# Design Pattern - Singleton Pattern

I. Sinle Pattern là gì?

Single Pattern là một design pattern mà

1. Đảm bảo rằng một class chỉ có duy nhất một instance (khởi tạo )

2. Và cung cấp một cách hoàn toàn cầu để truy cấp tới instance đó

II. Tại sao cần dùng Singleton Pattern ?

- hầu hết các đối tượng trong một ứng dụng đều chịu trách nhiệm cho công việc của chúng và truy xuất dữ liệu tự lưu trữ (self-contained data) và các tham chiếu trong phạm vi được đưa ra của chúng. Tuy nhiên , có nhiều đối tượng có thêm những nhiệm vụ và có ảnh hưởng rộng hơn, chẳng hạn như quản lý các nguồn tài nguyên bị giớii hạn hoặc theo dõi toàn bộ các trạng thái của hệ thống

Ví dụ : giả sử trong ứng dụng có chức năng bật tắt nhạc nền chẳng hạn , khi người dùng mở app thì ứng dụng sẽ tự động mở nhạc nền và nếu người dùng muốn tắt thì phải vào trong setting trong app để tắt nó, trong setting của app cho phép người dùng quản lí việc này. Chắc chắn bạn phản cần duy nhất 1 instance để có thể ra lệnh bật hay tắt , tại sao ? vì đơn gỉan bạn không thể tạo 1 instance để mở nhạc rồi sau đó lại tạo 1 instance khác để tắt nhạc , lúc này sẽ có 2 instance được tạo ra , 2 instance này không liên quan đến nhau nên không thể thực hiện việc cho nhau được.

III. Làm thế nào để implement Singleton Pattern ?

1. làm sao để 1 class chỉ có thể có duy nhất 1 instance?

- Private constructor của class đó để bảo rằng class lớp khác không thể truy cập vào constructor và tạo instance mới

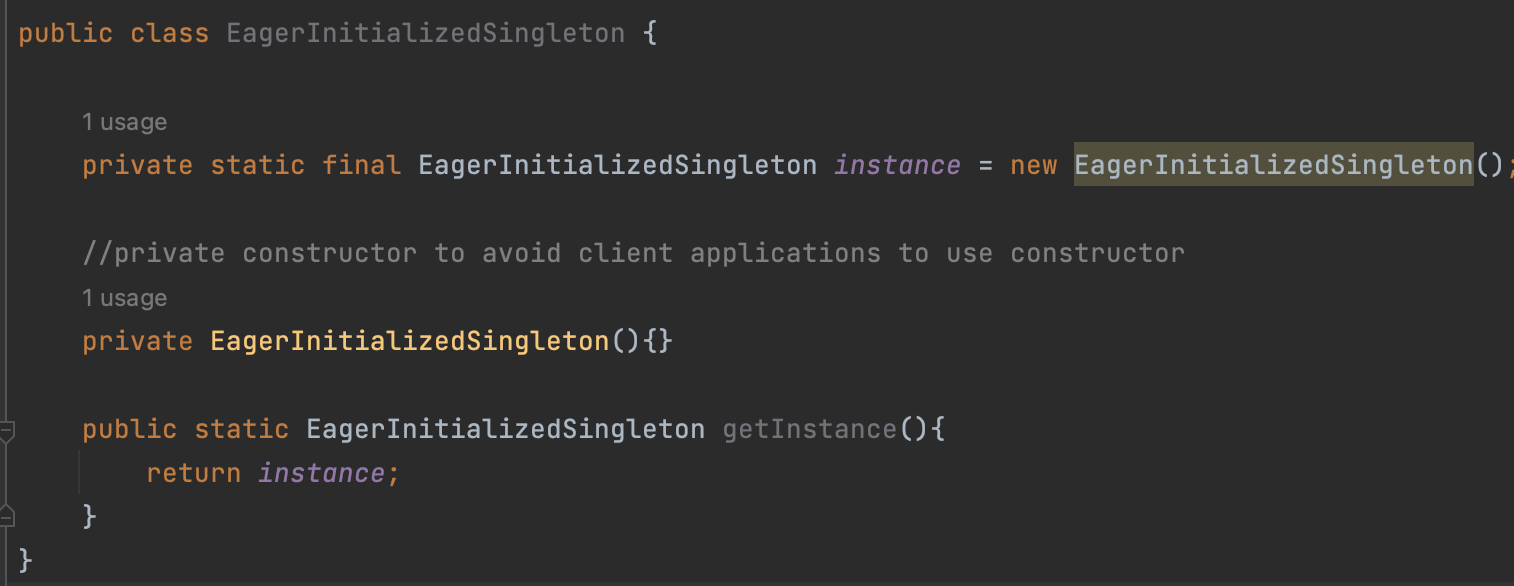
- Tạo một biến private static là thể hiện của class đó để đảm bảo rằng nó là duy nhất và chỉ được tạo ra trong class đó thôi

1. Làm sao để có thể cung cấp một cách toàn cầu để truy cấp tới instance đó

- Tạo một public static menthod trả về instance vừa khởi tạo bên trên , đây là cách duy nhất để các class khác có thể truy cập vào instance của class này

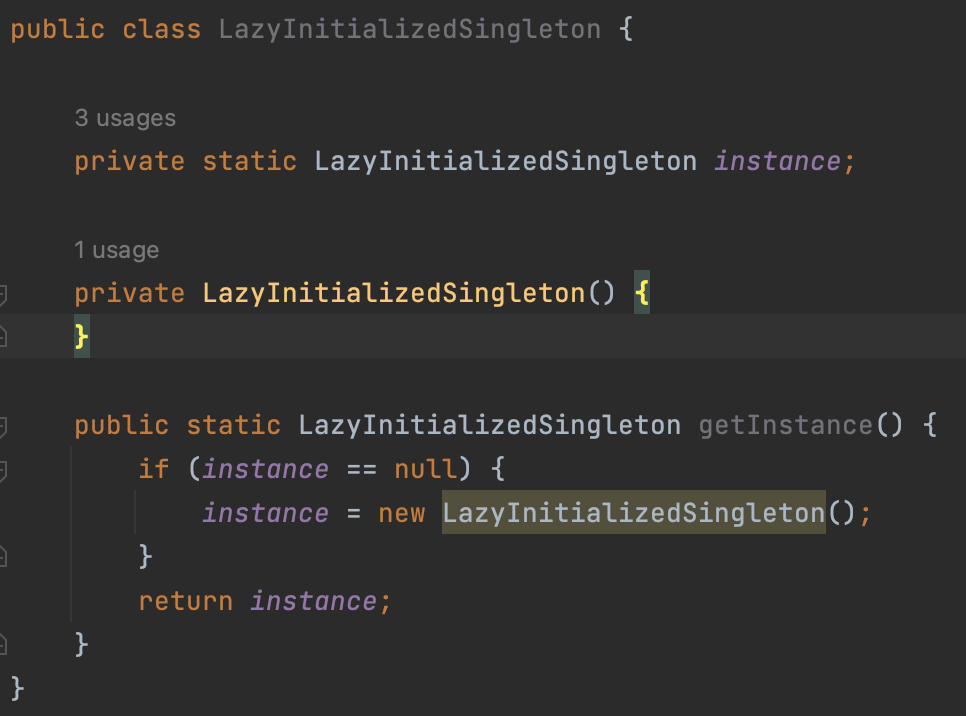
IV. Có những cách nào để implement Singleton Pattern ?

1.Eager initialization ?



