I, Functional Interface:

1, Functional Interface là gì?

* Trước java 8, functional inteface được gọi là Single Abstract Method Inteface (SAM interface), tức là interface mà chỉ có 1 abstract method duy nhất (không tính default method). Với mục đích sinh ra để nhằm implement một trong 5 nguyên lý của [SOLID](https://toidicodedao.com/2015/03/24/solid-la-gi-ap-dung-cac-nguyen-ly-solid-de-tro-thanh-lap-trinh-vien-code-cung/), đó chính là nguyên lý đầu tiên: Single responsibility principle (S) là mỗi một class/interface nên chỉ chứa 1 nhiệm vụ duy nhất.

2, Tại sao phải dùng functional Interface:

* nhằm implement một trong 5 nguyên lý của [SOLID](https://toidicodedao.com/2015/03/24/solid-la-gi-ap-dung-cac-nguyen-ly-solid-de-tro-thanh-lap-trinh-vien-code-cung/), đó chính là nguyên lý đầu tiên: Single responsibility principle (S) là mỗi một class/interface nên chỉ chứa 1 nhiệm vụ duy nhất.
* Để sử dụng [lambda expession](https://github.com/truongbb/java-8-tutorial/blob/master/02-lambda-expression.md), [method reference](https://github.com/truongbb/java-8-tutorial/blob/master/03-method-reference.md). -> code tối giản, tiện dụng hơn so với kĩ thuật anonymous class

3, Các trường hợp sử dụng Functional Interface

* Sử dụng khi mà inteface của ta dùng cho mục đích đơn (thực hiện 1 chức năng) hoặc

dùng để áp dụng lambda expression.

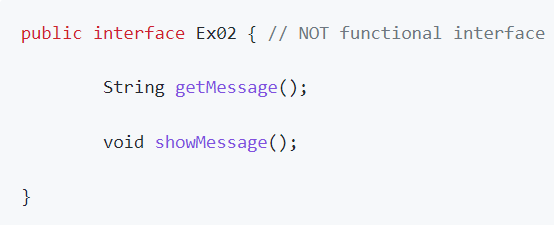
* Sử dụng các functional có sẵn trong package java.util.function

<https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html>

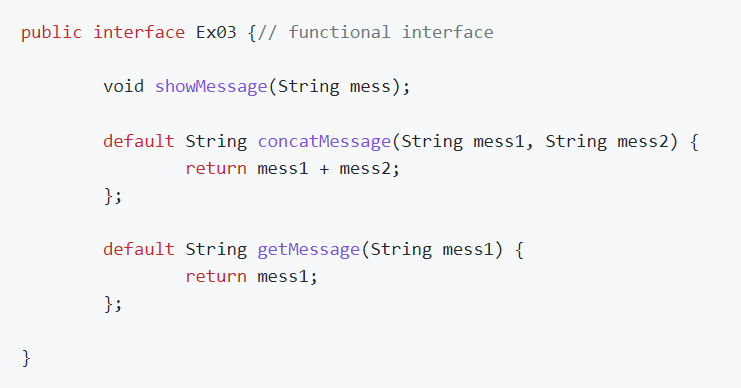
Một vài functional hay được sử dụng như: Consumer, Predicate,….

4, Khai báo chuẩn 1 functional interface:

* Nguyên tắc khai báo: 1 interface chỉ được gọi là 1 functional interface khi nó chỉ chứa duy nhất 1 Abstract Class ( Không tính default Method)
* 



Functional Inter Face chứa default method



* Mẹo: Sử dụng annotation @Functional Interface
* Ví dụ:
* Sử dụng Fuctional Interface kết hợp với Lambda Expresion đem lại sự tối giản, tiện dụng hơn so với sử dụng Anonymous Class

II, Lambda Expression

1, Lambda Expression là gì?

