* Design pattern là gì?
* Cấu trúc của 1 desgin pattern
* Lợi và hại
* Các loại design pattern

1. Singleton Pattern
2. Template Method Design Pattern
3. Factory Method Design Pattern
4. Design pattern là gì?

* Design pattern là các giải pháp tổng thể đã được tối ưu hóa, được tái sử dụng cho các vấn đề phổ biến trong thiết kế phần mềm mà chúng ta đã gặp phải thường ngày. Đây là các tập giải pháp đã được suy nghĩ, đã được giải quyết trong tình huống cụ thể.
* Những lập trình viên có thể áp dụng giải pháp này để giải quyết các vấn đề tương tự. Các vấn đề mà bạn gặp phải có thể bạn sẽ tự nghĩ ra cách giải quyết nhưng có thể đó chưa phải là cách tối ưu nhất
* Giống với cấu trúc dữ liệu và giải thuật, nó không phải là ngôn ngữ cụ thể nào cả. Design pattern có thể thực hiện bằng nhiều ngôn ngữ lập trình (tổng quát và thuần thúy như Java, C# hoặc hỗ trợ OOP như Python, JS,..). Nó giúp bạn giải quyết vấn đề 1 cách tối ưu nhất, cung cấp cho bạn các giải pháp lập trình hướng đối tượng.

1. Cấu trúc của 1 desgin pattern

* Mỗi design pattern đều giải quyết vấn đề của nó, và dựa trên vấn đề, nó gợi ý 1 hoặc nhiều design pattern cái được cho là phù hợp
* Giải pháp tốt nhất để giải quyết là tuân theo 3 vấn đề:  
  + problem: “What’s it solve?”

+ UML: “How’s it solve?”

+ code example: “How to implement it”

1. Lợi và hại

* Lợi:

+ giải quyết vấn đề 1 cách hiệu quả

+ cải thiện code (tái sử dụng, mở rộng quy mô khi dự án lớn hơn,..)

+ giảm thiểu lỗi và tiêu tốn ít thời gian hơn khi kiểm thử

+ better communication

* Hại:

+ đôi lúc giải quyết sai vấn đề

+ yêu cầu kiến thức sâu rộng

+ hạn chế tính sáng tạo và sức mạnh của ngôn ngữ lập trình

+

+

1. Các loại desgin pattern

* Creational problem: (nhóm khởi tạo)

+ signleton pattern

+ factory method

+ abtract method

+ builder

+ prototype

* Structural pattern: (nhóm cấu trúc)

+ adapter

+ façade

+ decorator

+ flyweight

+ proxy

+ composite

+ bridge

* Behavior pattern: (nhóm tương tác)

+ interpreter

+ template method

+ chain of responsibility

+ command

+ iterator

+ mediator

+ mementor

+ observer

+ stategy

+ state

+ visitor

\*) Singleton pattern

- Khái niệm: Singleton pattern là 1 design pattern mà đảm bảo rằng 1 class chỉ có duy nhất 1 instance (khởi tạo) và cung cấp 1 cách toàn cầu để truy cập tới instance đó. Nói cách khác, mục đích của Singleton là kiểm soát việc tạo đối tượng, giới hạn số lượng đối tượng, chỉ còn 1 đối tượng cho mỗi ứng dụng

- TH sử dụng Singleton pattern:

+ Khi giải quyết các bài toán cần truy cập vào các ứng dụng: shared resources, logger, configuration, thread pool, …

+ 1 số design pattern khác cũng sử dụng singleton để triển khai: abstract factory, builder, prototype, facade, …

+ đã được sử dụng trong 1 số class core của java: java.lang.Runtime, java.awt.Desktop, java.util.Calendar, ..

- Lợi ích khi dùng single pattern:

+ Có thể quản lý số lượng thể hiện của một lớp trong giới hạn ta muốn.

