1. Intent:

* Flyweight is a structural design pattern
* Use sharing to support large numbers of fine-grained objects efficiently
* Simply, it is best when reducing the memory needed to be used for a large number of similar objects

1. Real-world problem: Blog common system

* Under a post on Facebook, there are several comments, each of which has the following data (attributes)
* Author name (account name, nickname,..) + Author avatar + profile url (allowing others to visit your wall when clicking avatar or name) + Text (comment)
* All are the same for users except for the comment, therefore, there must be a way to store those 3 attributes when it comes to big social networking
* For storing data, we can treat each comment as an object. If one user adds several comments, storing data this way can result in redundant data, as all three attributes are the same.
* What if billions of users comment on a Facebook post, and they fight over a topic?
* So we need to store data more efficiently, which is when we turn to the flyweight pattern for help

1. Applicability:

* A large number of objects
* High memory costs:
* Shared state: Many objects share a common state or behaviour that can be reused
* Extrinsic state: The state can be stored externally, allowing shared objects to be used for the common state

1. Other real-world problems
   1. Graphical user interface (GUI)

* Contains a lot of icons of different types (file icon, folder icon, application icon,...) that need to be displayed on a screen
* Each icon has a specific purpose as well as an appearance
* But they also share similarities such as size and color
* With the flyweight pattern, only their unique properties should be stored separately, while others can be stored externally and can be accessed by other icons
* By doing so, the developer can reduce the memory footprint and enhance performance
  1. Games
* Contains different objects that can be divided into 3 types: characters, items, and enemies
* In the same type, objects can share similarities, such as moving, flying, and hindering players from winning easily
* For example, in character type, in the Mario game, there can be different characters (Luigi, Mario, Peach,..), they can all run, jump, defeat enemies, and collect items, but the way they do so might be different
* Then, all their similar properties can be stored externally and accessed by other characters, while unique ones should be kept internally
* By doing so, the developer can reduce the memory footprint and enhance performance

1. The pattern goes together
   1. Composite + Flyweight

* When to use?: Managing hierarchical game objects like a tree of UI elements, where many components are similar
* How?
  + Use the composite tree structure to treat individual components and group objects uniformly
  + Use the flyweight pattern to share data across those components
  1. Flyweight + Facade
* When to use? Provide a simplified interface for creating and using a flyweight object in a complex system
* How?
  + Use the facade pattern to hide complexity
  + Use the flyweight pattern to optimize resource usage
  1. Flyweight + Singleton
* When to use? Ensure a single shared factory or manager for flyweight instances across the game.
* How?
  + Use the singleton to control access to flyweight storage
  + Prevent duplicate flyweight creation across game modules

1. Quiz:
   1. What is the main purpose of the Flyweight pattern

* **Reduce memory usage**
* Increase processing speed
* Encapsulate data
* Manage object creation
* Explanation: By sharing common parts among multiple objects
  1. In which scenarios is the Flyweight Pattern most useful?
* **When there is a large number of similar objects**
* When objects are immutable (not able to change)
* When using a single instance class
* When performance is not a concern
  1. Which of the following is a key component of the Flyweight Pattern?
* **Flyweight factory**
* Observer
* Decorator
* Singleton
* Explanation: Responsible for creating and managing objects
  1. What type of state do Flyweight objects typically hold?
* **Intrinsic state**
* Extrinsic state
* Static state
* Dynamic state
  1. Which design principle does the Flyweight Pattern primarily utilize?
* Single Responsibility Principle
* Open/ Closed Principle
* **Separation of Concerns**
* Object Pooling
* Explanation: Separating the intrinsic state from the extrinsic one

<https://www.geeksforgeeks.org/flyweight-design-pattern/>

<https://www.tutorialspoint.com/design_pattern/quiz_on_flyweight_pattern.htm>

<https://refactoring.guru/design-patterns/flyweight>