* Lambda expression là một tính năng mới và cực kì hot của java 8, cải tiến của cách viết anonymous class, nó được sử dụng với [functional interface](https://github.com/truongbb/java-8-tutorial/blob/master/01-functional-interface.md) tạo nên sự mạnh mẽ của java 8.
* Cách viết rút gọn của lambda expression là [method reference](https://github.com/truongbb/java-8-tutorial/blob/master/03-method-reference.md).

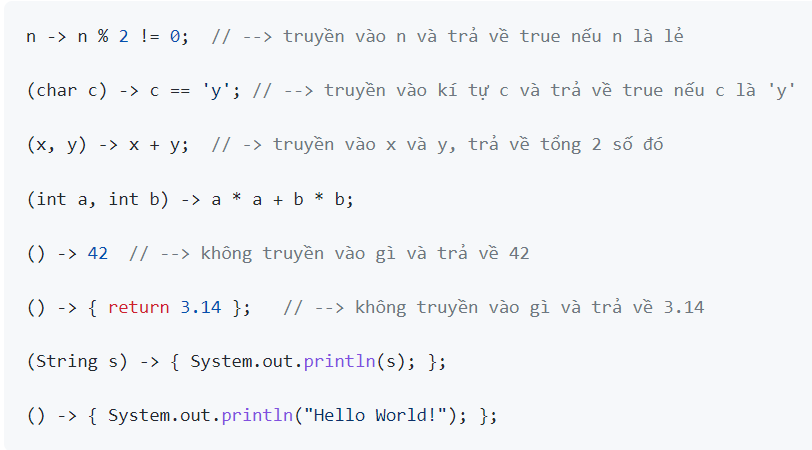
2, Tại sao nên sử dụng Lambda Expression?

* Sử dụng lambda expression để cho code ngắn gonj, dễ hiểu, trong sáng hơn.
* Đặt trường hợp nếu interface chỉ có 1 abstract method thì việc viết bắng anonymous class rất dài dòng, khó kiểm soát, sử dụng lambda expression cho code ngắn hơn nhưng hiệu năng tương tự.

3, Các trường hợp sử dụng Lambda Expression?

* Dùng bất cứ khi nào bạn muốn và được phép, cụ thể:
  + Sử dụng khi cài đè 1 hàm của một functional interface
  + Sử dụng khi làm việc với Collections và Stream (lọc, ánh xạ list, ...)
* Được sử dụng chủ yếu khi làm việc với Collections và Stream. ( vẫn là cách thể hiện của anonymous class thôi )
* Ví dụ:

4, Cách viết 1 Lambda expression:

* Cú pháp:
* Một số ví dụ sử dụng Lambda expression
* 

III, Method Reference

1,Method Reference là gì?

Chúng ta sử dụng [lambda expression](https://github.com/truongbb/java-8-tutorial/blob/master/02-lambda-expression.md) để tạo anonymous class, nhưng trong 1 vài ngữ cảnh, method trong anonymous class không thực hiện thêm bất cứ hành động nào mà chỉ sử dụng code có sẵn của hàm đã được viết khi ấy, [lambda expression](https://github.com/truongbb/java-8-tutorial/blob/master/02-lambda-expression.md) chỉ việc gọi, và như vậy, code sẽ cảm giác thừa, những lúc như thế, chúng ta sử dụng method reference.

2, Mục đích sử dụng method reference:

Method reference được sinh ra để giảm thiểu code, dễ đọc, dễ hiểu và dễ sử dụng khi cần (trường hợp nêu trên). ( gọn hơn cả Lambda Expression)

3, Các kiểu Method Reference

* Tham chiếu đến một static method: <Tên\_class>::<tên\_hàm\_Static\_tham\_chiếu\_tới>
* Tham chiếu tới một constructor: <Tên\_class>::new
* Tham chiếu tới một instance method của một đối tượng tùy ý của 1 kiểu cụ thể: <Tên\_class>::<tên\_instance\_method\_tham\_chiếu\_tới>
* Tham chiếu tới một instance method của một đối tượng tùy ý cụ thể: <Tên\_object>::<tên\_instance\_method\_tham\_chiếu\_tới>

4, Ví dụ:

IV, Stream

1, Stream là gì?

* Stream là 1 API của java 8 nằm trong package java.util.stream, gồm những API xử lý tuần tự các element cho collection
* Có 2 loại stream là tuần tự (stream) và song song (parallelStream)

2, Tại sao phải sử dụng Stream?

• Mục đích của stream là sử dụng để thao tác với các collection của java và thực hiện các tác vụ mà các câu truy vấn cơ sở dữ liệu đảm nhiệm. Ví dụ tính tổng, tìm số min, max, lọc danh sách theo tên, tìm kiếm, gom nhóm (group by) ... với câu lệnh ngắn nhưng hiệu năng cao

• Giảm thiểu code, tăng tính sử dụng, dễ đọc, dễ hiểu, dễ dùng.

3, Các trường hợp sử dụng Stream:

* Stream được sử dụng khi thao tác với Collection như list, map, set, ...

4, Sử dụng Stream với methods filter và map

Ví dụ:

5, Tạo mới 1 Stream:

* Tạo stream từ một collection, chúng ta chỉ cần sử dụng hàm stream() là được:

Ví dụ:

V, Optional

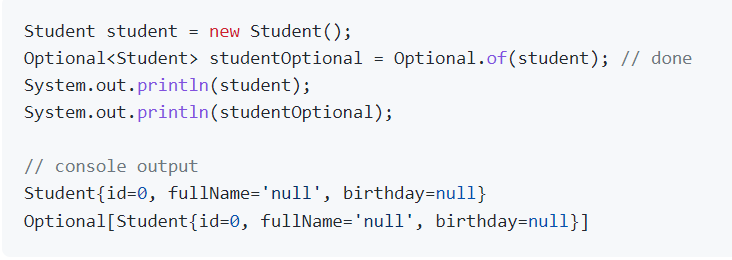
1, Opptional là gì?

**Optional là** 1 đối tượng generic, bản chất của **Optional là** một container (bao chứa) chứa đối tượng mà nó reference tới. Nó có thể rỗng hoặc chứa giá trị NULL (trường hợp đối tượng mà nó reference tới bị null). Cấu trúc này đảm bảo các biến được sử dụng thông qua **Optional** Class sẽ tránh được lỗi NullPointerException

2, Cách tạo 1 Optional

* Có nhiều cách để tạo 1 optional, tuy nhiên 3 cách cơ bản có thể đề cập tới:

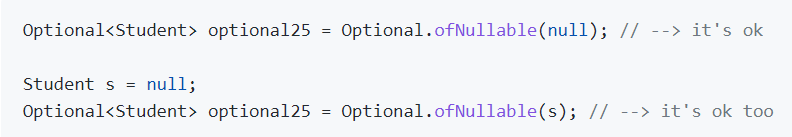
1.Sử dụng Optional.of(T)



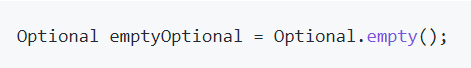
Nhưng khi truyền null để tạo thì chúng ta sẽ nhận lại NullPointerException khi sử dụng chúng:



2. Sử dụng Optional.ofNullable(null)



3. Sử dụng Optional.empty():Khi bạn chủ động muốn khởi tạo 1 Optinoal instace với giá trị bên trong nó là null mà không muốn truyền null hoặc object rỗng, java hỗ trợ sử dụng phương thức Optional.empty():



3, Các method của Optional

* Kiểm tra giá trị với isPresent(), isEmpty() ( chỉ có từ java 11 trở lên ) và ifPresent()
* orElse và orElseGet
* +orElse: Trả về giá trị của đối tượng Optional nếu có, ngược lại nó sẽ trả về đối tượng other mà bạn đã truyền vào phương thức này.
* +orElseGet: Trả về giá trị nếu tồn tại, ngược lại nó sẽ gọi other mà bạn đã truyền vào sau đó trả về kết quả của Supplier.

tài liệu tham khảo:

https://github.com/truongbb/java-8-tutorial/blob/master/05-optional.md