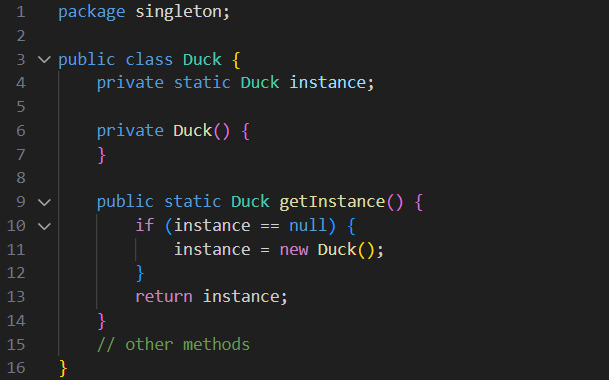
- How to implement signleton pattern

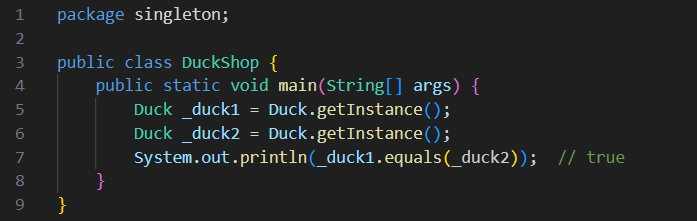
+ Hàm tạo hay còn gọi là constructor có thể truy cập với private tức chỉ có thể truy xuất trong nội bộ của class.

+ Thể hiện của nó được đặt dạng ' private static final variable' để đảm bảo biến chỉ được khởi tạo trong class.

+ có một method 'public static' để trả về thể hiện được đề cập bên trên.

+ gồm 4 TH: Eager initialization, Lazy Initialization, Thread Safe, Double Check Locking.





\*) Template method

- Khái niệm: Template method định nghĩa 1 bộ khung của 1 thuật toán trong 1 chức năng, chuyển giao việc thực hiện nó cho các lớp con. Mẫu template method cho phép lớp con định nghĩa lại cách thực hiện của 1 thuật toán mà không thay đổi cấu trúc thuật toán.

- TH sử dụng template method:

+ khi có thuật toán với nhiều bước và mong muốn cho phép tùy chỉnh chúng trong lớp con

+ mong muốn chỉ có một triển khai phương thức trừu tượng duy nhất của 1 thuật toán

+ mong muốn hành vi chung giữa các lớp con nên được được đặt ở 1 lớp chung

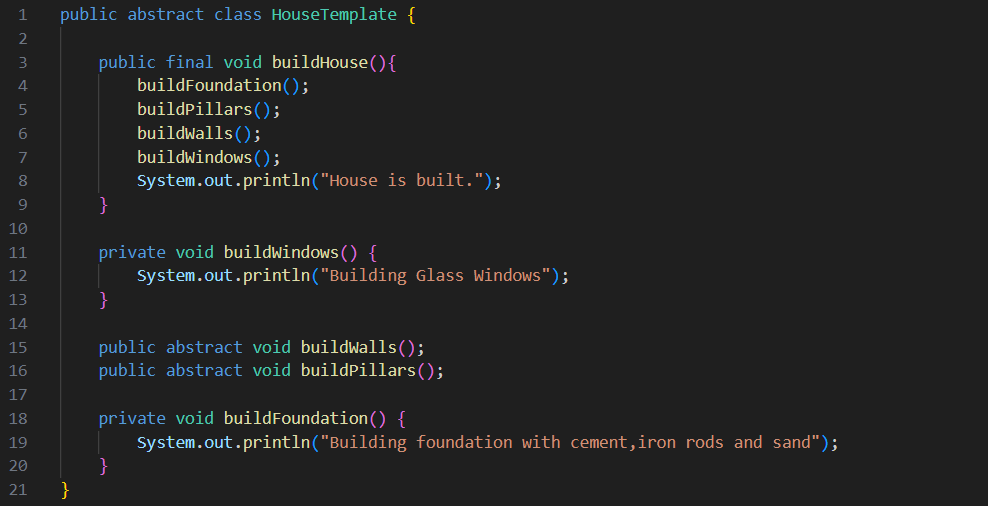
+ các lớp cha có thể gọi các hành vi trong các lớp con của chúng 1 cách thống nhất (step by step)

