Flyweight

Nombre:
Flyweight

Clasificación del patrón:
Estructural

Intención:
Uso compartido soporta un largo número de objetos de peso fino eficientemente.

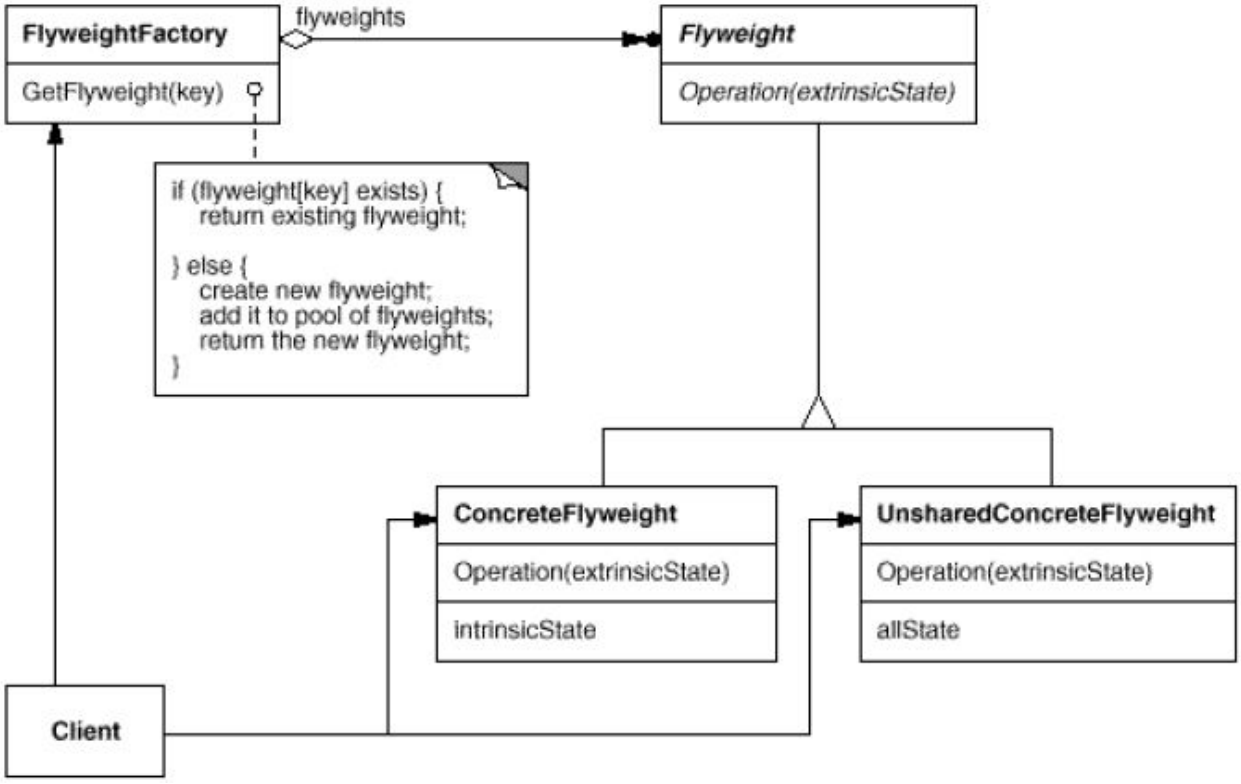
Otros nombres:
Peso mosca, Peso Ligero.

Motivación:
Se utilizan objetos que almacenan los estados compartidos y que pueden ser utilizados por varios objetos de forma simultánea, con éste describe cómo almacenar un gran número de objetos sin gran costo.

Aplicabilidad:
Se usa cuando:

- Una aplicación usa un gran número de objetos.
- El costo de almacenamiento es alto por una larga cantidad de objetos.
- El estado del objeto puede ser hecho extrínseco.
- La aplicación no depende de la identidad de los objetos.
- Muchos objetos pueden ser representados por relativamente pocos objetos compartidos una vez el estado extrínseco es removido.

Estructura:



Participantes:

- PesoLigeroConcreto: Implementa la interfaz PesoLigero.
- PesoLigero: Define una interfaz a través de la cual los PesosLigeros pueden recibir y actuar sobre estados no compartidos.
- PesoLigeroConcretoNoCompartido: No todas las subclases de PesoLigero son compartidas.
- Cliente: Contiene referencias de los PesosLigeros.
- FactoriaPesoLigero: Crea y gestiona los PesoLigero, garantiza que se compartan adecuadamente.

