**Giới thiệu**

Trước khi đi vào bài hôm nay, chúng ta cùng nghe cha đẻ của Null phát biểu :)))

Tôi gọi nó "sai lầm tỉ đô" 😂 hết

Đùa đấy, vẫn còn

null reference được tôi tạo ra năm 1965. vào thời điểm đó, tôi đã thiết kế tổng quan hệ thống tham chiếu dữ liệu cho ngôn ngữ lập trình hướng đối tượng. Mục tiêu của tôi là đảm bảo các tham chiếu tuyệt đối an toàn và được kiểm tra tự động bởi compiler. Nhưng,... Tôi đã không thể cưỡng lại được ham muốn đặt thêm thằng cu null vào hệ thống. Vì nó giúp implement dễ hơn :))) Điều này vô tình dẫn tới vô số lỗi, lỗ hổng và sự cố hệ thống, gây ra bao đau đớn và thương nhớ cho hàng triệu developer và cũng giúp nhiều công ty thiệt hại hàng tỉ dollar.

Những lời tâm sự muộn màng :(((

Chắc hẳn trong chúng ta ai cũng một lần bị ám ảnh bởi NullPointerException huhu. Mặc dù tới nay, các ngôn ngữ mới đều đã kiểm soát null để đảm bảo những dòng code được an toàn (Scala, Kotlin). Tuy nhiên, Java chưa nằm trong số đó :)))

Nhưng phòng còn hơn tránh, Java 8 ra đời cùng với một class mới tên là **Optional**. Nhiệm vụ của nó là kiểm soát null hộ chúng ta.

**Khái niệm Optional**

Optional<T> là một đối tượng Generic, nhiệm vụ chính của nó là **bọc** hay **wrapper** lấy một object khác. Nó chỉ chứa được một object duy nhất bên trong.

Việc bạn lấy giá trị của object bây giờ sẽ thông qua Optional và nếu object đó null cũng không sao, vì thằng Optional kiểm soát nó chặt chẽ hơn là if else.

Ví dụ bạn có một đối tượng bất kỳ:

String str = null;

// Tạo ra một đối tượng Optional

Optional<String> optional = Optional.ofNullable(str);

// Bây giờ Optional đã wrap lấy cái str.

Khi chúng ta thực hiện các thao tác, chúng ta có thể kiểm tra như thế này:

if (optional.isPresent()) {

System.out.println(opt.get()); // lấy ra cái str mình đã wrapper

}

Hmmm..... trông thế này thì khác đếch gì if (str != null) =))) Nhiều bạn sẽ tự nghĩ. Đúng là như vậy, nếu nó chỉ làm được đến đây, thì thôi.. nghỉ mịa đee huhu :((

Bây giờ mình sẽ giới thiệu từng tính năng lần lượt của Optional để bạn thấy nó kì diệu như nào.

**ifPresent**

optional.ifPresent(s -> System.out.println(s));

ifPresent nhận vào một Consumer, nó cũng chỉ là Functional Interface thôi các bạn. Nhận vào một đối tượng và thao tác trên nó, không return gì cả.

Nếu bạn chưa rõ Functional Interface và Lambda Expression thì bạn có thể xem ngay đây, dễ hiểu lém:

[Functional Interfaces & Lambda Expressions cực dễ hiểu](https://loda.me/java-8-functional-interfaces-and-lambda-expressions-cuc-de-hieu-loda1553649037877)

**orElse() và orElseGet()**

orElse() lấy ra object trong Optional. Nếu null, trả về giá trị mặc định do bạn quy định

String b = optional.orElse("Giá trị mặc định");

orElseGet() Tương tự orElse() nhưng trả ra bằng Supplier interface

String b = optional.orElseGet(() -> {

StringBuilder sb = new StringBuilder();

// Thao tác phức tạp

return sb.toString();

});

**map()**

map() giúp chúng ta biến đổi đối tượng bên trong Optional.

mình sẽ ví dụ bằng code dễ hiểu hơn.

class Outfit{

public String type;

public String getType() { return type; }

}

class Girl{

private Outfit outfit;

public Outfit getOutfit() { return outfit; }

}

public String getOutfitType(Girl girl){

return Optional.ofNullable(girl) // Tạo ra Optional wrap lấy girl

.map(Girl::getOutfit) // nếu girl != null thì lấy outfit ra xem kakaka :3 ngược lại trả ra Optional.empty()

.map(Outfit::getType) // nếu outfit != null thì lấy ra xem type của nó

.orElse("Không mặc gì"); // Nếu cuối cùng là Optional.empty() thì trả ra ngoài Không mặc gì.

}

code trông sáng sủa hơn nhiều phải không bạn :3

Trong code ở trên sử dụng Method reference, khái niệm này mình đã nói chi tiết tại đây:

[Hướng dẫn Method Reference và Lambda Expressions](https://loda.me/java-8-functional-interfaces-and-lambda-expressions-cuc-de-hieu-loda1553649037877)

Khái niệm map() mình có nói chi tiết tại đây:

[Stream Trong Java 8 cực dễ hiểu!](https://loda.me/java-8-huong-dan-stream-api-loda1553677531692)

**filter()**

filter() giúp chúng ta kiểm tra giá trị trong Optional nếu không thỏa mãn điều kiện, trả về empty()

public String getOutfitType(Girl girl){

return Optional.ofNullable(girl) // Tạo ra Optional wrap lấy girl

.map(Girl::getOutfit)

.map(Outfit::getType)

.filter(s -> s.contains("bikini")) // Nó chỉ chấp nhận giá trị bikini, còn lại dù khác null thì vẫn trả ra ngoài là Optiional.empty()

.orElse("Không mặc gì"); // Nếu cuối cùng là Optional.empty() thì trả ra ngoài "Không mặc gì".

Tới đây mình đã giới thiệu xong với các bạn các tính năng khá hay ho của Optional. Ngoài việc giúp chúng ta kiểm soát NullException thì còn giúp code của chúng ta sáng sủa hơn rất nhiều và thuận tiện hơn trong nhiều trường hợp yêu cầu điều kiện phức tạp