**Mẫu Singleton**

**Tên, phân loại:**

* Tên: Singleton Pattern.
* Bí danh: không có.

**Mục đích, ý định:**

* Đảm bảo rằng mỗi lớp chỉ có một thể hiện duy nhất và cung cấp khả năng truy xuất toàn cục đến thể hiện này.

**Khi nào sử dụng:**

* Khi ta cần giới hạn việc tạo ra các thể hiện từ một lớp. Đặc biệt là các đối tượng tốn nhiều tài nguyên hệ thống khi được tạo ra.

**Khả năng ứng dụng:**

* Các hệ thống cần quản lý các đối tượng tài nguyên được chia sẻ chung, chẳng hạn như printer spooler.

**Cấu trúc:**



**Các thành viên:**

* Chỉ có một lớp Singleton tham gia vào mẫu. Lớp này phải định nghĩa hàm khởi tạo là private để ngăn việc khởi tạo thể hiện từ lớp này từ bên ngoài, đồng thời bắt buộc phải có hàm getInstace() để trả một tham chiếu đến đối tượng sẽ được trả về có kiểu của lớp đó.

**Sự cộng tác:**

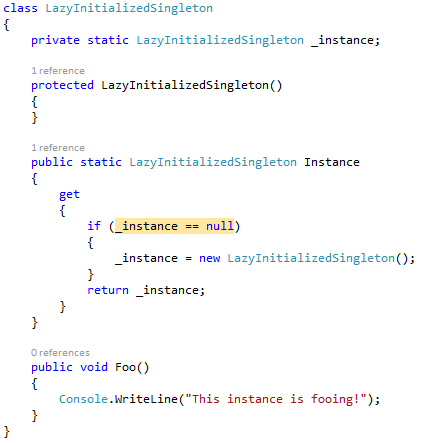
* Lớp nào muốn thể hiện mẫu Singleton cần định nghĩa một biến thành viên có kiểu là chính nó, đông thời biến này cần được đặt modifier là static private.
* Hàm khởi tạo của lớp Singleton được định nghĩa thành protected hoặc private để người dùng không thể tạo thể hiện của lớp trực tiếp từ bên ngoài.
* Phương thức getInstance() dùng để khởi tạo đối tượng duy nhất, có modifier là public static. Client chỉ dùng phương thức getInstance() này để nhận được một đối tượng lớp Singleton.

**Các hệ quả mang lại:**

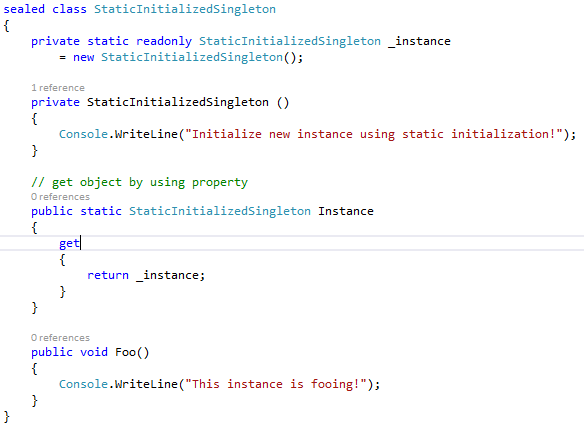
* Giới hạn số lượng thể hiện của lớp ở một con số nào đó, con số này phải có ý nghĩa nhất định trong chương trình của chúng ta.
* Nếu một lớp là Singleton, chúng ta có thể gặp khó khăn khi tạo lớp con của lớp đó.

**Chú ý liên quan đến việc cài đặt:**

* Có nhiều cách để khởi tạo đối tượng sẽ được trả về trong C#:
* Sử dụng Lazy initialization và property method:
  + Là cách thường dùng nhất.
  + Cách tiếp cận này sẽ làm trì hoãn quá trình khởi tạo đối tượng cho đến khi nào client thực sự cần đến nó.
  + Code minh họa:



* Sử dụng Static Initialization – Khởi tạo tĩnh:
  + Cách tiếp cận ở trên không phù hợp khi đối tượng được tạo ra cần được sử dụng chung, khi đó if (\_instance == null) sẽ gây ra lỗi khi lập trình đa luồng.
  + Trong cách tiếp cận này, thể hiện \_instance sẽ được tạo ra bất cứ khi nào một thành viên của lớp StaticInitializedSingleton được tham chiếu đến. Khi đó chúng ta để cho CLR khởi tạo đối tượng. Lớp tương ứng sẽ được đánh dấu là **sealed** để ngăn việc kế thừa, đồng thời biến \_instance sẽ được đánh dấu là readonly, chỉ được gán giá trị lúc khởi tạo.
  + Code minh họa:



* + Trong nhiều trường hợp, cách tiếp cận này phù hợp hơn cách đầu tiên.

**Mã nguồn minh họa:**

* **Bài toán:**
* **Sơ đồ lớp:**
* **Code mẫu:**

**Ví dụ về các hệ thống thực tế:**

* Các logger được sử dụng trong quá trình phát triển chương trình là một ví dụ về hệ thống sử dụng mẫu singleton.

**Các mẫu liên quan:**

* Abstract Factory: thường áp dụng mẫu Singleton để trả về các đối tượng của lớp ConcreteFactory duy nhất.
* Builder: dùng để xây dựng một đối tượng phức tạp, trong đó sử dụng mẫu singleton để tạo một đối tượng truy cập tổng quát.
* Prototype: dùng để sao chép một đối tượng hoặc tạo ra một đối tượng khác từ Prototype của nó, trong đó Singleton được dùng để chắc chắn chỉ có một Prototype.