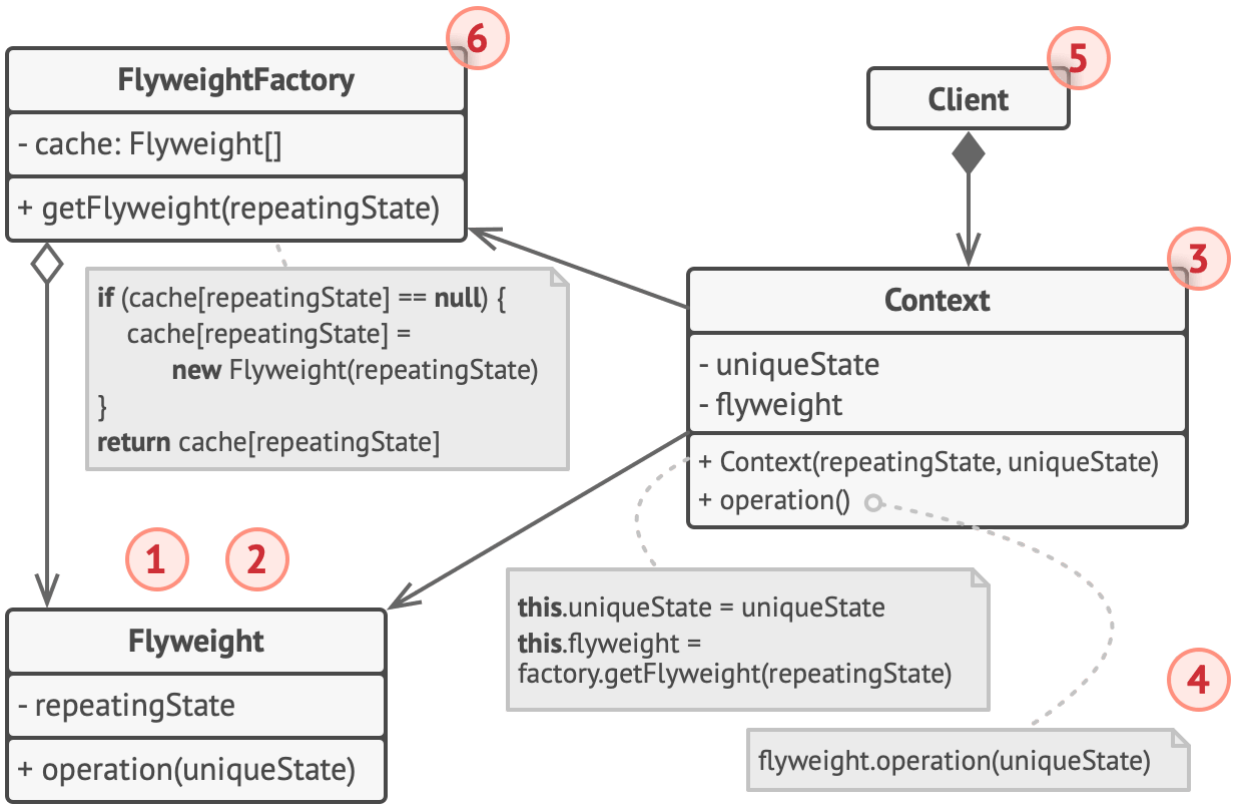
Structure



1. The Flyweight pattern is merely an optimization. Before applying it, make sure your program does have the RAM consumption problem related to having a massive number of similar objects in memory at the same time. Make sure that this problem can't be solved in any other meaningful way.

2. The **Flyweight** class contains the portion of the original object's state that can be shared between multiple objects. The same flyweight object can be used in many different contexts. The state stored inside a flyweight is called *intrinsic*. The state passed to the flyweight's methods is called *extrinsic*.

3. The **Context** class contains the extrinsic state, unique across all original objects. When a context is paired with one of the flyweight objects, it represents the full state of the original object.

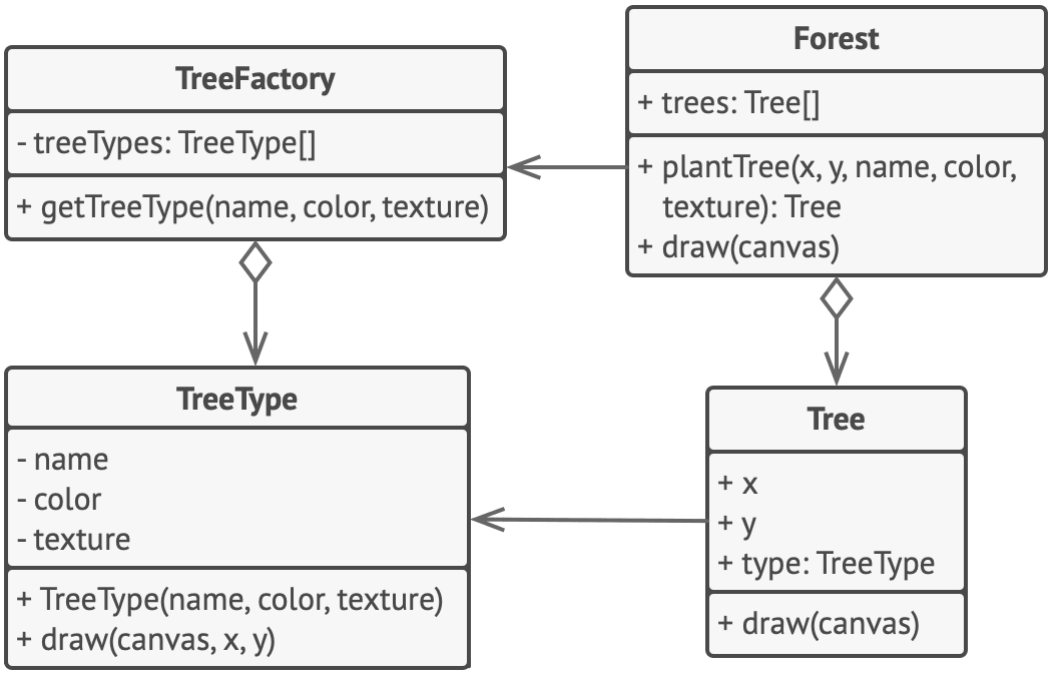
4. Usually, the behavior of the original object remains in the flyweight class. In this case, whoever calls a flyweight's method must also pass appropriate bits of the extrinsic state into the method's parameters. On the other hand, the behavior can be moved to the context class, which would use the linked flyweight merely as a data object.

5. The **Client** calculates or stores the extrinsic state of flyweights. From the client's perspective, a flyweight is a template object which can be configured at runtime by passing some contextual data into parameters of its methods.

6. The **Flyweight Factory** manages a pool of existing flyweights. With the factory, clients don't create flyweights directly. Instead, they call the factory, passing it bits of the intrinsic state of the desired flyweight. The factory looks over previously created flyweights and either returns an existing one that matches search criteria or creates a new one if nothing is found.

Pseudocode

In this example, the **Flyweight** pattern helps to reduce memory usage when rendering millions of tree objects on a canvas.



The pattern extracts the repeating intrinsic state from a main `Tree` class and moves it into the flyweight class `TreeType`.

Now instead of storing the same data in multiple objects, it's kept in just a few flyweight objects and linked to appropriate `Tree` objects which act as contexts. The client code creates new tree objects using the flyweight factory, which encapsulates the complexity of searching for the right object and reusing it if needed.