- đây là cách dễ nhất nhưng có một nhược điểm là mắc dù instance đã được khởi tạo nhưng có thể sẽ không dùng tới. Vì vậy chúng ta có cách thứ 2

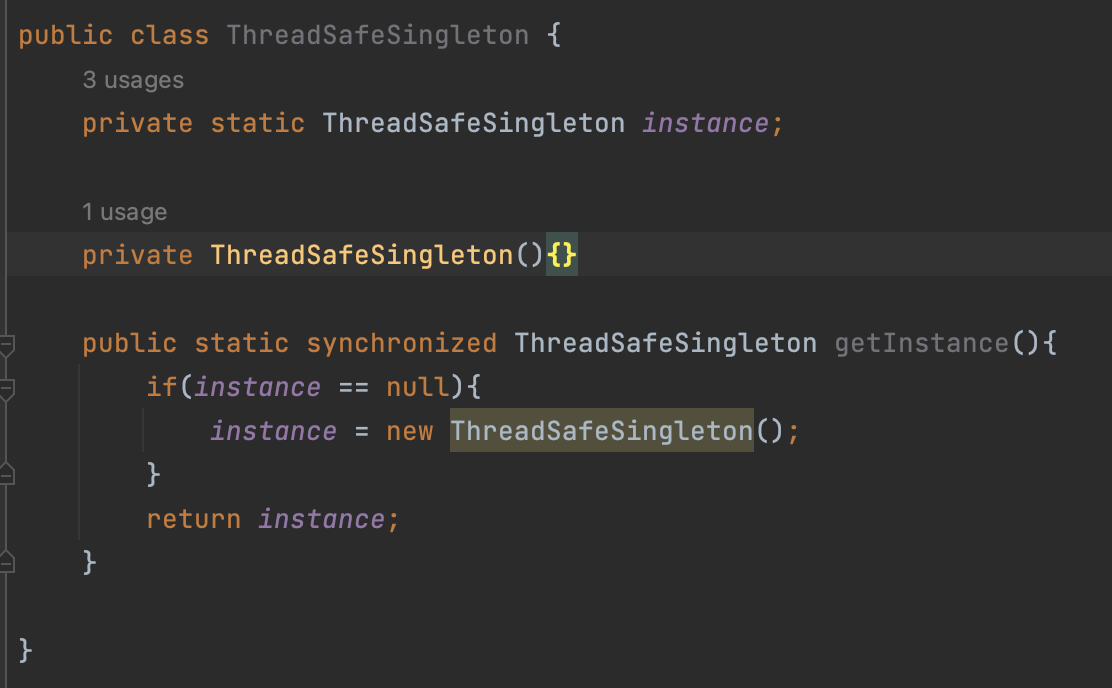
1. Lazy initialization



- Cách này đã khắc phục được nhược điểm của cách 1 Eager initialization, chỉ khi nào geinstance được gọi thì instance mới được khởi tạo. Tuy nhiên cách này chỉ sử dụng tốt trong trường hợp đơn luồng, trường hợp nếu có 2 luồng cùng chạy và cùng gọi hàm getinstance tại cùng 1 thời điểm thì đương nhiên chúng ta có ít nhất 2 thể hiện của instance. Vậy ta phải làm sao với trường hợp đa luồng . chúng ta đi tới cách tiếp thep

1. Thread safe initialization

- Cách đơn giản nhất là chúng ta gọi phương thức synchronized của hàm getinstance() và như vậy hệ thống đảm bảo rằng tại cùng một thời điểm chỉ có thể có 1 luồng có thể truy cập vào hàm getinsstance(), và đảm bảo rằng chỉ có duy nhất 1 thể hiện của class Tuy nhiên một menthod synchronized sẽ chạy rất chậm và tốn hiện năng vì vậy chúng ta cần cải tiến nó đi 1 chút.



1. Thread Safe Upgrade initializaion



- Mình tạm gọi là Thread Safe Upgrade initialization, thay vì chúng ta Thread Safe cả menthod getinstance() chúng ta chủ Thread Safe một đoạn mã quan trọng