C# Design Patterns: Flyweight

IMPLEMENTING THE FLYWEIGHT PATTERN

Harrison Ferrone
SOFTWARE DEVELOPER

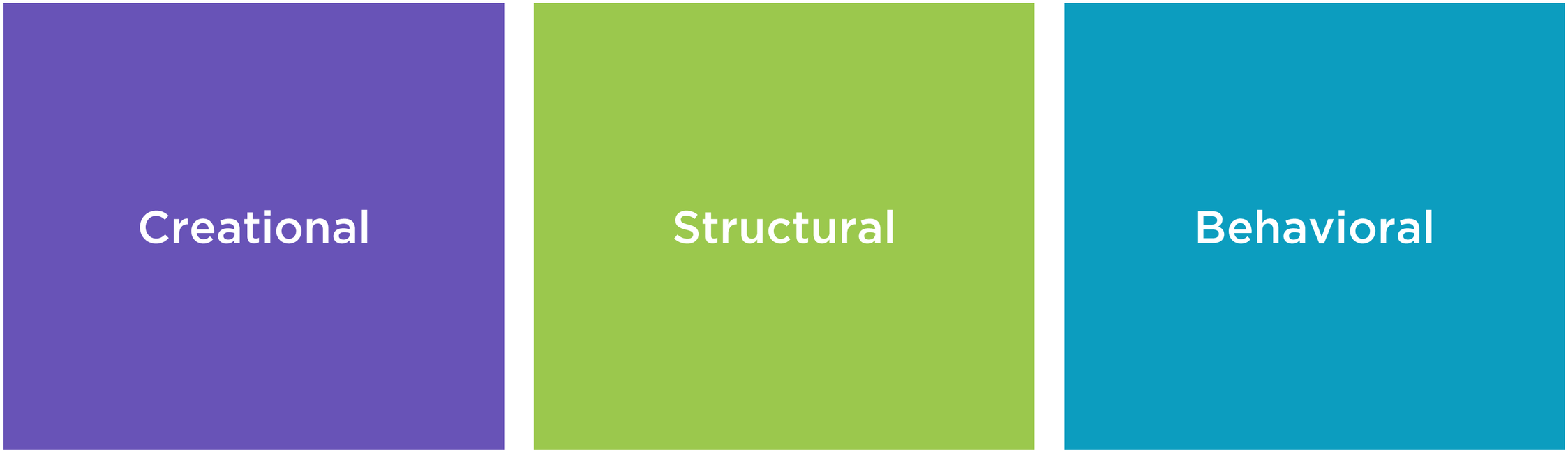
@journeyman_programmer www.paradigmshiftdev.com

Overview

The Flyweight Pattern in Practice:

- Defining a flyweight interface
- Creating concrete flyweight classes
- Handling intrinsic and extrinsic state
- Using a flyweight factory
- Reviewing use cases/practical applications

Design Pattern Categories



"Use sharing to support large numbers of fine-grained objects efficiently."

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*

Allows unchanging data to be shared

- Referred to as intrinsic state

Object-specific data can be passed in

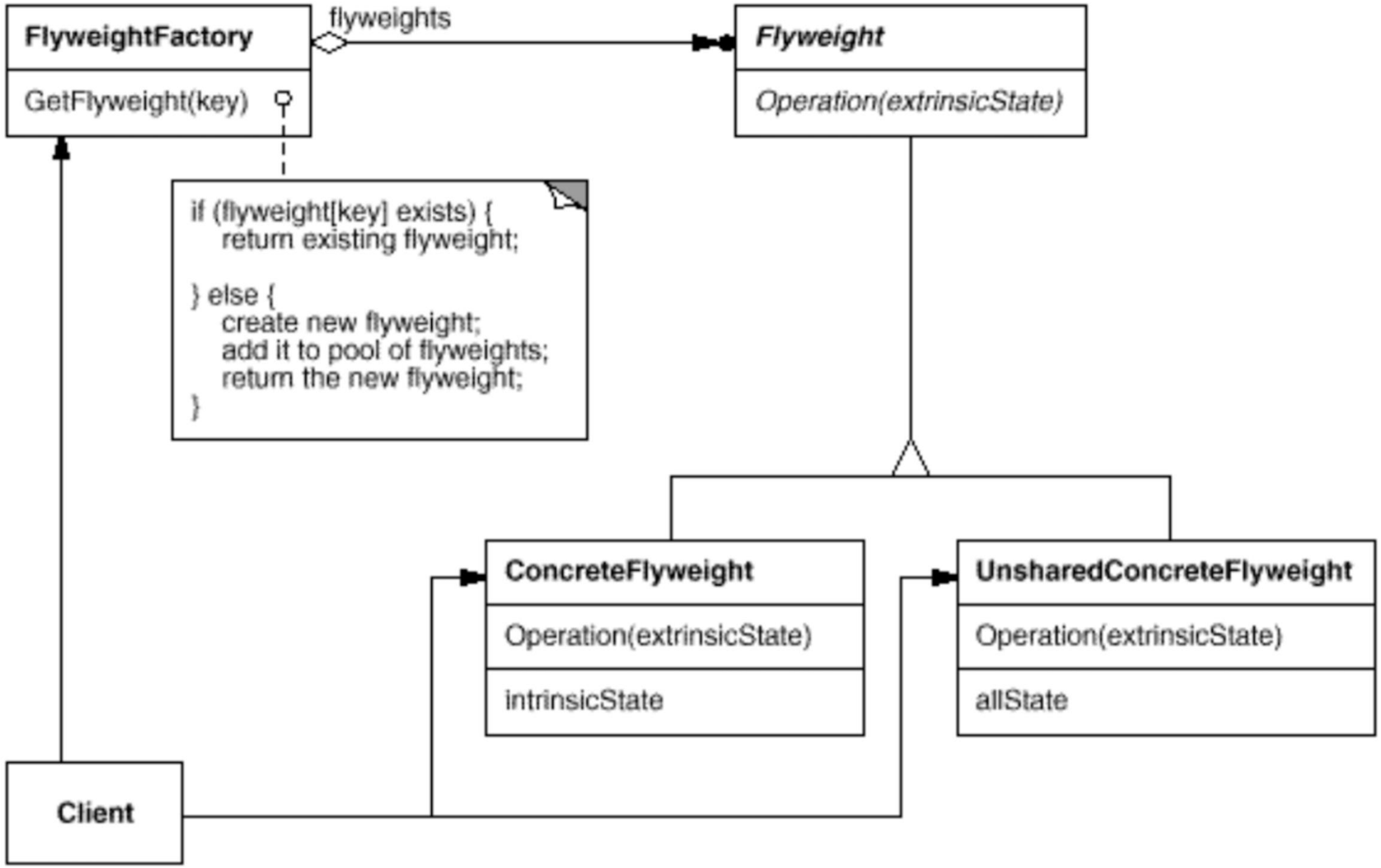
- Referred to as extrinsic state

Shared objects act as an unchanging model

- Reused as need

- Significantly cuts down on storage

Flyweight Pattern Diagram



When a large number of objects need to be created and most object state is categorized as extrinsic

Creating the Flyweight Structure

Example Scenario

Simplified drink ordering app

Each drink item will be a Concrete Flyweight

Drink factory will act as the ordering platform

Drink giveaway to show Unshared Flyweight

Intrinsic vs. Extrinsic State

Intrinsic State Simplified

Every person has a name and inner character

These remain constant throughout life

Anything unchanging is your intrinsic state

Sometimes referred to as an objects context

Extrinsic State Explained

Age, height, weight are changing properties

Worldly interactions also qualify

This is all considered extrinsic state

Separate from your context

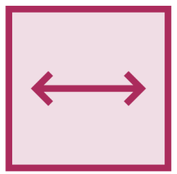
Understanding Unshared Flyweights

Use Cases and Applications

Flyweight Use Cases

Not for everything - all object creation doesn't have to use the flyweight pattern

When an application needs a lot of objects, making storage costs high



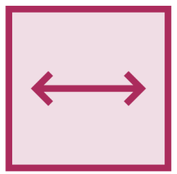
When object state can mostly be extrinsic, and many groups of objects can be replaced with a few shared objects

When the application doesn't depend on object identity

Design Pattern Implications

Run-time costs may play a part in finding and computing a shared objects extrinsic state

If applied in correct situations, this will be offset by the space savings from each additional shared flyweight object used



Savings are at their maximum capacity when lots of shared objects are used and both stored intrinsic and computed extrinsic are substantially used

Hierarchical relationships and communications can still be used by passing references to parent, or other hierarchy objects, in with the extrinsic state

Related Patterns

Composite used for hierarchical structures

State and Strategy for flyweight objects

Summary

- How to create a Flyweight interface and concrete classes
- How to distinguish and implement intrinsic and extrinsic state
- How to use a flyweight factory
- How unshared flyweights fit into the pattern
- How to identify use cases and correct applications

Contact and Course Discussion

Leave a course rating

Use the Discussion section as a resource

Get in contact and let's talk code

Follow my author profile

Happy Coding!