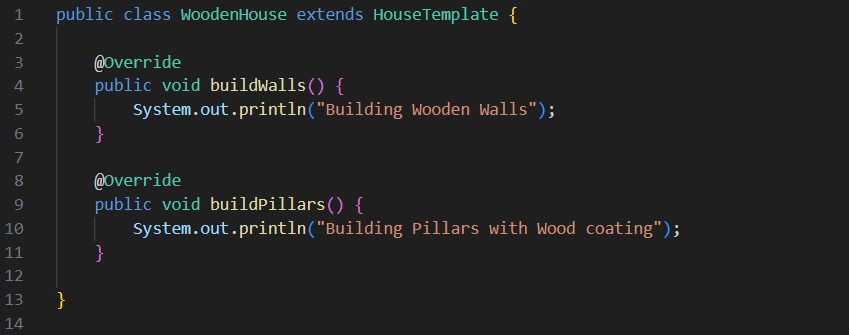
- Lợi ích khi dùng template method:

+ tái sử dụng code, tránh trùng lặp code: đưa ra phần trùng lặp vào lớp cha

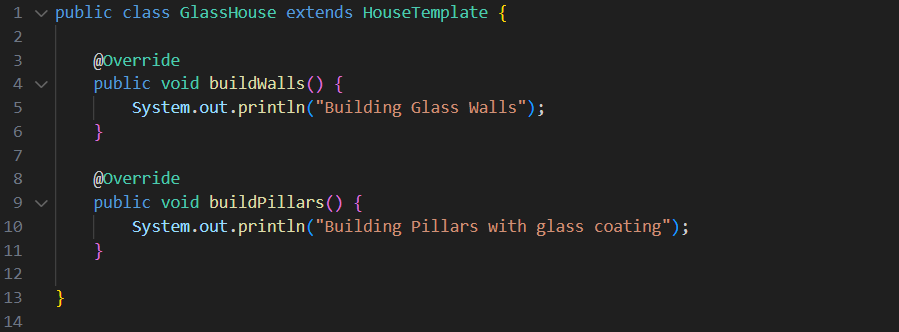
+ cho phép người dùng overide chỉ 1 số thành phần nhất định của thuật toán lớn, làm cho chúng ít bị ảnh hưởng hơn bởi những thay đổi xảy ra với thành phần khác của thuật toán

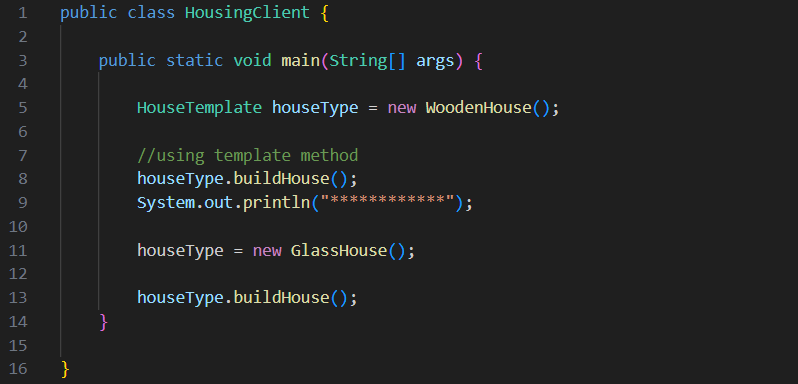
- How to implement template method

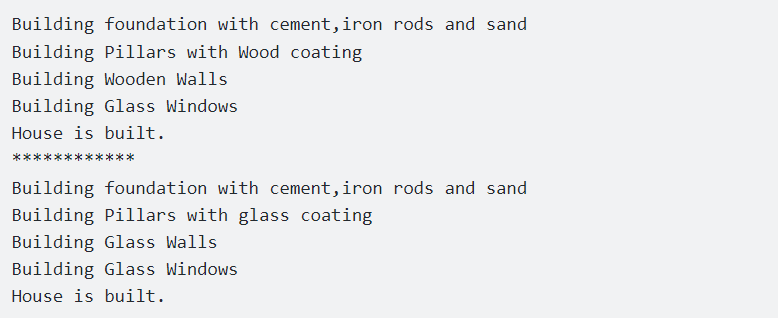
HouseTemplate:  


Từ template trên, xây dựng ra 2 loại nhà kính và nhà gỗ:  
+ WoodenHouse:  


+ GlassHouse:



Thực thi chương trình:  


Output:  


\*) Factory method

- Khái niệm: Factory method là 1 creational design pattern nhằm định nghĩa 1 class interface cho việc tạo ra tạo ra 1 đối tượng, nhưng các lớp con (sub-classes) sẽ quyết định lớp nào sẽ khởi tạo. Nhiệm vụ của factory method là quản lý và trả về các đối tượng theo yêu cầu, giúp cho việc khởi tạo đối tượng 1 cách linh hoạt hơn.

- Factory (Nhà máy) làm nhiệm vụ sản xuất hàng hóa thì Factory Patterns cũng vậy nhưng giúp chúng ta sản xuất đối tượng. Nó sẽ che giấu đi logic tạo ra những đối tượng đó giống như việc chúng ta chỉ có thể nhìn thấy nguyên liệu đi vào và sản phẩm đi ra từ nhà máy mà không thể biết sản phẩm phải trải qua những quy trình ra sao.

- TH sử dụng factory method:

+ chúng ta có 1 super class với nhiều class con và dựa trên đầu vào, chúng ta cần trả về 1 class con. Mô hình này giúp chúng ta đưa trách nhiệm của việc khởi tạo 1 lớp từ phía người dùng (client) sang lớp Factory

+ chúng ta không biết sau này sẽ cần đến những lớp con nào nữa. Khi cần mở rộng, hãy tạo ra sub class và implement thêm vào factory method cho việc khởi tạo sub class này

- Lợi ích khi dùng factory method:

+ giúp giảm sự phụ thuộc giữa các module (loose coupling): cung cấp 1 hướng tiếp cận với interface thay thì các implement, giúp chương trình độc lập với những lớp cụ thể mà chúng ta cần tạo 1 đối tượng, code ở phía client không bị ảnh hưởng khi thay đổi logic ở factory hay sub class

+ mở rộng code dễ dàng hơn: khi cần mở rộng, chỉ việc tạo ra sub class và implement tgeen vào factory method

+ khởi tạo các objects mà che giấu đi xử lý logic của việc khởi tạo đấy. người dùng không biết logic thật sự được khởi tạo bên dưới phương thức factory

+ dễ dàng quản lý life cycle của các object được tạo bởi factort pattern

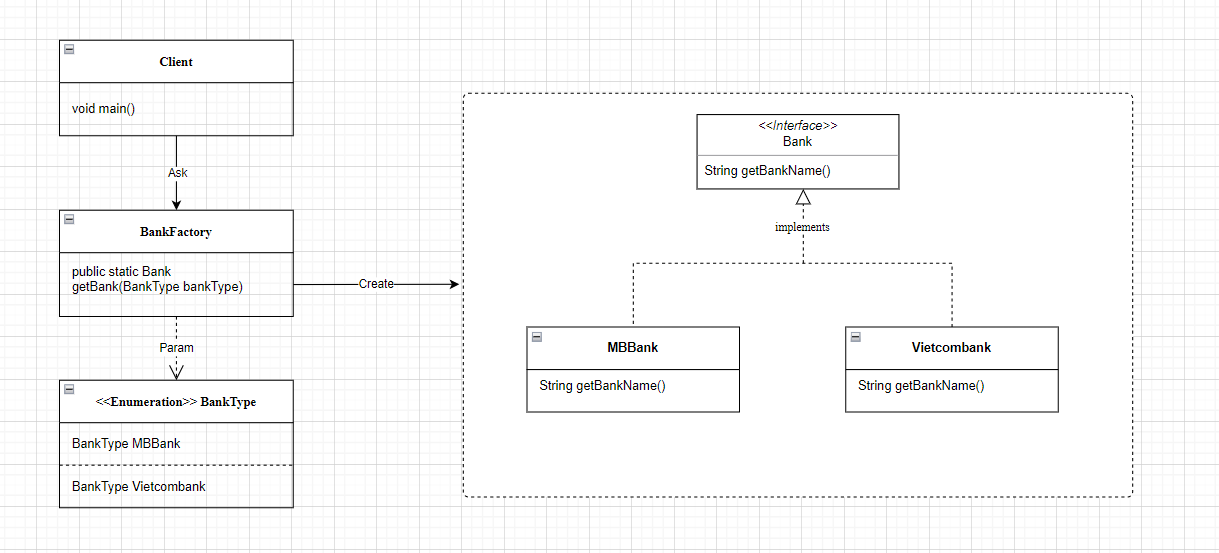
+ thống nhất về name convention: giúp cho các dev có thể hiểu cấu trúc của source code

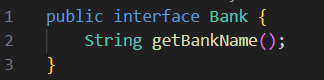
- các thành phần tạo nên 1 factory pattern:  
 + Super Class: môt supper class trong Factory Pattern có thể là một interface, abstract class hoặc cũng có thể là một class thông thường.

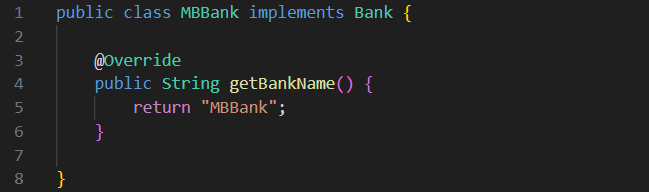
+ Sub Classes: các class con sẽ kế thừa từ class cha theo nghiệp vụ riêng của nó.

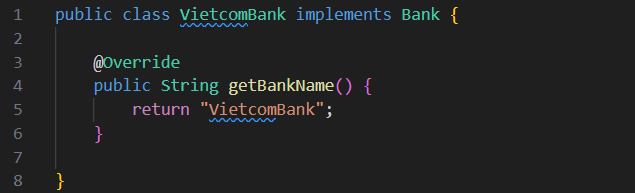
+ Factory Class: một class chịu tránh nhiệm khởi tạo các đối tượng class con dựa theo tham số đầu vào. (Lớp này là Singleton hoặc cung cấp một public static method cho việc truy xuất và khởi tạo đối tượng. Factory class sử dụng if-else hoặc switch-case để xác định class con đầu ra.)

- How to implement factory method

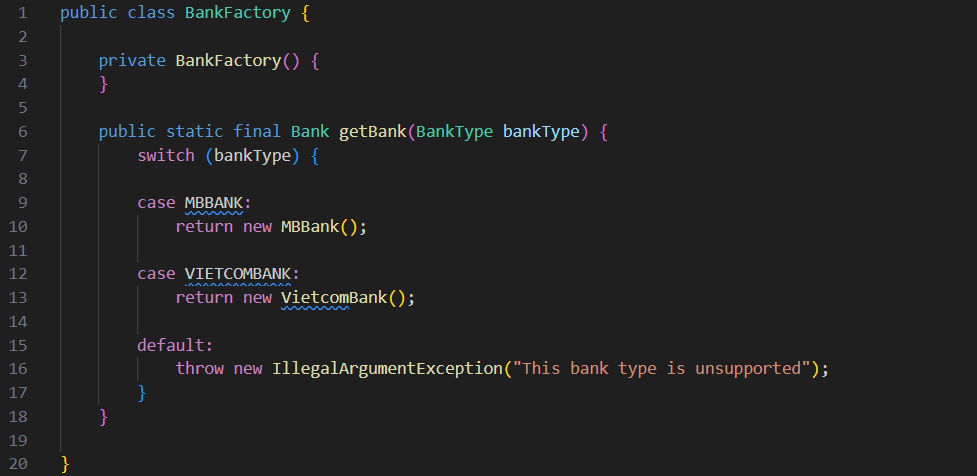


Supper class:  


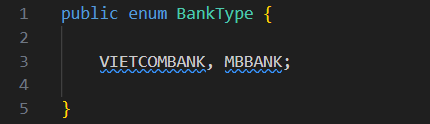
Sub classes:  




Factory class:



Bank type:



Client:

