ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

**VALIDATION FRAMEWORK**

**Nhóm 7:**

18127014 – Huỳnh Nhật Nam

18127223 – Nguyễn Phúc Thịnh

18127208 – Ngô Thanh Phương Thái

**Lớp:** 18CLC

**Môn : Design Patterns**

Thành phố Hồ Chí Minh năm 2021

# MỤC LỤC

Contents

[MỤC LỤC 2](#_Toc92729804)

[SƠ ĐỒ LỚP 4](#_Toc92729805)

[1. Sơ đồ lớp: 4](#_Toc92729806)

[2. Giải thích: 6](#_Toc92729807)

[2.1. Các lớp validator: 6](#_Toc92729808)

[2.2. Các lớp Builder: 7](#_Toc92729809)

[2.3. Comparators: 7](#_Toc92729810)

[2.4. Các lớp Result: 7](#_Toc92729811)

[2.5. Các lớp Helpers: 7](#_Toc92729812)

[2.6. Annotations: 8](#_Toc92729813)

[CÁC MẪU THIẾT KẾ SỬ DỤNG 9](#_Toc92729814)

[1. Chain Of Responsibility: 9](#_Toc92729815)

[1.1. Sơ đồ lớp: 9](#_Toc92729816)

[1.2. Đoạn code 9](#_Toc92729817)

[1.3. Ý nghĩa: 10](#_Toc92729818)

[2. Builder: 11](#_Toc92729819)

[2.1. Sơ đồ lớp: 11](#_Toc92729820)

[2.2. Đoạn code: 11](#_Toc92729821)

[2.3. Ý nghĩa: 12](#_Toc92729822)

[3. Factory Method: 13](#_Toc92729823)

[3.1. Sơ đồ lớp: 13](#_Toc92729824)

[3.2. Đoạn code: 13](#_Toc92729825)

[3.3. Ý nghĩa: 15](#_Toc92729826)

[4. Strategy: 16](#_Toc92729827)

[4.1. Sơ đồ lớp: 16](#_Toc92729828)

[4.2. Đoạn code: 16](#_Toc92729829)

[4.3. Ý nghĩa: 17](#_Toc92729830)

[5. Template Method: 17](#_Toc92729831)

[5.1. Sơ đồ lớp: 17](#_Toc92729832)

[5.2. Đoạn code 17](#_Toc92729833)

[5.3. Ý nghĩa: 18](#_Toc92729834)

# SƠ ĐỒ LỚP

## A picture containing timeline Description automatically generatedSơ đồ lớp:

A screenshot of a computer

Description automatically generated with low confidence

## Giải thích:

### Các lớp validator:

#### Interface:

* **Validator:** là interface chung cho tất cả các validator. Sử dụng Chain Of Responsibility pattern. Tất cả các validator sẽ validate 1 object thuộc Type T (generic)

#### Các lớp cơ bản:

* **BaseValidator:** lớp validator cơ bản. Lớp này đã implement một số chức năng cơ bản của một validator như setNext(), hasNext(),… Tất cả các validator đều sử dụng lớp này
* **BuiltInValidator:** là lớp để xây dựng một lớp validator dễ dàng. Lớp này đã sử dụng template method để xử lý chung việc ghi nhận, trả về kết quả và truyền tiếp tới validator kế tiếp trong chuỗi. Các lớp Validator kế thừa lớp này chỉ cần implement một số hàm đơn giản để hoạt động

#### Các lớp xử lý một lớp phức tạp do người dùng tạo ra:

* **AbstractValidator:** là công cụ để người dùng xây dựng một validator cho một Object không thuộc các kiểu dữ liệu cơ bản của java như String, boolean, … Người dùng sẽ kế thừa class này để xây dựng một validator cho một Object. Class này có các hàm đã được cài sẵn để xây dựng một cách dễ dàng các luật cho các trường dữ liệu của Class cần được validated
* **Rule:** Về cơ bản, Rule là một validator của một lớp T để validate một trường (một biến) S của lớp đó. Do đó, nó sẽ có hàm để lấy giá trị của trường đó và một chuỗi validator cho trường đó. Khi Rule được yêu cầu validate một object của lớp T, nó sẽ lấy giá trị của trường S và truyền vào chuỗi validator.

