**Design Pattern** là một kỹ thuật trong lập trình hướng đối tượng, nó khá quan trọng và mọi lập trình viên muốn giỏi đều phải biết. Được sử dụng thường xuyên trong các ngôn ngữ OOP. Nó sẽ cung cấp cho bạn các "mẫu thiết kế", giải pháp để giải quyết các vấn đề chung, thường gặp trong lập trình. Các vấn đề mà bạn gặp phải có thể bạn sẽ tự nghĩ ra cách giải quyết nhưng có thể nó chưa phải là tối ưu. Design Pattern giúp bạn giải quyết vấn đề một cách tối ưu nhất, cung cấp cho bạn các giải pháp trong lập trình OOP.

Design Patterns không phải là ngôn ngữ cụ thể nào cả. Nó có thể thực hiện được ở phần lớn các ngôn ngữ lập trình, chẳng hạn như Java, C#, thậm chí là Javascript hay bất kỳ ngôn ngữ lập trình nào khác.

Mỗi pattern mô tả một vấn đề xảy ra lặp đi lặp lại, và trình bày trọng tâm của giải pháp cho vấn đề đó, theo cách mà bạn có thể dùng đi dùng lại hàng triệu lần mà không cần phải suy nghĩ.

-design pattern chủ yếu được dựa theo những quy tắc sau đây về thiết kế hướng đối tượng.

* Lập trình cho interface chứ không phải để implement interface đó.
* Ưu tiên object composition hơn là thừa kế.

Phân loại:

Hệ thống các mẫu Design pattern hiện có **23 mẫu** được định nghĩa trong cuốn “Design patterns Elements of Reusable Object Oriented Software” và được chia thành 3 nhóm:

* [Creational Pattern](https://gpcoder.com/category/design-pattern/creational-pattern/) (nhóm khởi tạo – 5 mẫu) gồm: Factory Method, Abstract Factory, Builder, Prototype, Singleton. Những Design pattern loại này cung cấp một giải pháp để tạo ra các object và che giấu được logic của việc tạo ra nó, thay vì tạo ra object một cách trực tiếp bằng cách sử dụng method new. Điều này giúp cho chương trình trở nên mềm dẻo hơn trong việc quyết định object nào cần được tạo ra trong những tình huống được đưa ra.
* [Structural Pattern](https://gpcoder.com/category/design-pattern/structuaral-pattern/) (nhóm cấu trúc – 7 mẫu) gồm: Adapter, Bridge, Composite, Decorator, Facade, Flyweight và Proxy. Những Design pattern loại này liên quan tới class và các thành phần của object. Nó dùng để thiết lập, định nghĩa quan hệ giữa các đối tượng.
* [Behavioral Pattern](https://gpcoder.com/category/design-pattern/behavior-pattern/) (nhóm tương tác/ hành vi – 11 mẫu) gồm: Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy và Visitor. Nhóm này dùng trong thực hiện các hành vi của đối tượng, sự giao tiếp giữa các object với nhau.

**Builder patern**

**Builder Pattern giải quyết vấn đề gì?**

Tất cả chúng ta đều biết, ở ngôn ngữ Java, trong mỗi lớp đều có những hàm *constructor* và nếu chúng không được khai báo thì trình biên dịch sẽ tự động xây dựng một hàm *constructor* mặc định cho lớp ấy.

Khi chúng ta xây dựng, chúng ta cũng thể tùy ý lựa chọn những tham số truyền vào cho *constructor* miễn là nó hữu ích cho chúng ta.

Như vậy, vấn đề đặt ra ở đây khi một đối tượng của chúng ta có thể được khởi tạo với rất nhiều những tham số truyền vào và có thể một vài trong số đó không nhất thiết phải truyền vào trong lúc khởi tạo(có thể có hoặc không).

Không sao cả, chúng ta có thể dễ dàng giải quyết bằng cách truyền những giá trị null vào tham số trong hàm constructor hoặc là nạp chồng nhiều hàm constructor khác với những tham số tùy ý. Khi đó lớp của chúng ta sẽ trông như này:

public Student(String id, String firstName, String lastName) {

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

}

public Student(String id, String currentClass, String address, String phone) {

this.id = id;

this.currentClass = currentClass;

this.address = address;

this.phone = phone;

}

Cách này ban đầu nghe có vẻ ổn nhưng sẽ có những nhược điểm như sau:

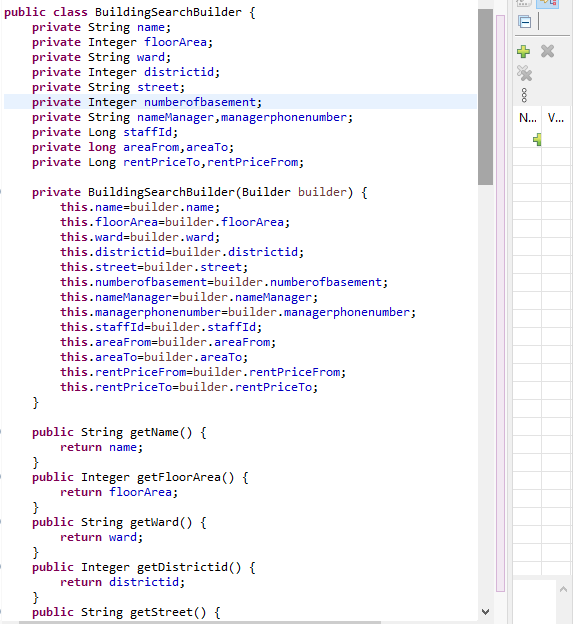
* Phải tạo rất nhiều hàm constructor trong những trường hợp khác nhau
* Khó khăn trong việc xác định thứ tự của những tham số truyền vào

