SDET Course

Design Patterns - Flyweight

3 Types of Design Patterns

- Creational
  - Singleton
  - Builder
  - Prototype
  - Factory Method
  - Abstract Factory
- Structural
  - Adapter
  - Composite
  - Proxy
  - **Flyweight**
  - Bridge
  - Facade
  - Decorator
- Behavioral
  - Strategy
  - Observer
  - Command
  - Memento
  - State
  - Template Method
  - Mediator
  - Chain of Responsibility
  - Interpreter
  - Visitor
  - Iterator

Agenda

- Description
- Diagram
- Code sample (Java)
- Use cases

Descrip

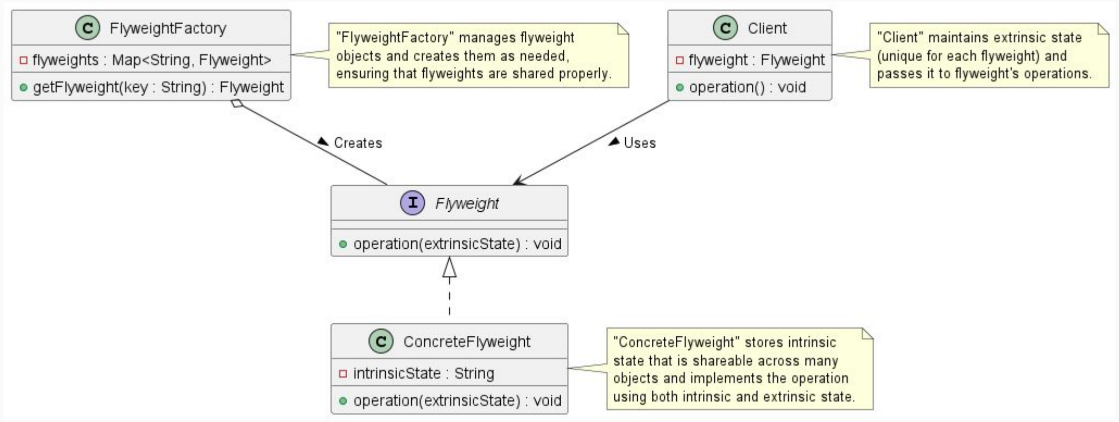
on

Introduction - A Library

Description

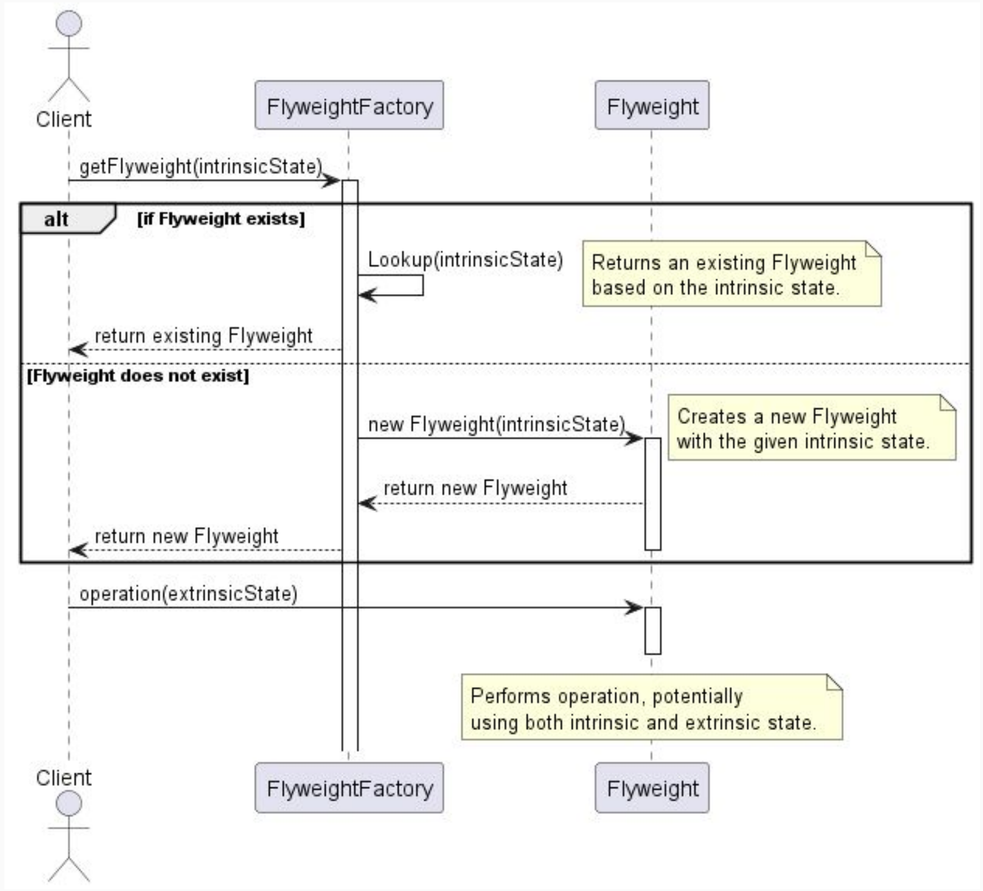
The Flyweight design pattern is a structural pattern used to minimize the memory usage and improve performance in contexts that use a large number of similar objects. By sharing common parts of the object state among multiple objects instead of keeping all the data in each object, it significantly reduces the overall memory footprint of the application. This is especially useful in applications involving extensive graphics, such as a word processor or a game, where objects like characters, textures, or trees can be reused without the need to create new ones. The pattern distinguishes between intrinsic state, which is shared and can be stored in the flyweight, and extrinsic state, which varies between objects and cannot be shared. The flyweight acts as a factory that creates and manages the flyweight objects; it ensures that objects that are identical from a resource perspective are shared rather than duplicated.

Class Diagram



FlyweightFactory
flyweights : Map<String, Flyweight>
getFlyweight(key : String) : Flyweight
"FlyweightFactory" manages flyweight objects and creates them as needed, ensuring that flyweights are shared properly.
Client
flyweight : Flyweight
operation() : void
"Client" maintains extrinsic state (unique for each flyweight) and passes it to flyweight's operations.
Creates
Uses
Flyweight
operation(extrinsicState) : void
ConcreteFlyweight
intrinsicState : String
operation(extrinsicState) : void
"ConcreteFlyweight" stores intrinsic state that is shareable across many objects and implements the operation using both intrinsic and extrinsic state.

Sequence Diagram


Client
FlyweightFactory
Flyweight
getFlyweight(intrinsicState)
alt
[if Flyweight exists]
Lookup(intrinsicState)
Returns an existing Flyweight based on the intrinsic state.
return existing Flyweight
[Flyweight does not exist]
new Flyweight(intrinsicState)
Creates a new Flyweight with the given intrinsic state.
return new Flyweight
return new Flyweight
operation(extrinsicState)
Performs operation, potentially using both intrinsic and extrinsic state.
Client
FlyweightFactory
Flyweight

Code Sample

Use cases

- General
  - Text editors (characters as flyweights)
  - Repetitive GUI elements

- In Test Automation
  - Browser instance management (one browser, multiple tabs?)
  - Elements instance management

Verisoft

Happy Coding