Group 6:

Lê Duy Bách 1551003

Đào Thanh Danh 1551004

Nguyễn Khắc Tuấn 1551043

CS 202: PROGRAMMING SYSTEM

SEMINAR DOCUMENT

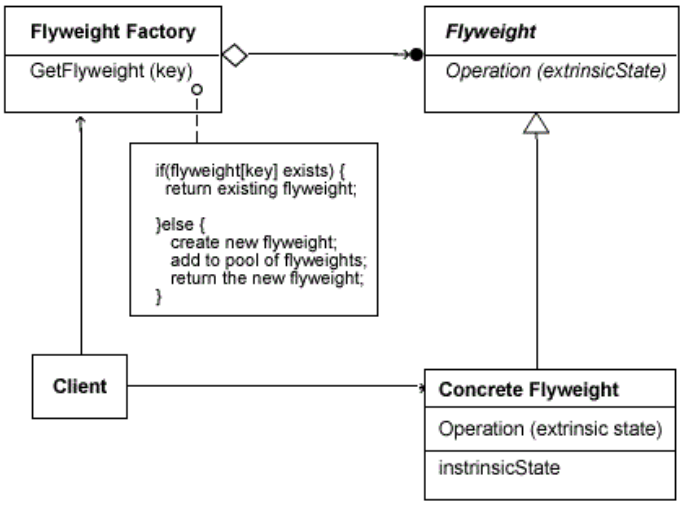
FLYWEIGHT DESIGN PATTERN

I. Introduction:

- Flyweight is one of many structural design patterns. The main purpose of flyweight is to minimize space usage by sharing as much mutual data as possible with similar objects. This desgin pattern is helpful in case of simple representation repitition taking unacceptable amount of memory.
- Flyweight design pattern shares many obvious similarities with "cache".

II. Components:

- Flyweight
  - An abstract class
  - It is an interface for flyweights to receive and act on extrinsic state.
- Concrete Flyweight
  - A derived from flyweight object
  - Is sharable
  - Any state stored must be intrinsic (independent of object's context)
- Flyweight Factory
  - Create and manage Flyweights
  - When requested, the factory returns an existing instance of flyweight or create one, if none exists
- Client
  - Maintain references to flyweights
  - Computes or stores the extrinsic state of flyweights



Class diagram of Flyweight pattern

Intrinsic: Constant and shared parts of Flyweights. Stored in memory.
Extrinsic: Non-constant parts, must be calculated or passed as arguments. Is not stored.

III. Example:

- Situation: There is one coffee shop and they want customers to be happy with their service, so they enable delivery. They have 2 kinds of coffee, served without sugar, and many customers with different information to deliver. After the delivery, the store

owner does not want to store customers' information, however he wants to keep track of how many coffee cups they sold each day. As the owner is using a very old computer, he does not want the program to takes all of his RAM to run. Let us apply the Flyweight Design Pattern to solve this problem.

- Approach:
  - Abstract flyweight class: Coffee
  - Concrete flyweight class: Coffee type A, Coffee type B
  - Flyweight factory class: Coffee Store Staves
  - Client: Coffee Store
- Action:
  - Coffee class method: virtual serveCoffee(void) = 0;
  - Concrete intrinsic state: const bool noSugar = true;
  - Coffee Store Staves: deliver(Coffee* type, int num, string name, string address);
    updateSale(Coffee* type, int num);
  - Coffee Store: Call deliver and update function
    staff.deliver(new CoffeeTypeA(void), 3, "Mr. A", "Mr. A adress");
    staff.updateSale(new CoffeeTypeA(void), 3);
  - Numbers of servings, customers' name and address are extrinsic states.

IV. Reference:

http://dofactory.com/net/flyweight-design-pattern
https://dzone.com/articles/design-patterns-in-the-real-world-flyweight
https://en.wikipedia.org/wiki/Flyweight_pattern