Vấn đề này lại sinh ra vấn đề khác, vậy liệu **Builder Pattern** sẽ giải quyết vấn đề này như thế nào? Chúng ta cùng xem cấu tạo của **Builder Pattern** nhé!

Tôi có thể tóm gọn nó lại như sau : nó sẽ là kiểu 1 class lồng vào 1 class:

Tạm gọi class Lồng là class con

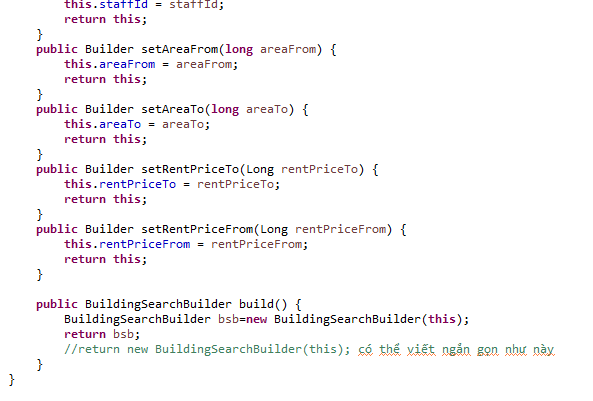
Class cha sẽ chứa các thuộc tính và các hàm getter + một hàm khởi tạo với tham số là biến mang kiểu dữ liệu class con:



Class con sẽ chứa các thuộc tính giống hệt class cha và chứa các setter + hàm gọi class cha:

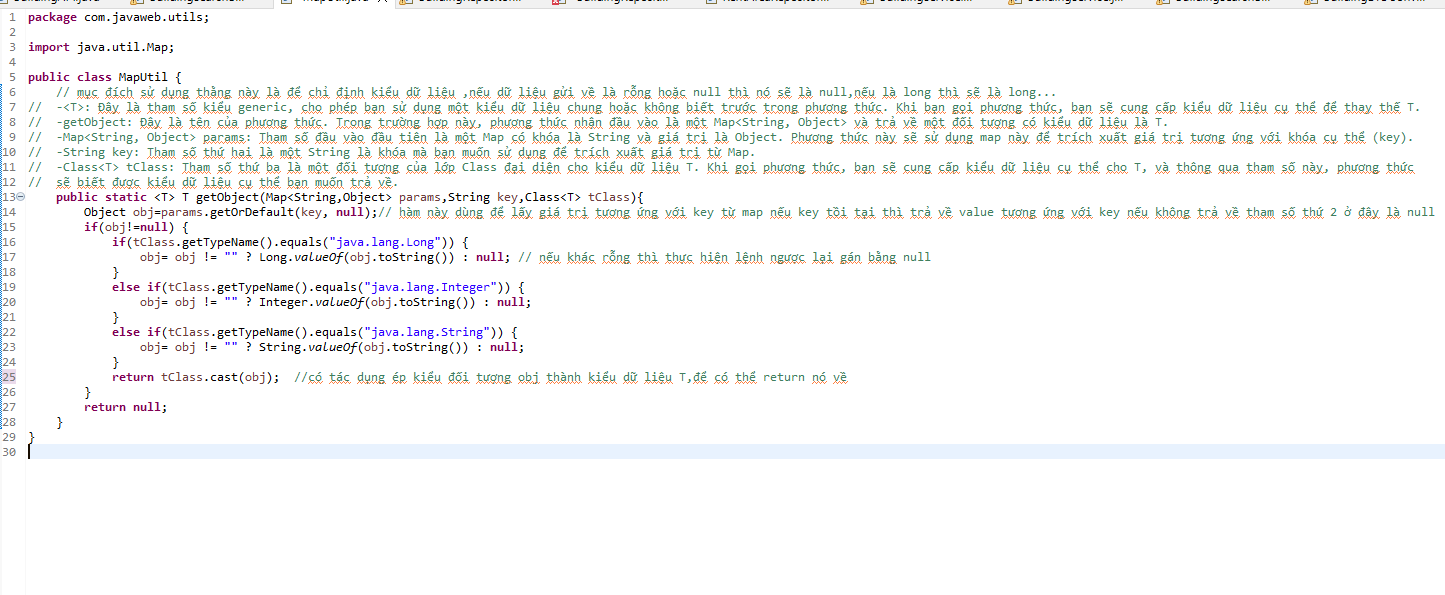


………………………………

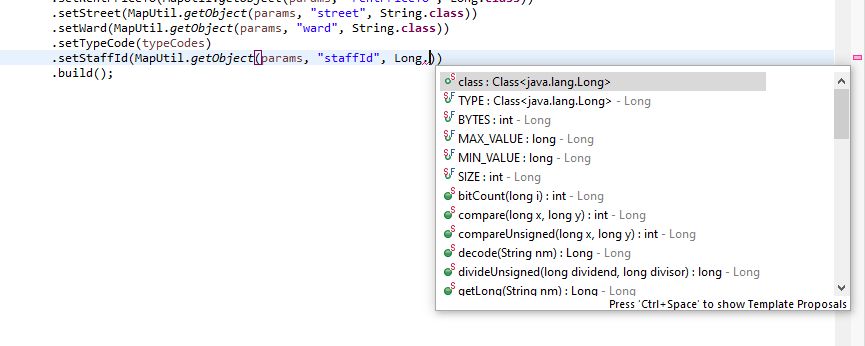


Thằng này còn một ưu điểm là ta sẽ có thể set những thuộc tính cần thiết cho contruct ter mà không phải xử lí như ví dụ trụ trên, và đặc biệt sẽ giúp cho code chúng ta clean hơn về chuyện tham số truyền vào vì khi mà ta muốn set thuộc tính cho một đối tượng thì phải truyền qua tham số của contructer giờ chả nhẽ có 16 thuộc tính ta lại phải truyền 16 tham số vào ? ở đây ta set là ở dưới class con rồi sau đó ném lên class cha bằng các truyền qua đối số chính bản thân class con vậy lúc này việc tham số truyền vào sẽ chỉ có duy nhất là 1 tham số giúp code chúng ta clean hơn

T\_Class



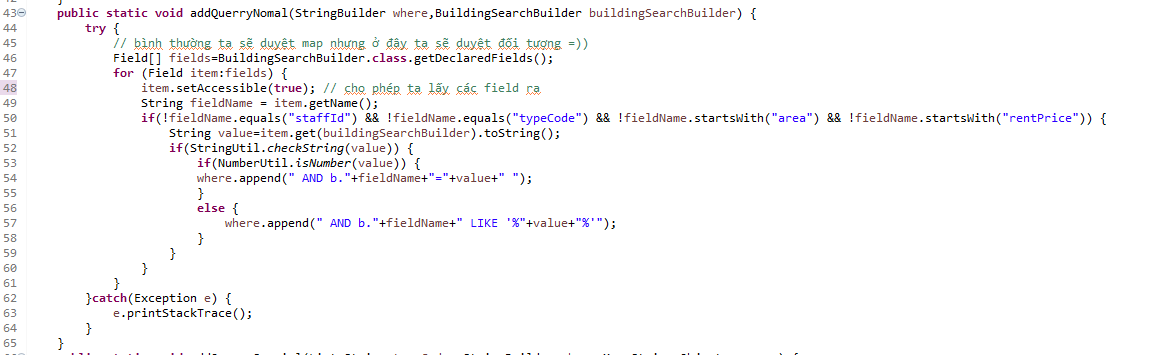
Giờ ta sẽ cùng phân tích :



Đối số thứ 3 yêu cầu kiểu dữ liệu là Class vậy nên khi truyền đối số ta phải sử dụng phương thức .class cho phép ta tạo một đối tượng thuộc lớp Class

Áp dụng Builder patern để thay cho map:

Sử dụng java reflection



-Dòng mã **Field[] fields = BuildingSearchBuilder.class.getDeclaredFields();** trong Java sử dụng reflection để lấy danh sách các trường (fields) khai báo trong lớp **BuildingSearchBuilder**

-Dòng mã **item.setAccessible(true);** sử dụng reflection để đặt quyền truy cập của trường được đại diện bởi biến **item** thành true có thể hiểu là mỗi lần ta dùng phương thức này ứng với item tương ứng nó sẽ cho phép thằng item ánh xạ vào các cái field của đối tượng đó

Trong Java, các trường (fields) của một lớp có thể có các mức quyền truy cập khác nhau như public, private, protected hoặc mức mặc định (package-private). Mặc định, Java giữ nguyên quyền truy cập này, tức là nếu một trường là private, bạn không thể truy cập trực tiếp nó từ bên ngoài lớp đó.

-Trong ngữ cảnh của Java, reflection thường được sử dụng để:

1. **Lấy Thông Tin Lớp (Class Information):** Bạn có thể lấy thông tin về các lớp trong thời gian chạy, chẳng hạn như tên lớp, các thành phần của lớp (fields, methods, constructors), v.v.
2. **Tạo Đối Tượng:** Bạn có thể tạo đối tượng mới của một lớp mà bạn chỉ biết tên của nó tại thời điểm chạy.
3. **Truy Cập và Thay Đổi Các Trường (Fields) và Phương Thức (Methods):** Cho phép truy cập và thậm chí là thay đổi giá trị của các trường và gọi các phương thức của một đối tượng mà không cần biết chúng tại thời điểm biên dịch.