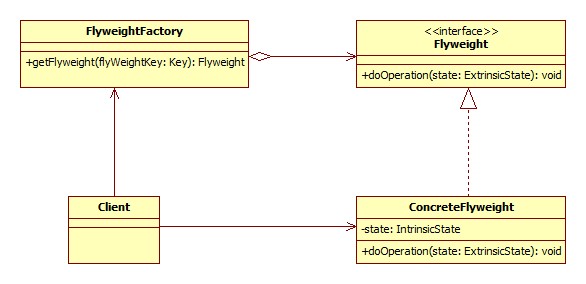
# **Flyweight Pattern**

## **Description**

Flyweight is one of the structural design patterns. The intent of this pattern is to use sharing to support a large number of objects that have part of their internal state in common where the other part of state can vary. Flyweight was invented in order to solve the problems that some programs require a large number of objects that have shared state among them and they often consume lots of memory and may incur unacceptable runtime overhead.

## **Implementation**



**Flyweight** – Declares an interface through which flyweights can receive and act on extrinsic state

**ConcreteFlyweight** – Implements the Flyweight interface and stores intrinsic state. A ConcreteFlyweight object must be sharable. The Concrete flyweight object must maintain state that it is intrinsic to it, and must be able to manipulate state that is extrinsic.

**FlyweightFactory** – The factory creates and manages flyweight objects. In addition the factory ensures sharing of the flyweight objects. The factory maintains a pool of different flyweight objects and returns an object from the pool if it is already created, adds one to the pool and returns it in case it is new.

**Client** – A client maintains references to flyweights in addition to computing and maintaining extrinsic state

A client needs a flyweight object; it calls the factory to get the flyweight object. The factory checks a pool of flyweights to determine if a flyweight object of the requested type is in the pool. If there is, it returns that object. If there is no object of the required type, the factory creates a flyweight of the requested type, adds it to the pool, and return it. The flyweight maintains intrinsic state which is shared among the large number of objects that we have created the flyweight for. It also provides methods to manipulate external state which vary from object and is not common among objects that we have created the flyweight for.

## **Advantages and Disadvantages**

* Advantages
  + Reuse objects effectively
  + Memory saving
* Disadvantages
  + Introduce runtime costs with
    - Transferring
    - Finding
    - Computing extrinsic state

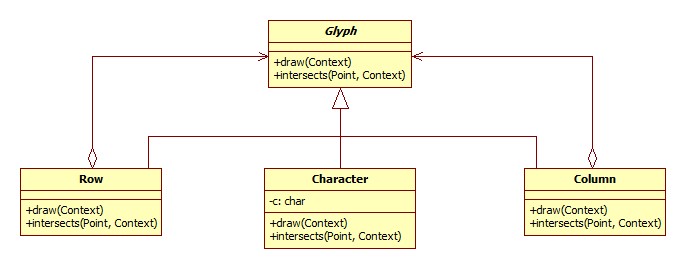
**Usage:**

* The more flyweights are shared, the greater the storage savings.
* The saving increase with the amount of share state.
* The greatest savings occur when the objects use substantial quantities of both intrinsic state, and the extrinsic state can be computed rather than stored.

## **Example**: Text formatting and editing

Object-oriented document editors typically use objects to represent embedded elements like tables and figures. However, they usually stop short of using an object for each character in the document in order to increase flexibility. The characters could then be formatted and drawn. The application could be extended to support new character sets without disturbing other functionalities.

If we just normally create object for each character, it will consume lots of memory and huge overhead. A moderate sized documents may require hundreds of thousands of character objects. On the other hands, using Flyweight pattern will save lots of memory. A flyweight is created for each letter of the alphabet. Each flyweight stores a character code, but its coordinate position in the document and its style can be determine from the text layout algorithms and formatting commands. We can say that the character code is intrinsic state, while other information is extrinsic state. Basically, there is one shared flyweight per character, and it appear in different contexts in the document structure. Each occurrence of particular character object refers to the same instance pool of flyweight objects.



## **Related pattern**

**Factory and Singleton patterns**: Flyweights are usually created using a factory and the singleton is applied to that factory so that for each type or category of flyweights, a singleton instance is returned.

**State and Strategy patterns**: State and Strategy objects are usually implemented as Flyweights because State and strategy increase number of objects if an application. We can reduce this overhead by implementing them as stateless objects that contexts share.

**Composite pattern**: The flyweight pattern is often combined with the Composite pattern to implement a logically hierarchical structure in terms of a directed-acyclic graph with shared leaf nodes.

## **Reference**

OODesign 2012, ‘Flyweight Pattern’, OODesign.com, viewed 20 Nov 2012, <<http://www.oodesign.com/flyweight-pattern.html>>

Gamma, E, Helm, R, Johnson, R, Vlissides, J 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st edn, M.C. Escher, Holland.