# Creational Pattern

# 1. Creational Pattern là gì?

Creational Pattern trong Java là một nhóm các mẫu thiết kế (design patterns) tập trung vào việc khởi tạo đối tượng một cách tối ưu, hiệu quả và có kiểm soát. Mục tiêu của các mẫu này là tách biệt quá trình khởi tạo đối tượng khỏi phần logic, cho phép ứng dụng linh hoạt trong việc tạo ra các đối tượng mà không phụ thuộc vào loại cụ thể của chúng. Các Creational Patterns giúp cải thiện khả năng mở rộng, bảo trì mã nguồn và giảm độ phức tạp của quá trình khởi tạo.

# 2. Singleton Pattern là gì?

**Singleton Pattern** là một trong những mẫu thiết kế (design pattern) thuộc nhóm Creational Pattern, với mục tiêu chính là đảm bảo chỉ có **một đối tượng duy nhất** (instance) của một lớp tồn tại trong suốt vòng đời của ứng dụng, và cung cấp một điểm truy cập toàn cục đến đối tượng đó.

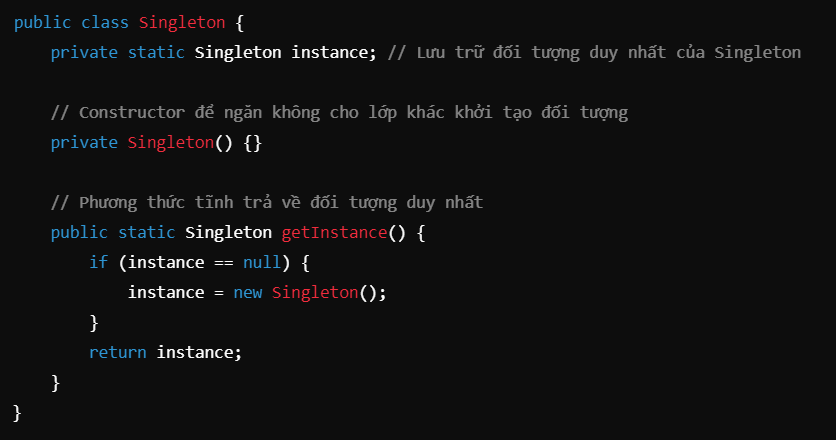
### **1. Mục đích của Singleton Pattern**

* **Đảm bảo tính duy nhất**: Đảm bảo rằng chỉ có một đối tượng duy nhất của một lớp được khởi tạo.
* **Quản lý tài nguyên**: Singleton thường được sử dụng khi có những tài nguyên cần được quản lý tập trung, chẳng hạn như kết nối cơ sở dữ liệu, cấu hình hệ thống, hay các dịch vụ toàn cục.
* **Cung cấp điểm truy cập toàn cục**: Singleton cung cấp một phương thức để truy cập đối tượng từ bất kỳ đâu trong ứng dụng mà không cần phải tạo thêm đối tượng mới.

### **2. Cách thức triển khai Singleton Pattern**

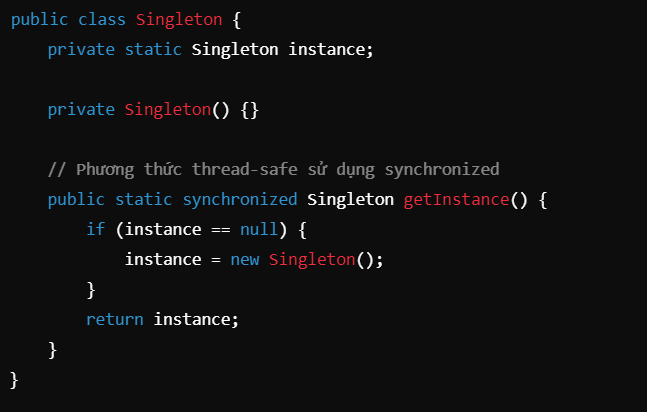
Có nhiều cách triển khai Singleton Pattern, nhưng cách phổ biến nhất là sử dụng một biến tĩnh để giữ đối tượng duy nhất và một phương thức tĩnh để trả về đối tượng này.

#### a. **Triển khai cơ bản (Lazy Initialization Singleton)**:



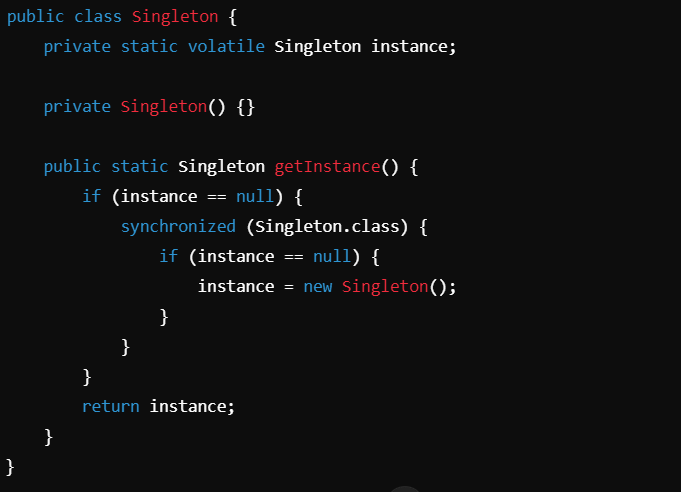
#### **b. Triển khai an toàn với đa luồng (Thread-safe Singleton):**

Để đảm bảo Singleton hoạt động đúng trong môi trường đa luồng (multi-threading), ta cần thêm các biện pháp bảo vệ như dùng synchronized hoặc Double-Checked Locking



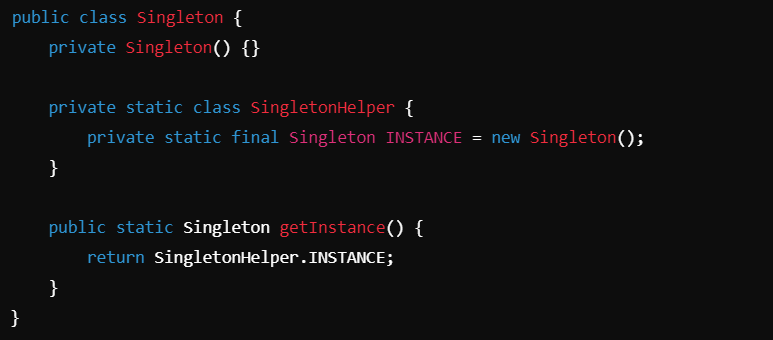
#### **c. Double-Checked Locking (DCL):**

Giải pháp này giúp cải thiện hiệu suất bằng cách tránh việc dùng synchronized mỗi khi gọi phương thức getInstance().



#### **d. Bill Pugh Singleton (Inner Static Helper Class):**

Cách này dựa vào việc sử dụng lớp nội bộ tĩnh để tạo đối tượng Singleton, vừa đơn giản vừa an toàn với đa luồng mà không cần sử dụng từ khóa synchronized.



# 3. Factory Pattern là gì?

Factory Pattern giúp **tách biệt quá trình tạo đối tượng** khỏi phần logic sử dụng đối tượng. Điều này có nghĩa là bạn có thể thay đổi cách tạo đối tượng mà không cần sửa đổi các phần khác của mã nguồn.

**Factory Pattern** là một mẫu thiết kế thuộc nhóm **Creational Design Patterns** (mẫu thiết kế tạo dựng), được sử dụng để tạo đối tượng mà không cần chỉ rõ lớp cụ thể của đối tượng đó. Factory Pattern cung cấp một cách tiếp cận linh hoạt trong việc tạo ra các đối tượng mà không yêu cầu code phải biết chính xác lớp nào sẽ được khởi tạo. Thay vì sử dụng trực tiếp toán tử new để khởi tạo đối tượng, một lớp Factory sẽ xử lý quá trình này.

### **1. Mục đích của Factory Pattern**

Factory Pattern giúp **tách biệt quá trình tạo đối tượng** khỏi phần logic sử dụng đối tượng. Điều này có nghĩa là bạn có thể thay đổi cách tạo đối tượng mà không cần sửa đổi các phần khác của mã nguồn.

Ví dụ: Trong một hệ thống cần tạo nhiều loại đối tượng dựa trên điều kiện nào đó (ví dụ, dựa trên đầu vào của người dùng hoặc dữ liệu), Factory Pattern giúp kiểm soát quá trình tạo đối tượng này, đồng thời cho phép dễ dàng mở rộng khi cần thêm các lớp mới mà không ảnh hưởng đến code hiện tại

### **2. Cách triển khai Factory Pattern**

Factory Pattern thường bao gồm:

* **Interface hoặc Abstract Class**: Định nghĩa loại đối tượng cần tạo.
* **Lớp Factory**: Lớp này chứa logic tạo đối tượng, thường là một phương thức static hoặc instance method, trả về đối tượng phù hợp dựa trên tham số đầu vào.
* **Các lớp con (Concrete Classes)**: Các lớp thực thi cụ thể của interface hoặc abstract class mà factory sẽ khởi tạo.

### **3. Ưu điểm của Factory Pattern**

* **Tính linh hoạt**: Dễ dàng thay đổi hoặc mở rộng các lớp cụ thể mà không ảnh hưởng đến mã nguồn hiện tại. Nếu cần tạo một lớp mới, chỉ cần thêm một lớp con mà không cần thay đổi lớp Factory.
* **Giảm sự phụ thuộc giữa các lớp**: Code sử dụng đối tượng không cần biết chính xác đối tượng thuộc lớp nào, chỉ cần gọi factory để tạo đối tượng.
* **Tái sử dụng mã**: Logic tạo đối tượng được gom lại tại một nơi, giúp dễ bảo trì và tái sử dụng.

### **4. Nhược điểm của Factory Pattern**

* **Phức tạp hơn**: Đối với các hệ thống nhỏ, việc tạo thêm một lớp factory có thể làm phức tạp mã nguồn và không cần thiết.
* **Khó kiểm soát khi có quá nhiều lớp con**: Khi hệ thống phát triển và có quá nhiều lớp con, lớp Factory có thể trở nên phức tạp, dẫn đến việc quản lý khó khăn hơn.