Flyweight Pattern

Structural Design Pattern

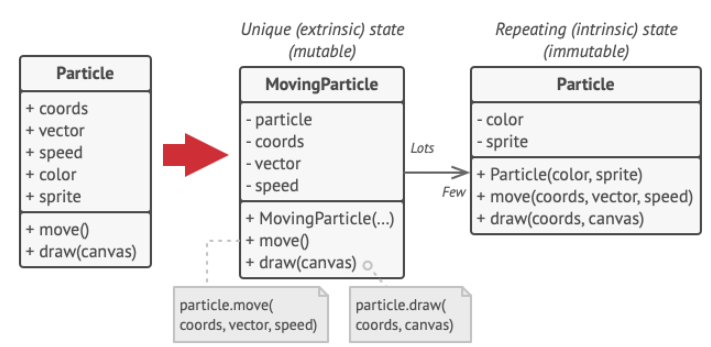
**What is Flyweight:**

1. Our system needs a large number of objects of a particular class & maintaining these instances is a performance concern.
2. Flyweight allows us to share an object in multiple contexts. But instead of sharing an entire object, which may not be feasible, we divide the object state in two parts.
3. Intrinsic state: state that is shared in every context
4. Extrinsic stateL context specific state.
5. We create objects with only intrinsic state and share them in multiple contexts.
6. Client or user of the object provides the extrinsic state to the object to carry out its functionality.

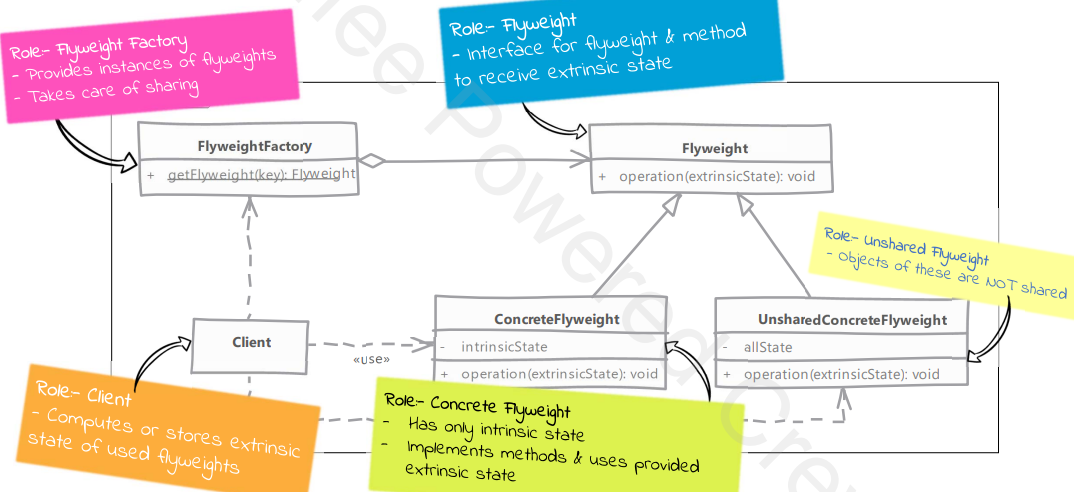
**Intent:** lets you fit more objects into the available amount of RAM by sharing common parts of state (parameters) between multiple objects instead of keeping all of the data in each object.

**Problem:** In a video game we have a particle system(bullets, missiles, shrapnel, etc.). Each particle is represented by a separate object containing plenty of data. At some point, when there are many particles in the game, newly creating particles no longer fit into the remaining RAM, so the program crashes.

**Solution:** On closer inspection of the Particle class, you may notice that the color and sprite fields consome a lot more memory than other fields. These two fields also store almost identical data across all particles. Other parts of the particle’s state such as coordinates, movement vectors are unique to every particle. The flyweight pattern suggests that you stop storing extrinsic state inside the object. Instead, you should pass this state to specific methods which rely on it. Only the intrinsic state stays within the object.



**UML:**



**Implementation Steps:**

1. Identify intrinsic & extrinsic state of our object.
   1. Create an interface for flyweight to provide common methods that accept extrinsic state.
   2. In implementation of shared flyweight we add intrinsic state & also implement methods.
   3. In unshared flyweight implementation we simply ignore the extrinsic state argument as we have all states within the object.
2. Implement the flyweight factory which caches flyweights & also provides methods to get them.
3. In our client we either maintain the extrinsic state or compute it on the fly when using flyweight.

**Comparison with Object Pool:**

1. Flyweight: state is divided. Client must provide part of the state to it.
2. Flyweight: client will not change the intrinsic state of flyweight instance as it is shared.
3. Object pool: a pooled object contains all of its state encapsulated within itself.
4. Object pool: can and will change the state of pooled objects.