BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC GIA ĐỊNH

KHOA CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN MÔN HỌC

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG NÂNG CAO

GVHD: ThS.LÊ HUỲNH PHƯỚC.

NĂM HỌC: 2022.

NHÓM: 11.

THÀNH VIÊN: NGUYỄN ĐỨC TÀI – 2008110304.

PHAN HOÀNG NAM -.2004110054.

NGUYỄN MINH TUẤN – 2008110220.

NGUYỄN HƯỞNG – 2004110008.

Tp. HỒ CHÍ MINH, 08/2022

Mục Lục

LỜI NÓI ĐẦU.

Lập trình hướng đối tượng (Object Oriented Programming – OOP) là một trong những kỹ thuật lập trình rất quan trọng và sử dụng nhiều hiện nay. Hầu hết các ngôn ngữ lập trình hiện nay như Java, PHP, .NET, Ruby, Python... Qua môn học sinh viên có thể hiểu rõ và nắm những kiến thức cơ bản nhất về hướng đối tượng cho riêng mình. Ngoài ra, sinh viên còn được trang bị những kiến thức từ cơ bản đến nâng cao nhằm phục vụ cho quá trình học và nghiên cứu thêm về Lập trình hướng đối tượng trong tương lai.

Sau quá trình học tập, trau dồi kiến thức lập trình trong học kỳ vừa qua, nhóm tôi đã ôn tập và tiến hành thực hiện bài tiểu luận này để kết thúc học phần.

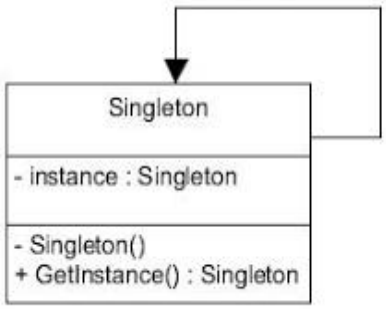
Do hạn chế về mặt thời gian và kiến thức nên bài tiểu luận còn nhiều hạn chế và sai sót. Rất mong nhận được sự đóng góp ý kiến của các thầy cô và bạn đọc để bài tiểu luận được hoàn thiện hơn.

Đề Tài: ***Hướng Dẫn Và Ví Dụ Singleton patter.***

Đôi khi, trong quá trình phân tích thiết kế một hệ thống, chúng ta mong muốn có những đối tượng cần tồn tại duy nhất và có thể truy xuất mọi lúc mọi nơi. Làm thế nào để hiện thực được một đối tượng như thế khi xây dựng mã nguồn? Chúng ta có thể nghĩ tới việc sử dụng một biến toàn cục (global variable : public static final). Tuy nhiên, việc sử dụng biến toàn cục nó phá vỡ quy tắc của **OOP (encapsulation)**. Để giải bài toán trên, người ta hướng đến một giải pháp là sử dụng **Singleton pattern**.

I. Singleton Pattern là gì?

- ✓ **Singleton** là 1 trong 5 design pattern của nhóm **Creational Design Pattern**.
- ✓ **Singleton** đảm bảo chỉ duy nhất **một thể hiện (instance)** được tạo ra và nó sẽ cung cấp cho bạn một method để có thể truy xuất được thể hiện duy nhất đó mọi lúc mọi nơi trong chương trình.



- ✓ Sử dụng Singleton khi chúng ta muốn (.):
  - * Đảm bảo rằng chỉ có một instance của lớp.
  - * Việc quản lý việc truy cập tốt hơn vì chỉ có một thể hiện duy nhất.
  - * Có thể quản lý số lượng thể hiện của một lớp trong giới hạn chỉ định.

II. Implement Singleton Pattern như thế nào?

Có rất nhiều cách để implement Singleton Pattern. Nhưng dù cho việc implement bằng cách nào đi nữa cũng dựa vào nguyên tắc dưới đây cơ bản dưới đây:

- * **private constructor** để hạn chế truy cập từ class bên ngoài.
- * Đặt **private static final variable** đảm bảo biến chỉ được khởi tạo trong class.
- * Có một method **public static** để **return instance** được khởi tạo ở trên.

Chúng ta sẽ có 5 bước chính để triển khai Singleton bao gồm:

- * Định nghĩa thuộc tính **static** riêng tư trong class "single instance"
- * Định nghĩa hàm **static** công khai trong một class
- * Thực hiện **"Lazy initialization"** cho lần sử dụng đầu tiên trong hàm cần truy cập.

* Định nghĩa các hàm đều được **protected** hoặc **private.**
* Client chỉ có thể sử dụng hàm để truy cập và thao tác với Singleton.

III. Những cách nào để implement Singleton Pattern:

1. Eager initialization:

- Singleton Class sẽ được khởi tạo ngay khi chương trình chạy. Đây là cách dễ dàng nhất, đơn giản nhất nhưng nó có một nhược điểm: mặc dù instance đã được khởi tạo nhưng có thể sẽ không dùng tới.

- Eager initialization dễ cài đặt nhưng nó dễ dàng bị phá vỡ bởi Reflection.

2. Static block initialization:

- Chúng ta làm tương tự như Eager initialization chỉ khác ở phần static block cung cấp thêm lựa chọn cho việc handle exception hay các xử lý khác.

3. Lazy Initialization:

- Lazy Initialization ra đời đã giúp khắc phục nhược điểm của Eager Initialization. Chỉ khi nào được gọi, instance mới được khởi tạo. Nó giúp bạn không phải khởi tạo class khi bạn không cần. Tuy nhiên, đối với thao tác create instance quá chậm thì người dùng có thể phải chờ lâu cho lần sử dụng đầu tiên.

- Lazy Initialization cũng chỉ hoạt động tốt với trường hợp chương trình của chúng ta là đơn luồng (single-thread). Nếu tại một thời điểm mà có hai luồng (multi-thread) cùng gọi đến phương thức getInstance() thì sẽ xảy ra trường hợp 2 thực thể (instance) sẽ được khởi tạo cùng một lúc.Để khắc phục nhược điểm trên, phương pháp Thread Safe Singleton đã được ra đời.

4. Thread Safe Singleton:

- Để giải quyết vấn đề của lazy initialization singleton, chúng ta sử dụng từ khóa synchronized. Việc set method về *synchronized* sẽ lock tất cả các thread không cho thread đó gọi method *threadSafeSingletonInstance()* cho đến khi thread đang gọi thực thi xong.

- Vấn đề của cách implement trên là trong trường hợp nhiều thead cùng gọi, các thread phải xếp hàng tuần tự thực hiện việc request instance. Chúng ta có thể giải quyết vấn đề trên bằng cách chỉ lock khi *threadSafeSingletonInstance* này chưa được khởi tạo.

- Để khắc phục nhược điểm của Lazy Initialization, chúng ta thêm synchronized vào public method. Khi đó, chỉ có một instance được khởi tạo bởi một thread tại một thời điểm.

- Tuy nhiên, cách trên vẫn có nhược điểm là làm giảm hiệu năng của app khi mỗi lần gọi vì getInstance() là một synchronized method. Vậy nên chúng ta có 1 cách khác bổ sung như sau.

=> Như vậy, chúng ta chỉ tốn sức trong lần gọi getInstance() đầu tiên.

5. Double Check Locking Singleton:

- Để có một singleton trong môi trường đa luồng, chúng ta có thể sử dụng một khóa.Nếu đối tượng đã tồn tại, không có ích gì khi lấy ra một ổ khóa.Như vậy, bạn có một kiểm tra đầu tiên bên ngoài ổ khóa.

- Tuy nhiên, ngay cả khi đối tượng không tồn tại trước khi bạn xem xét, một luồng khác có thể đã tạo nó giữa điều kiện if và câu lệnh lock.

=> Do đó, bạn cần kiểm tra lại bên trong ổ khóa.

- Tuy nhiên, cách tốt nhất để viết một singleton là sử dụng một hàm tạo tĩnh:
- Bạn nên chuyển sang lớp Lazy khi có thể vì mẫu này được sử dụng với tùy chọn Khởi tạo và Công bố. (lưu ý: nghịch đảo cũng có sẵn trong trường hợp quá trình tạo không an toàn cho luồng nhưng việc xuất bản thể hiện thông qua tùy chọn Publication) Multithreaded Singleton: Cách tốt nhất để sử dụng khóa kiểm tra kép.

- Nếu bạn tạo đối tượng trong trình khởi tạo trường, bạn không cần khóa:

- Ngoài ra - hãy nhớ rằng khóa chỉ kiểm soát việc tạo đối tượng, đối tượng vẫn cần được bảo mật theo luồng nếu bạn đang sử dụng nó trong nhiều luồng.

Khi chúng ta cố gắng thực thi phương thức của lớp singleton bằng cách sử dụng các thư viện Parallel. Nó không nhận dạng được hành vi singleton do đa luồng đang thực thi trong TPL, điều này gây ra lỗi của khái niệm Singleton Pattern. Để khắc phục vấn đề này, có khái niệm khóa đối tượng để tại một thời điểm chỉ có một luồng có thể truy cập nó. Nhưng đây không phải là cách tiếp cận hiệu quả vì việc kiểm tra khóa sẽ tạo ra những chiếc đồng hồ không cần thiết cho đối tượng. Để tránh nó, chúng tôi sử dụng "Kiểm tra khóa kép".

6. Bill Pugh Singleton Implementation:

- Trước Java memory có rất nhiều issue và các cách trên đều fail khi có quá nhiều thread gọi method getInstance() của Singleton class đồng thời. Vì vậy, Bill Pugh đưa ra một cách triển khai Singleton mới bằng cách sử dụng inner static helper class.
- Mọi người thấy cách này quá nhanh, quá gọn mà vẫn an toàn. Khi Singleton class được load, SingletonHelper class sẽ vẫn chưa được load vào memory. Chỉ khi method getInstance() được gọi, helper class mới được load và tạo singleton class instance. Cách này cũng không yêu cầu synchronization và check null nhiều lần.

7. Phá vỡ cấu trúc Singleton Pattern bằng Reflection:

- **Phá vỡ cấu trúc Singleton Pattern bằng Reflection**

- **Reflection** có thể được dùng để phá vỡ Pattern của **Eager Initialization**

8. Enum Singleton:

- Khi dùng **enum** thì các params chỉ được khởi tạo 1 lần duy nhất, đây cũng là cách giúp bạn tạo ra Singleton instance

Lưu ý:

- Enum có thể sử dụng như một Singleton, nhưng nó có nhược điểm là không thể extends từ một lớp được, nên khi sử dụng cần xem xét vấn đề này.

Hàm **constructor** của **enum** là **lazy**, nghĩa là khi được sử dụng mới chạy hàm khởi tạo và nó chỉ chạy duy nhất một lần. Nếu muốn sử dụng như một eager singleton thì cần gọi thực thi trong một **static block** khi start chương trình.

9. Serialization and Singleton:

- Đôi khi trong các hệ thống phân tán (distributed system), chúng ta cần implement interface **Serializable** trong lớp Singleton để chúng ta có thể lưu trữ trạng thái của nó trong file hệ thống và truy xuất lại nó sau.

IV. Sử dụng Singleton Pattern khi nào?

Dưới đây là một số trường hợp sử dụng của Singleton Pattern thường gặp:

* Vì class dùng Singleton chỉ tồn tại 1 Instance (thể hiện) nên nó thường được dùng cho các trường hợp giải quyết các bài toán cần truy cập vào các ứng dụng như: Shared resource, Logger, Configuration, Caching, Thread pool, …
* Một số design pattern khác cũng sử dụng Singleton để triển khai: Abstract Factory, Builder, Prototype, Façade,…
* Đã được sử dụng trong một số class của core java như: java.lang.Runtime, java.awt.Desktop.

V. Tổng Kết:

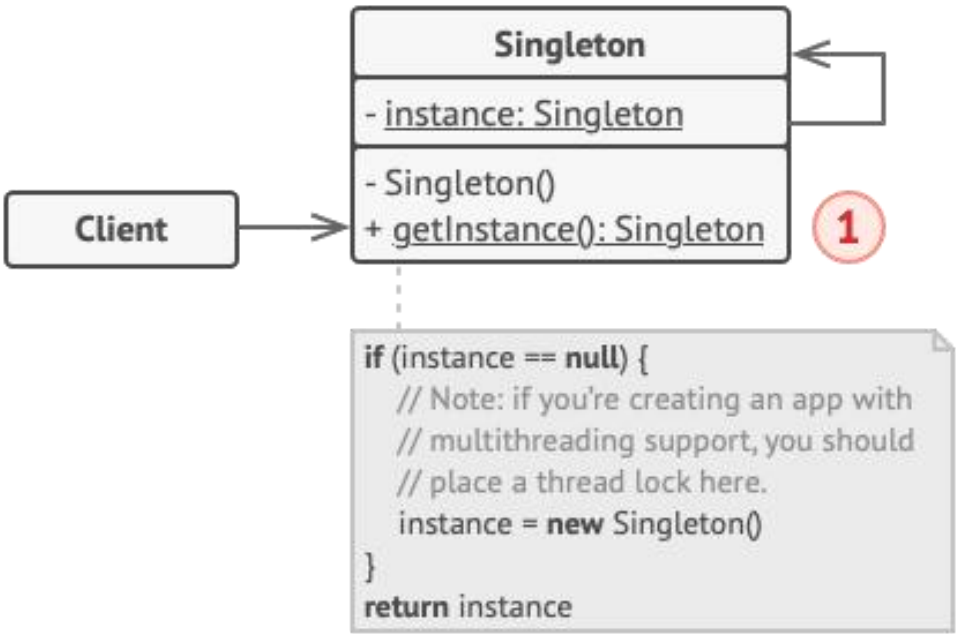
Có rất nhiều cách implement cho Singleton, mình thường sử dụng **BillPughSingleton** vì có hiệu suất cao, sử dụng **LazyInitializedSingleton** cho những ứng dụng chỉ làm việc với ứng dụng **single-thread** và sử dụng **DoubleCheckLockingSingleton** khi làm việc với ứng dụng **multi-thread**. Tùy theo trường hợp cụ thể, bạn hãy chọn cho mình cách implement phù hợp.

VI. Ưu Nhược Điểm của Singleton:

1. Ưu Điểm:

Ba ưu điểm vượt trội nhất của Singleton chính là những đặc điểm chúng ta đã nhắc lại khác nhiều lần trong bài đó là:

* Bạn có thể chắc chắn rằng mỗi class chỉ có một instance duy nhất.
* Bạn có thể truy cập instance ở bất cứ đâu và bất cứ khi nào
* Singleton chỉ khởi tạo khi bạn gọi chúng lần đầu tiên (gọi khi nào khởi tạo khi ấy).



2. Nhược Điểm:

Singleton cũng có khá nhiều nhược điểm và đặc biệt những nhược điểm này sẽ thể hiện rất rõ, ảnh hưởng lớn nếu như bạn thực hiện một dự án lớn như:

- Vi phạm nguyên tắc **Single Responsibility Principle –** *nguyên tắc đơn nhiệm***,** một pattern giải quyết cùng lúc 2 vấn đề.
- Mẫu Singleton có thể ẩn đi những thiết kế xấu cho instance khi các component trong chương trình biết rõ về nhau.
- Singleton yêu cầu xử lý đặc biệt trong mộ trường đa luồng, để nhiều luồng không tạo ra một đối tượng Singleton nhiều lần.
- Singleton thường xuyên bị các lập trình viên lâu năm đánh giá là "Evil", vì Singleton tạo ra quá nhiều phụ thuộc, không thể sử dụng đa hình và dễ tạo ra các bug làm họ phải **debug** "thâu đêm".

Singleton có rất nhiều nhược điểm thực sự rất đáng quan ngại. Vì thế, trước khi triển khai Singleton, bạn nên cân nhắc và lưu ý về những nhược điểm của Singleton nhé!

Bảng Công Tác Và Tiến Độ Thực Hiện:

Họ Và Tên	Nhiệm Vụ	Tiến Độ Hoàn Thành
Nguyễn Đức Tài (NT)	Tìm hiểu + code ví dụ: + Khái niệm + Sử dụng Singleton Pattern khi nào +. Tác hại của Singleton Pattern	
Phan Hoàng Nam	Tìm hiểu + code ví dụ: +Powpoint + Eager initialization: + Static block initialization: + Lazy Initialization:	
Nguyễn Hưởng	Tìm hiểu + code ví dụ: + Thread Safe Singleton: + Double Check Locking Singleton: + Bill Pugh Singleton Implementation	
Nguyễn Minh Tuấn	Tìm hiểu + code ví dụ: + Phá vỡ cấu trúc Singleton Pattern bằng Reflection: + Enum Singleton: + Serialization and Singleton:	Covid

LỜI CÁM ƠN

Trước tiên, tôi xin bày tỏ lòng biết ơn sâu sắc đến thầy ThS. LÊ HUỲNH PHƯỚC, người đã truyền cho tôi niềm đam mê khoa học, người đã luôn tận tình bảo ban và đồng hành cùng tôi trong suốt quá trình học tập và nghiên cứu.

Tôi xin gửi lời cám ơn đến quý thầy cô khoa Công Nghệ Thông Tin, đã dạy bảo và truyền đạt kiến thức cho tôi trong suốt quá trình học tập.

Cuối cùng, tôi xin chân thành cám ơn Ban giám hiệu Trường Đại Học Gia Định đã luôn tạo điều kiện cho tôi trong suốt học kỳ vừa qua.

Tp.Hồ Chí Minh, ngày 26 tháng 08 năm 2022.

Sinh viên thực hiện