Colaboraciones:

Los clientes invocan a la FactoriaPesoLigero. El estado se mantiene por el cliente y se pasa cuando se invocan los métodos que lo solicitan.

Ventajas:

- Las ventajas dependerán de la habilidad de compartir intrínsicamente estados entre objetos.
- Si el estado instrínseco es largo, resultaría menos memoria de uso.

- Si el número de PesoLigeros es grande mayor será el almacenamiento ahorrado.

Desventajas:

- El costo del tiempo de ejecución dependerá del tiempo del cálculo extrínseco y luego el de transferirlo a los objetos.

Implementación:

- Asegurar que el rendimiento es un tema primordial y el cliente está dispuesto a asumir los reajustes.
- Dividir el objetivo principal en estados: Intrínseco (Elementos compartidos o comunes) y Extrínsecos (Elementos particulares de cada tipo).
- Retirar elementos con estado extrínseco de los atributos y añadir llamadas a métodos.
- Crear una fábrica que pueda almacenar y reutilizar instancias existentes de clases.
- Se debe usar la fábrica en vez de utilizar “new” o cualquier otra palabra reservada para la creación de objetos.

Código de ejemplo:

- Main:

```
1  package estructurales.flyweight.flyweight01;
2  public class Main
3  {
4      public static void main(String[] args)
5      {
6          FabricaDeLineas fabrica = new FabricaDeLineas();
7          ILineaLigera linea1 = fabrica.getLine( "AZUL" );
8          ILineaLigera linea2 = fabrica.getLine( "ROJO" );
9          ILineaLigera linea3 = fabrica.getLine( "AMARILLO" );
10         ILineaLigera linea4 = fabrica.getLine( "AZUL" );
11         System.out.println("--------------------");
12         //can use the lines independently
13         linea1.dibujar( 100, 400 ) ;
14         linea2.dibujar( 200, 500 );
15         linea3.dibujar( 300, 600 );
16         linea4.dibujar( 400, 700 );
17     }
18 }
```

-

- FabricaDeLineas:

```
1  package estructurales.flyweight.flyweight01;
2  import java.util.ArrayList;
3  import java.util.List;
4  public class FabricaDeLineas
5  {
6      private List<ILineaLigera> lineas;
7      // --------------------
8      public FabricaDeLineas()
9      {
10         this.lineas = new ArrayList<ILineaLigera>();
11     }
12     // --------------------
13     public ILineaLigera getLine( String color )
14     {
15         // Comprobar si hemos creado una línea con el color solicitado, y devolverla en tal caso
16         for(ILineaLigera linea : this.lineas)
17         {
18             if( linea.getColor().equals(color) )
19             {
20                 System.out.println("Línea de color [" + color + "] encontrada, la devolvemos");
21                 return linea;
22             }
23         }
24         // Si no ha sido creada la creamos ahora, la agregamos a la lista y la devolvemos
25         System.out.println("Creando una línea de color [" + color + "]");
26         ILineaLigera linea = new Linea( color );
27         this.lineas.add( linea );
28         return linea;
29     }
30 }
31
```

- ILineaLigera:

```
1  package estructurales.flyweight.flyweight01;
2  public interface ILineaLigera
3  {
4      public String getColor();
5      public void dibujar( int col, int fila );
6  }
```

- Línea:

```
1  package estructurales.flyweight.flyweight01;
2  public class Linea implements ILineaLigera
3  {
4      private String color;
5      // --------------------
6      public Linea( String color )
7      {
8          this.color = color;
9      }
10     // --------------------
11     @Override
12     public String getColor()
13     {
14         return this.color;
15     }
16     // --------------------
17     @Override
18     public void dibujar( int col, int fila )
19     {
20         System.out.println( "Dibujando línea de color [" + this.color + "] en [" + col + ", " + fila + "]" );
21     }
22 }
```

Usos conocidos:

- Interfaces de usuario de aplicaciones.

Bibliografía:

No específico. (No específico). GoF Design Patterns (Versión 2.1.0) [Aplicación móvil]. Descargado de: https://drive.google.com/file/d/0BywiVyFlIabXcVhGZlJBcnhWTkU/view.

Patrones de Diseño Software [Página Web]. (s.f.). Ubicación: https://informaticapc.com/patrones-de-diseno/flyweight.php.

Junta de Andalucía. (s.f). Peso Ligero. Marco de Desarrollo de la Junta de Andalucía. http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/197.