#### Built-in validators: kế thừa BultInValidator

* **MinValidator:** So sánh giá trị đưa vào lớn hơn giá trị cho trước (bằng hoặc không bằng)
* **MaxValidator:** So sánh giá trị đưa vào nhỏ hơn giá trị cho trước (bằng hoặc không bằng)
* **EqualValidator:** So sánh giá trị đưa vào bằng giá trị cho trước
* **NotNullValidator:** Kiểm tra giá trị đưa vào khác null
* **NotEmptyValidator:** Kiểm tra giá trị string đưa vào không rỗng
* **RegexValidator:** Kiểm tra giá trị string đưa vào trùng với regex
* **AssertTrueValidator:** Kiểm tra giá trị boolean đưa vào true
* **AssertFalseValidator:** Kiểm tra giá trị boolean đưa vào false

#### Các lớp validator xử lý Annotation:

* **AnnotationValidator:** validate một class theo ràng buộc dữ liệu (annotation). Khi validate một object lần đầu sẽ tạo chain AnnotatedFieldValidator cho từng trường của class đó
* **AnnotatedFieldValidator:** validate một trường của class theo ràng buộc dữ liệu. sử dụng builder tùy theo field

### Các lớp Builder:

#### Interface:

* **ChainValidatorBuilder:** 
  + hàm build trả về validator đầu trong chuỗi
  + Hàm processAnnotatedField để build một chain validator từ các annotation.

#### Lớp cơ bản:

* **BaseChainValidatorBuilder:** builder cơ bản. Gồm các hàm có thể được sử dụng lại bởi các ChainValidatorBuilder khác.

#### Các lớp Builder cho các kiểu dữ liệu:

* **StringChainValidatorBuilder***:* các hàm để xây dựng chuỗi validator cho kiểu String
* **NumericChainValidatorBuilder:** các hàm để xây dựng chuỗi validator cho kiểu numeric (Integer, Short, Long, Double, Float, Byte )
* **BooleanChainValidatorBuilder**: các hàm để xây dựng chuỗi validator cho kiểu Boolean
* **DateChainValidatorBuilder:** các hàm để xây dựng chuỗi validator cho kiểu Date
* **CustomChainValidatorBuilder:** các hàm để xây dựng chuỗi validator cho kiểu dữ liệu người dùng tạo ra

### Comparators:

#### Interface:

* **Comparator<T1, T2>:** so sánh biến t1 thuộc kiểu T1 và biến t2 thuộc kiểu T2

#### Các lớp:

* **StringIntComparator**: so sánh độ dài của một String
* **IntegerComparator**: so sánh 2 Integer
* **ShortComparator**: so sánh 2 Short
* **LongComparator**: so sánh 2 Long
* **DoubleComparator**: so sánh 2 Double
* **FloatComparator**: so sánh 2 Float
* **ByteComparator**: so sánh 2 Byte
* **StringComparator**: so sánh 2 String
* **DateComparator**: so sánh 2 Date

### Các lớp Result:

* Lớp Enum **RESULT**: enum cho kết quả OK hoặc Failed
* **ValidationResult**: gồm kết quả, tên (của validator trả về kết quả), và failed message
* **ValidationResults**: Lớp bao bên ngoài của ValidationResult array list. Gồm các hàm để tương tác với list.

### Các lớp Helpers:

* **ChainValidatorLinker:** dùng để nối các validator lại với nhau trong 1 chuỗi
* **ClassUtils:** một số hàm dùng cho các kiểu dữ liệu như parse String về một kiểu dữ liệu khác,…
* **FieldValueParser:** lấy giá trị của 1 field trong object.

