ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

**HƯỚNG DẪN SỬ DỤNG**

**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

Contents

[DEMO APP 3](#_Toc92998174)

[Cài đặt Validation Framework để sử dụng 4](#_Toc92998175)

[Một số lưu ý 5](#_Toc92998176)

[1. Các class tự định nghĩa: 5](#_Toc92998177)

[2. Kết quả validate: 5](#_Toc92998178)

[3. Annotation: 5](#_Toc92998179)

[Validate một object 6](#_Toc92998180)

[1. Tạo ràng buộc bằng code: 6](#_Toc92998181)

[1.1. Validate một object thuộc class tự định nghĩa: 6](#_Toc92998182)

[1.2. Validate một object thuộc các kiểu dữ liệu cơ bản của java: 6](#_Toc92998183)

[2. Tạo ràng buộc bằng annotation: (Xem mục ANNOTATIONS) 6](#_Toc92998184)

[3. Tạo Custom validator: 6](#_Toc92998185)

[ChainValidatorBuilder cho các kiểu dữ liệu cơ bản của Java 8](#_Toc92998186)

[1. Khởi tạo builder sử dụng Factory: 8](#_Toc92998187)

[2. Các loại builder: 8](#_Toc92998188)

[ANNOTATIONS 9](#_Toc92998189)

[1. Các loại annotation cho field: 9](#_Toc92998190)

[2. Tạo custom annotation: 10](#_Toc92998191)

# DEMO APP

Link github: <https://github.com/thinhnguyenphuc/DemoAppValidation>

# Cài đặt Validation Framework để sử dụng

* Validation Framework là một library riêng biệt có thể được cài đặt trong các phần mềm sử dụng Java.
* Cách cài đặt file jar như là library cho một số IDEA:
  + Intelliji IDEA: <https://www.jetbrains.com/help/idea/library.html>
  + Eclipse IDE: <https://stackoverflow.com/a/23420543>

# Một số lưu ý

## Các class tự định nghĩa:

* Bất kì field nào cần validate của class đó, field đó hoặc là public hoặc là có phương thức getter mới có thể validate được

## Kết quả validate:

* Có 2 class là ValidationResult để lưu giữ kết quả của các validator và ValidationResults là class để tương tác với mảng các kết quả. ValidationResults gồm các hàm cần thiết để lấy các kết quả validate thất bại hoặc xóa kết quả ra khỏi mảng.
* bool validate(T object, ValidationResults returnResults) là phương thức chung để validate một object thuộc kiểu T (T là class nào tùy vào validator).
* returnResults sẽ không reset cho nên khi cần validate các object khác nhau mà cần phân biệt kết quả giữa các object đó. Cần gọi returnResults.clear() hoặc truyền vào returnResults khác.

## Annotation:

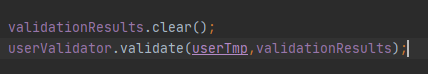
* Khi sử dụng annotation cho một class. AbstractValidator (xem phần dưới) sẽ không quan tâm tới các annotation của class đó. Tuy nhiên có thể sử dụng nested (xem phần dưới) để validate các annotation của một field của class đó.

# Validate một object

## Tạo ràng buộc bằng code:

### Validate một object thuộc class tự định nghĩa:

* B1: Tạo một lớp kế thừa (extends) AbstractValidator
* B2: Tạo ràng buộc cho từng field của class cần validate (trong AbstractValidator đã có sẵn các hàm cho các loại dữ liệu cơ bản như string, int,… và cho kiểu dữ liệu tự định nghĩa) 
* B3: gọi hàm validate với tham số là một object thuộc class cần validate



### Validate một object thuộc các kiểu dữ liệu cơ bản của java:

* Sử dụng ChainValidatorBuilder tùy theo kiểu dữ liệu (xem ChainValidatorBuilder cho các kiểu dữ liệu cơ bản của Java)

## Tạo ràng buộc bằng annotation: (Xem mục ANNOTATIONS)

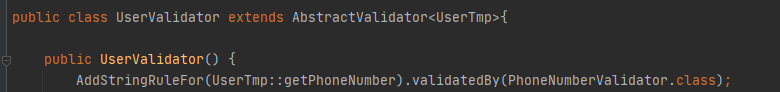
* B1: Đối với các field cần validate, thêm annotation cho các field đó
* B2:
  + Nếu chỉ cần validate một class nào đó, tạo một AnnotationValidator và gọi validate cho object của class đó.
  + Nếu class cần validate là một field của một class khác. Có thể sử dụng phương thức AddCustomRuleFor() của AbstractValidator sau đó gọi nested hoặc sử dụng annotation Nested

