", "73342119M"};

        //Listado de fechas de matricula
        String fechaMatriculacion[] = {"06/01/2018", "09/08/2001", "03/08/1990", "19/09/2016", "06/06/2018"};

        //Listamos.
        fflyweight.listarCoches(marca, modelo, color, matricula, nifTitular, fechaMatriculacion);


    }
}
```

Resultado código

```
Datos del coche:

   Marca: Mazda
   Modelo: CX5
   Color: Rojo
 Matricula: 0909PPX
 NifTitular: 77722888R
 Fecha de Matricula: 06/01/2018

Datos del coche:

   Marca: Mazda
   Modelo: CX5
   Color: Rojo
 Matricula: 1980MLF
 NifTitular: 34888874L
 Fecha de Matricula: 09/08/2001

Datos del coche:

   Marca: Ford
   Modelo: Focus
   Color: Negro
 Matricula: 6765BAD
 NifTitular: 98989274R
 Fecha de Matricula: 03/08/1990
```

```
Datos del coche:

   Marca: Seat
   Modelo: Ateca
   Color: Blanco
 Matricula: 1188NXS
 NifTitular: 77723788M
 Fecha de Matricula: 19/09/2016

Datos del coche:

   Marca: Mercedes
   Modelo: Clase A
   Color: Gris
 Matricula: 9234XCS
 NifTitular: 73342119M
 Fecha de Matricula: 06/06/2018
```