Flyweight

Objective

Eliminate or reduce redundancy when there are a large number of objects containing identical information, while achieving a balance between flexibility and performance.

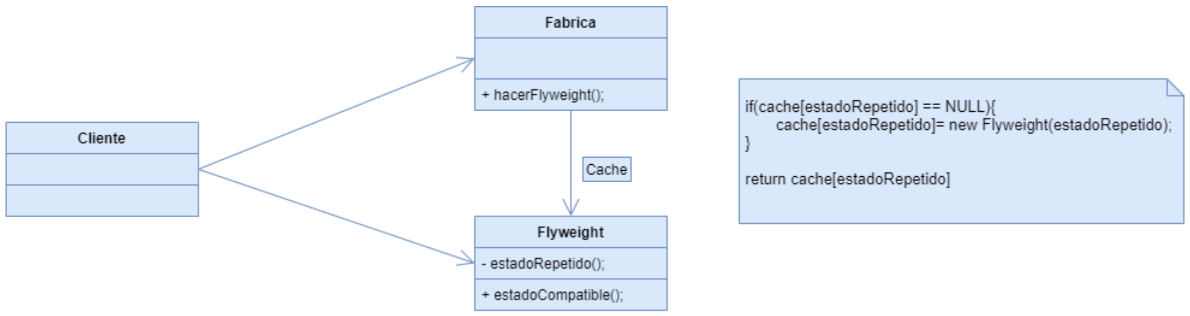
Function

Reduce redundancy in situations where large quantities of objects that have identical information.

Structure

The Flyweight pattern is stored in a factory repository. The client refrains from creating Flyweights directly and requests them from Factory. Flyweight can't stand on its own. Any attribute that makes impossible to share must be provided by the customer as long as it is done a request to the Flyweight.

The structure that meets this pattern is shown in Figure 1



```
if(cache[estadoRepetido] == NULL){
      cache[estadoRepetido]= new Flyweight(estadoRepetido);
}

return cache[estadoRepetido]
```

Figure 1: UML Diagram Flyweight Pattern

Applications

The use of the Flyweight pattern is recommended when:

- The system does not depend on the identity of the object.
- The system requires the use of large numbers of objects; this produces high storage costs.
- Most groups of objects can be replaced by a few objects shared.

Design Patterns Collaborators

- The Flyweight pattern works in conjunction with Composite, to represent a hierarchical structure by means of a node graphic.

- Both the State and Strategy standards work ideally with a view to Flyweight.

Scope of action

Applied at the object level.

Problem

The application requires instantiating the same objects, with different functionality, which is highly costly in terms of memory.

Solution

The Flyweight pattern allows the sharing of objects in such a way as to employ fine granularity without the high costs of implementation; to which each object is divided into two parts: state-independent or intrinsic, stored in the Flyweight object; and state-dependent or extrinsic, stored in the client objects that are responsible for passing on their operations to Flyweight when they are invoked.

Diagram or Implementation

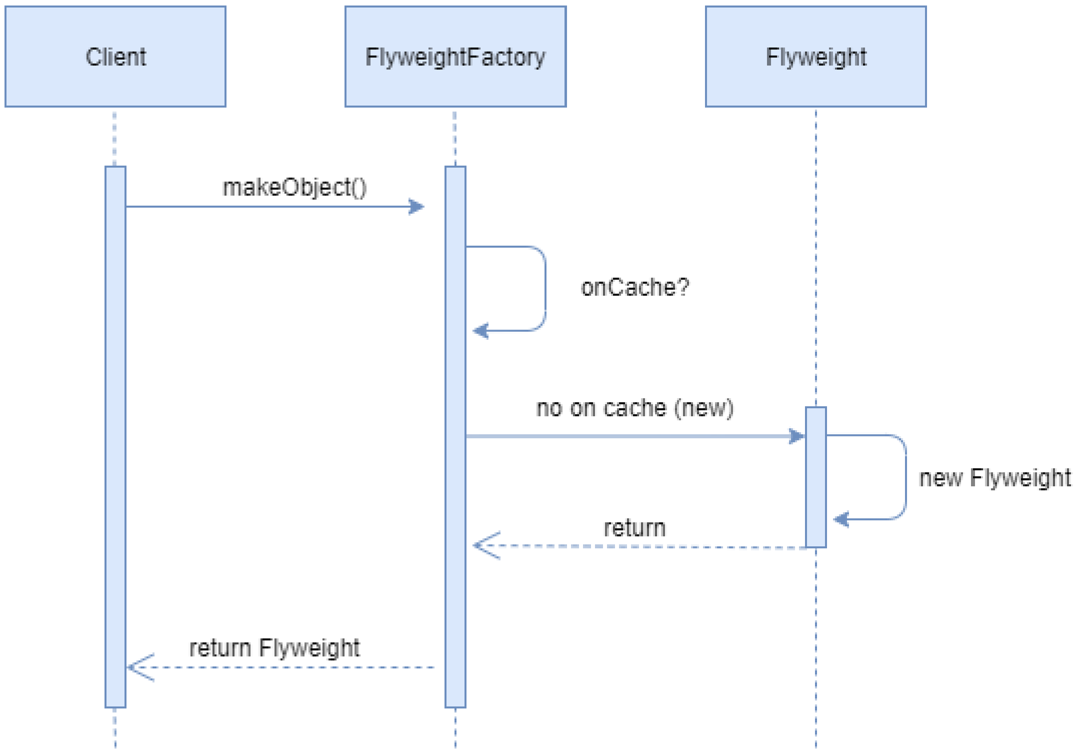


Figure 2: UML Diagram Flyweight Pattern

Figure 2 explains the behaviour of the Flyweight pattern by means of a sequence diagram.

- The client class asks the Factory component to create a Flyweight object.

- The Factory class before creating the object, validates if an identical object already exists who is being solicited. If so, it returns the existing object; if not, it returns the existing object, creates the new object and caches it for later use.

- The Flyweight object is created or taken from the cache and returned to the client.