### Annotations:

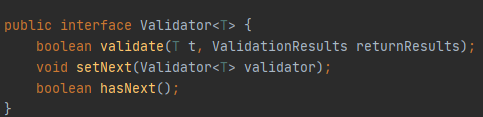
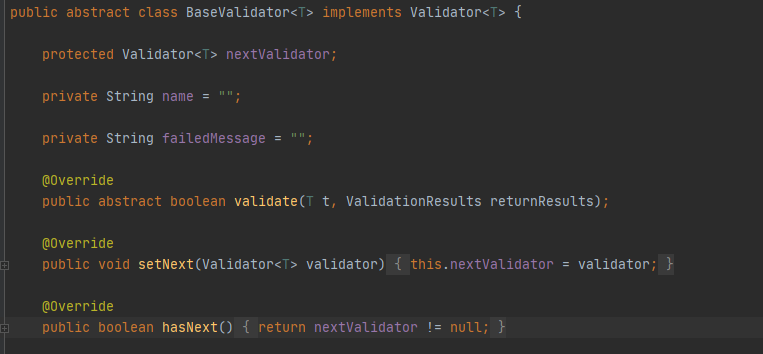
* **NotNull:** validate field khác null
* **NotEmpty:** validate field kiểu String không trống
* **Nested:** validate field bằng các annotation của chính field đó (dùng cho các kiểu dữ liệu do người dùng tạo ra đã được annotated)
* **Email:** validate field kiểu string là 1 email
* **Min:** validate field phải có giá trị min là một giá trị nhất định (giá trị này parse từ kiểu string qua kiểu dữ liệu của field)
* **Max:** validate field phải có giá trị max là một giá trị nhất định (giá trị này parse từ kiểu string qua kiểu dữ liệu của field)
* **AssertTrue:** validate field kiểu boolean là true
* **AssertFalse:** validate field kiểu boolean là false
* **Regex:** validate field kiểu string trùng regular expression
* **Equal:** validate field phải có giá trị bằng một giá trị nhất định (giá trị này parse từ kiểu string qua kiểu dữ liệu của field)
* **ValidatedBy:** sử dụng để tạo custom annotation.

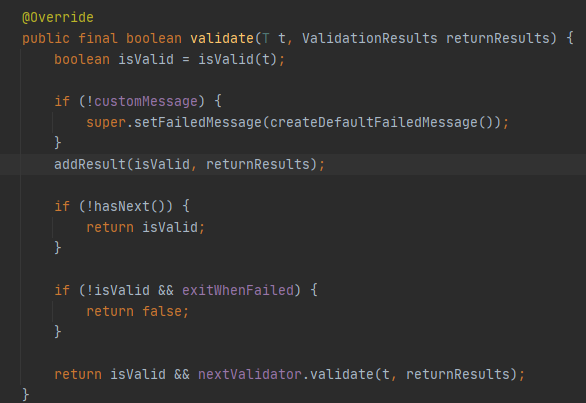
# CÁC MẪU THIẾT KẾ SỬ DỤNG

## Chain Of Responsibility:

### Sơ đồ lớp:

### Đoạn code

* Interface cho validator
* BaseValidator implement 1 số hàm như setNext, getNext và chứa nextValidator
* BuiltInValidator xử lý và truyền dữ liệu tiếp cho validator tiếp theo trong chuỗi

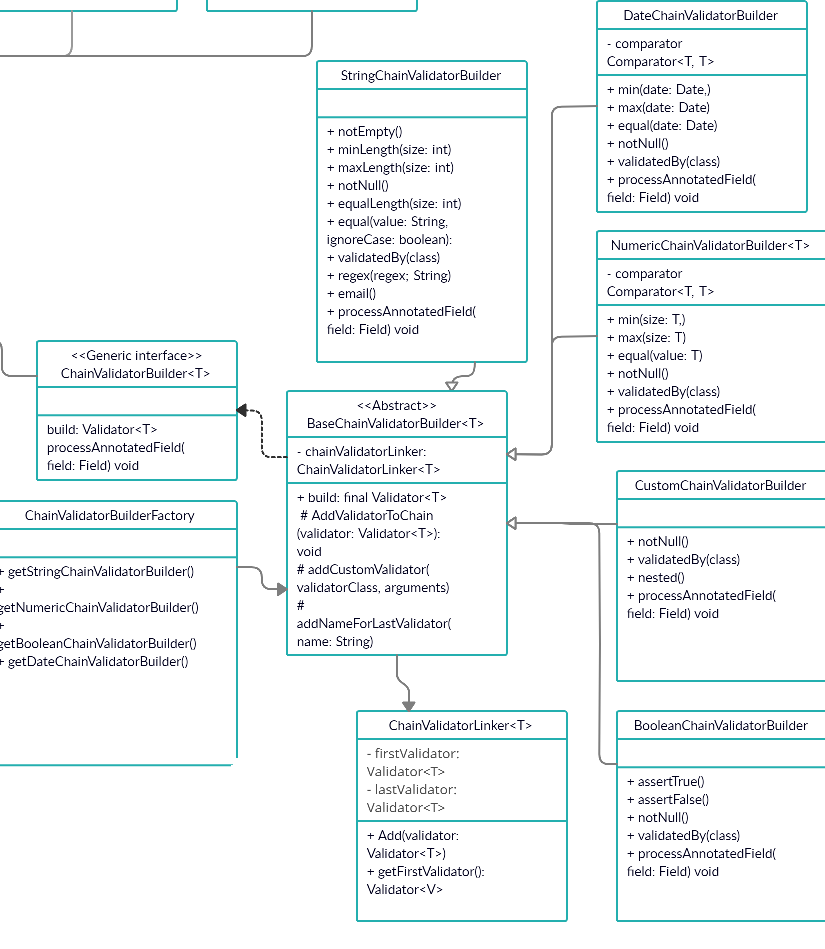


### Ý nghĩa:

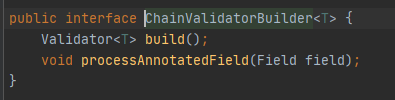
* Sử dụng mẫu chain of responsibility để ta có thể liên kết bất kì số lượng validator có thứ tự cho một kiểu dữ liệu ở runtime.
* Mỗi validator không cần biết về chuỗi mà chỉ cần thực hiện validate, ghi kết quả và quyết định việc nên truyền cho validator tiếp theo hay nên terminate chuỗi khi việc validate thất bại.Điều này khiến cho việc thêm một validator dễ dàng hơn rất nhiều và sẽ không ảnh hưởng tới các validator khác đã được cài đặt.

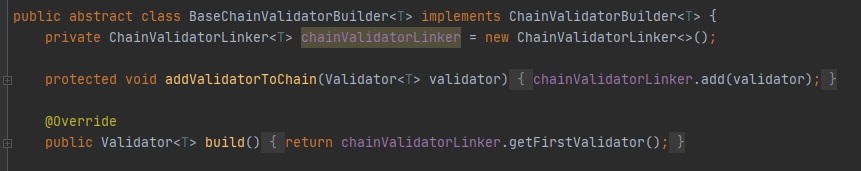
## Builder:

### Sơ đồ lớp:



### Đoạn code:

* Builder interface:
* Base builder:



Ví dụ 1 loại builder là NumericValidatorBuilder:

### Ý nghĩa:

* Sử dụng Builder, ta có thể thêm vào chuỗi từng validator một. Và nhờ đó, ta có thể tạo chuỗi các validator phức tạp một cách hợp lý và tuần tự từng bước
* Builder giúp tạo nhiều kiểu chuỗi Validator khác nhau cho cùng một loại kiểu dữ liệu

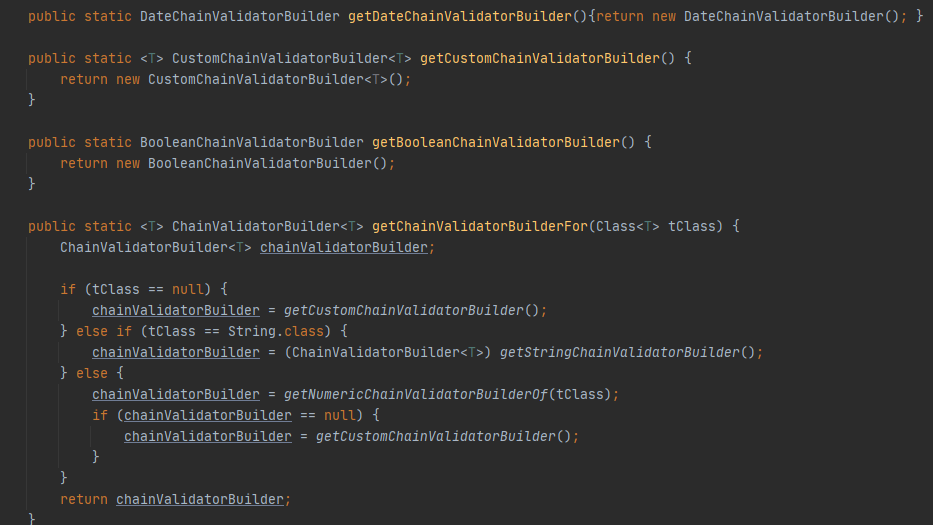
## Factory Method:

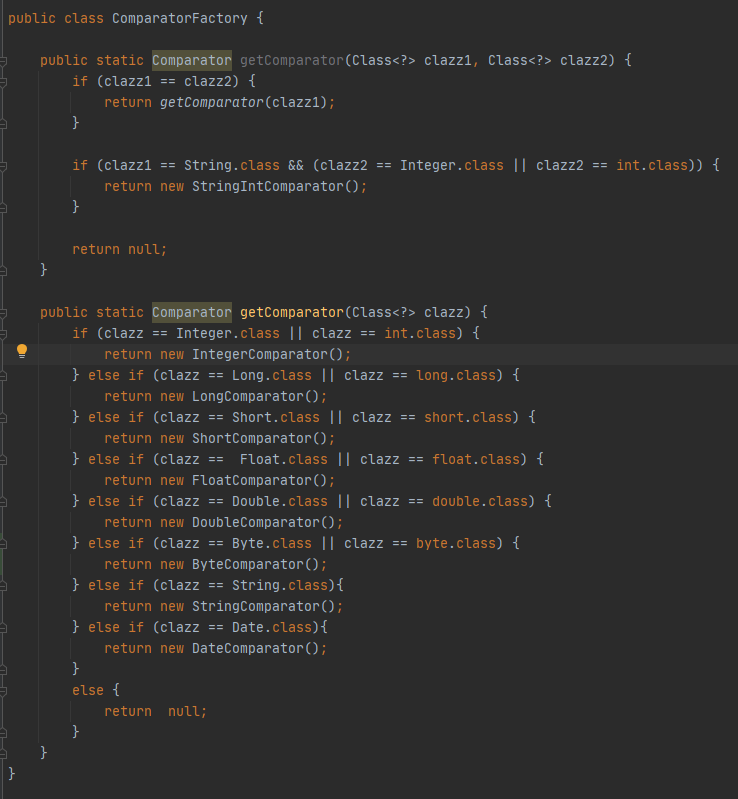
### Sơ đồ lớp:

### Đoạn code:

* ChainValidatorBuilderFactory:





* ComparatorFactory:
* 

### Ý nghĩa:

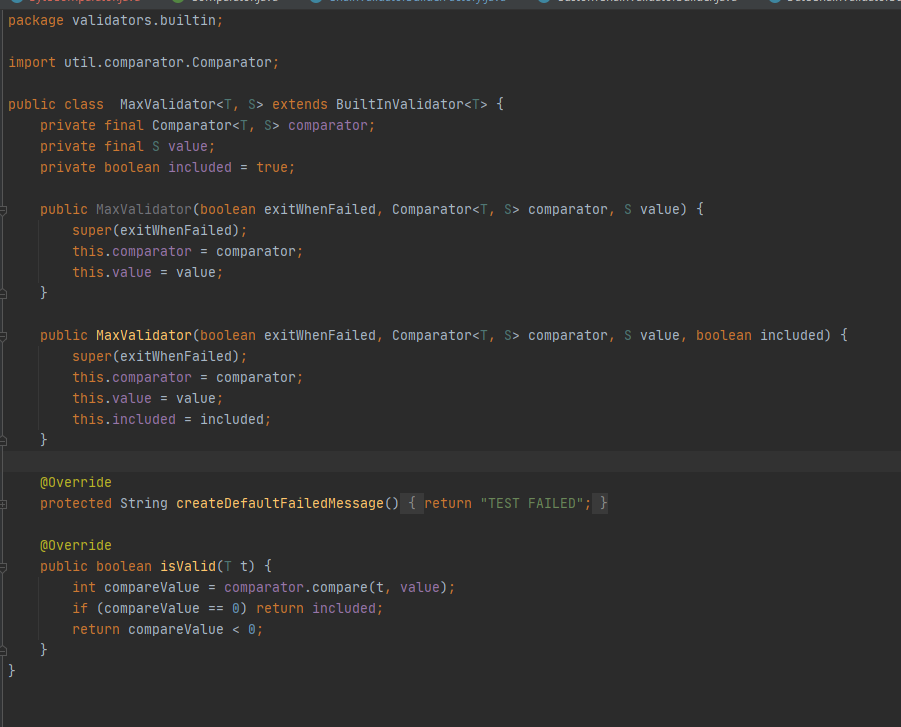
* Sử dụng Factory cho các lớp builder và comparator tách logic tạo các builder và comparator ra khỏi các class khác và tập trung nó trong cùng 1 class. Khi đó ta sẽ kiểm soát việc tạo ra các lớp builder hoặc comparator một cách dễ dàng hơn. Và nếu ta muốn thêm một lớp Builder hoặc Comparator sẽ không ảnh hưởng tới các class đã cài đặt ngoài Factory

## Strategy:

### Sơ đồ lớp:

### Đoạn code:

* Ở ví dụ MaxValidator dưới đây, Comparator là một Strategy

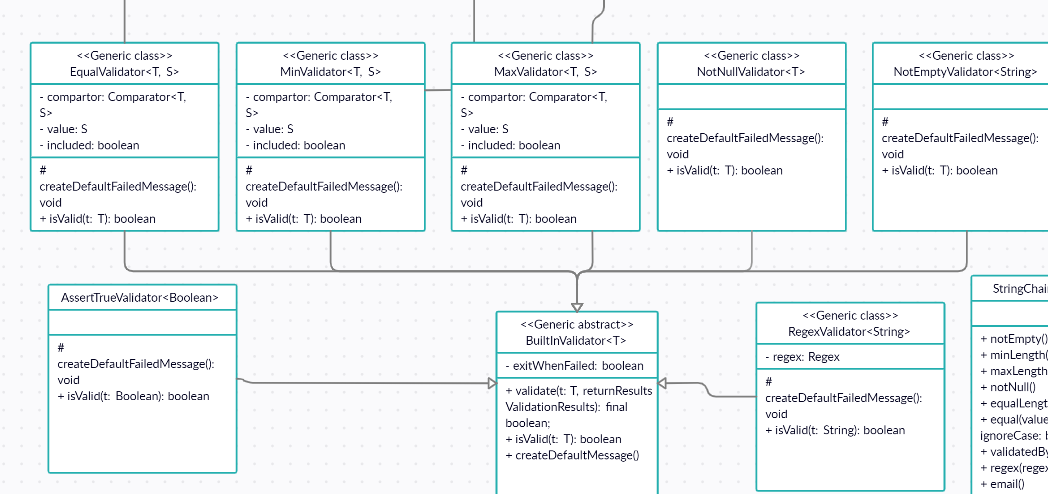


### Ý nghĩa:

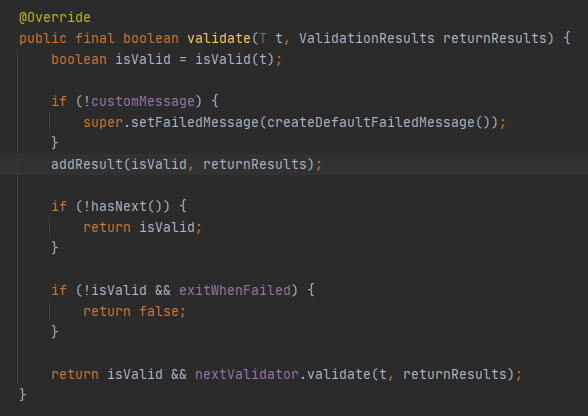
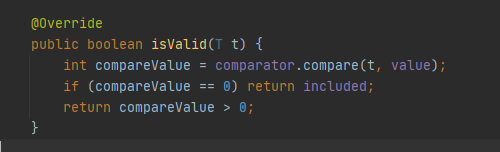
* Bằng cách sử dụng Strategy pattern, các lớp Validator như MinValidator, MaxValidator không cần biết logic đằng sau việc compare các giá trị. Ta chỉ cần thay đổi Comparator là có thể validate cho nhiều kiểu dữ liệu khác nhau mà không cần thay đổi MinValidator,...

## Template Method:

### Sơ đồ lớp:



### Đoạn code

* Abstract class BuiltInValidator:
* Ví dụ: MinValidator:

### Ý nghĩa:

* BuiltInValidator sử dụng Template method để chắc chắn các Validator kế thừa từ nó sẽ theo một logic đã được cài đặt sẵn. Việc xử lý và truyền dữ liệu tới validator tiếp theo đã được cài đặt và các lớp kế thừa lớp này chỉ cần xử lý việc dữ liệu có hợp lệ hay không.
* Sử dụng pattern này cũng làm giảm việc duplicate code giữa các Validator kế thừa lớp BuiltInValidator.