## Tạo Custom validator:

* B1: Tạo 1 class kế thừa 1 trong 3:
  + Validator<T> là interface cơ bản
  + BaseValidator<T> là class đã tạo 1 số hàm cơ bản như

**\* Lưu ý, vì các validator sẽ kết nối với nhau kiểu chuỗi (có thể hiểu là linked list 1 chiều) nên khi tạo custom validator kế thừa 2 class trên nên để ý tới việc gọi nextValidator.validate() nếu nextValidator khác null)**

* + BuiltInValidator<T> là class đã xử lý logic lấy kết quả và truyền tới validator tiếp theo. Chỉ cần cài đặt hàm isValid() trả về bool và hàm tạo message cơ bản. **Nên sử dụng class này**
* B2: Sau khi tạo CustomValidator. Các builder có hỗ trợ phương thức validatedBy() để thêm validator vào chuỗi các validator.



# ChainValidatorBuilder cho các kiểu dữ liệu cơ bản của Java

## Khởi tạo builder sử dụng Factory:

* Sử dụng ChainValidatorBuilderFactory. Class này có các hàm static để lấy builder tương ứng.

## Các loại builder:

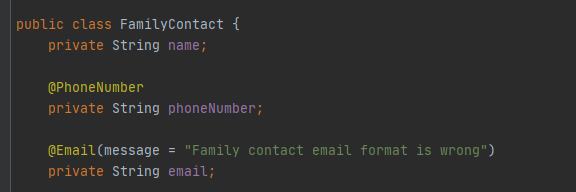
* StringChainValidatorBuilder
* DateChainValidatorBuilder
* BooleanChainValidatorBuilder
* CustomChainValidatorBuilder
* NumericChainValidatorBuilder: cho int, short, long, double, float, byte và các class tương ứng (Integer, Short,…)

# ANNOTATIONS

## Các loại annotation cho field:

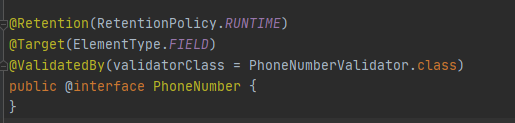
* **AssertTrue:** 
  + sử dụng cho kiểu bool.
  + Validate bool phải bằng true
* **AssertFalse:**
  + sử dụng cho kiểu bool.
  + Validate bool phải bằng false
* **Email:**
  + Sử dụng cho kiểu String
  + Validate String phải là 1 email
* **Min:**
  + Sử dụng cho kiểu Numeric, Date, String
  + Validate:
    - Độ dài string
    - Giá trị numeric
    - Giá trị Date
* **Max:**
  + Sử dụng cho kiểu Numeric, Date, String
  + Validate:
    - Độ dài string
    - Giá trị numeric
    - Giá trị Date
* **Equal:**
  + Sử dụng cho kiểu Numeric, Date
  + Validate:

Giá trị numeric

* + - Giá trị Date
* **EqualLength:**
  + Sử dụng cho kiểu String
  + Validate: Độ dài string
* **Nested:**
  + Sử dụng cho user defined classes
  + Validate theo annotation của class đó
* **NotEmpty**:
  + Sử dụng cho kiểu String
  + Validate string không được empty
* **NotNull:**
  + Sử dụng cho tất cả các kiểu:
  + Validate object khác null
* **Regex:**
  + Sử dụng cho kiểu String
  + Validate string đúng với regular expression
* Ví dụ: custom annotation PhoneNumber và annotation Email
* 

## Tạo custom annotation:

* **ValidatedBy:**
  + Sử dụng để tạo một custom annotation
* Ví dụ:



* + Retention và Target là bắt buộc phải có để có thể xử lý annotation
  + ValidatedBy với validatorClass là một class custom validator đã tạo
  + Sau đó sử dụng annotation này như sau, nó sẽ hoạt động bằng cách tạo một object validator thuộc validatorClass và gán nó vào chuỗi validator để validate field được annotated:

