Ministry of SMEs and Startups × KOSME Korea SMEs and Startups Agency × LIKELION

Optional, Consumer, Supplier

Nguyễn Anh Tuấn

K TECH COLLEGE

Nội dung bài giảng

1 Optional<T>

2 Consumer<T> và Supplier<T>

3 Ứng dụng thực tế F-P-C-S

Optional<T>

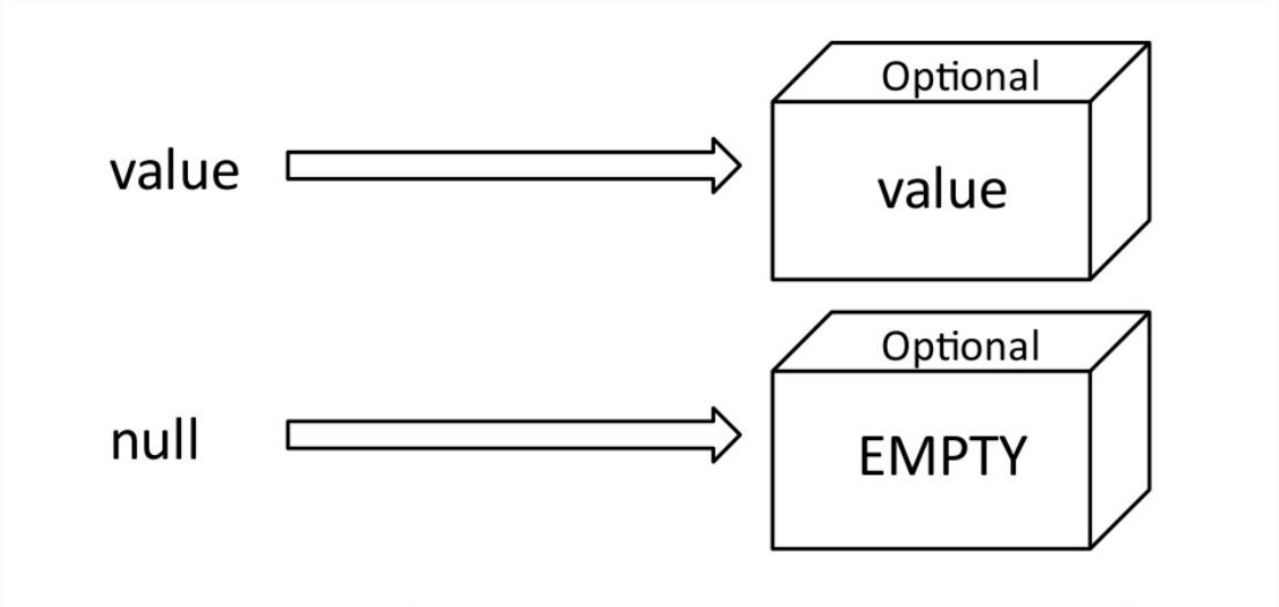
Optional<T>

- **Null**:
  - Null là một giá trị đặc biệt trong Java. Khi một biến đối tượng có giá trị null, nghĩa là nó không trỏ tới bất kỳ vùng nhớ nào của đối tượng.
  - Khi gọi 1 biến có giá trị null, chương trình sẽ báo lỗi **NullPointerException**.
  - Ví dụ 1 biến null: **String str = null;**

- **Empty**:
  - Một chuỗi rỗng khác với null, empty là 1 đối tượng hợp lệ trong bộ nhớ, nhưng không có ký tự nào trong đó.
  - Một chuỗi rỗng có độ dài (length) bằng 0.
  - Ví dụ 1 biến empty: **String str = “”;**

Optional<T>

- **Định nghĩa**:
  - Optional được sử dụng để xử lý với ngoại lệ NullPointerException.
  - Cung cấp các phương thức để kiểm tra biến hoặc đối tượng có phải là null hay không.



Optional<T>

- **Các phương thức thường được sử dụng trong Optional**:
  - Tạo đối tượng **Optional()**:
    - **empty()**: Tạo 1 đối tượng Optional rỗng.
    - **of()**: Tạo 1 đối tượng Optional non-null.
    - **ofNullable()**: Tạo 1 đối tượng Optional có thể chứa giá trị null hoặc non-null.
  - Kiểm tra trong **Optional()** có giá trị hay không:
    - **isPresent()**: Kiểm tra Optional có chứa giá trị hay không.
    - **ifPresent()**: Thực hiện 1 tác vụ nếu Optional có chứa giá trị.

Optional<T>

- **Các phương thức thường được sử dụng trong Optional**:
  - Lấy giá trị trong **Optional()**:
    - **get()**: Lấy giá trị nếu Optional có giá trị.
    - **orElse()**: Trả về giá trị nếu có, hoặc không sẽ trả về giá trị khác.
    - **orElseGet()**: Trả về giá trị nếu có, hoặc không sẽ trả về giá trị do **consumer** hoặc **supplier** cung cấp.
    - **orElseThrow()**: Trả về giá trị nếu có, hoặc không sẽ ném ra 1 ngoại lệ.

  - Một số phương thức khác:
    - **map()**: Chuyển đổi giá trị nếu có.
    - **filter()**: Lọc giá trị thoả mãn điều kiện.

Optional<T>

- **org.apache.common.lang3.StringUtils.isEmpty()**:
  - Mục đích của **isNullOrEmpty** cũng giống như **Optional**.
  - Tiện lợi cho việc kiểm tra null hoặc empty 1 cách nhanh chóng trong các tình huống tạm thời.

- **Optional**:
  - Cung cấp một cách tiếp cận có cấu trúc hơn để xử lý giá trị có thể là null, với nhiều phương thức tiện ích và một API rõ ràng.
  - Tránh được các lỗi NullPointerException mà không cần phải kiểm tra null thủ công mọi nơi.

Consumer<T>
Supplier<T>

Consumer<T>

Predicate<T>	⟶	boolean test(T t)
Consumer<T>	⟶	void accept(T t)
Function<T,R>	⟶	R apply(T t)
Supplier<T>	⟶	T get()

Consumer<T> và Supplier<T>

- **Định nghĩa**:
  - **Function** là một **functional interface** có phương thức **apply** ⇒ nhận vào một đối tượng thuộc kiểu T và trả về một đối tượng thuộc kiểu R
  - **Predicate** là một **functional interface** có phương thức **test** ⇒ nhận vào một đối tượng thuộc kiểu T và trả về một giá trị boolean.
  - **Consumer** là một **functional interface** có phương thức **accept** ⇒ nhận vào một đối tượng thuộc kiểu T và không trả về giá trị.
  - **Supplier** là một **functional interface** có phương thức **get** ⇒ không nhận tham số và trả về một đối tượng thuộc kiểu T.

Consumer<T>

```
public static void main(String[] args) {
        // Tạo Consumer interface kiểu cổ ngữ
        Consumer<String> consumer = new Consumer<String>() {
                @Override
                public void accept(String name) { System.out.println("Hello " + name); }
        };
        consumer.accept("KTC");

        // Tạo Consumer interface với lambda expression
        Consumer<String> consumer1 = (name) -> System.out.println("Hello " + name);
        consumer1.accept("KTC");

        // Tạo Consumer interface với method reference
        Consumer<String> consumer2 = System.out::println;
        consumer2.accept("Hello KTC");
}
```

Consumer<T>

- **Sử dụng phương thức andThen()**:

```
public static void main(String[] args) {
        int TEST_NUMBER = 5;
        Consumer<Integer> math1 = (e) -> System.out.println(e * 2);
        Consumer<Integer> math2 = (e) -> System.out.println(e * e);
        Consumer<Integer> math3 = (e) -> System.out.println(e % 2 == 1);

        // thực hiện phương thức accept của lần lượt 3 Consumer
        math1.accept(TEST_NUMBER);
        math2.accept(TEST_NUMBER);
        math3.accept(TEST_NUMBER);

        // thực hiện tuần tự Consumer
        Consumer<Integer> combineConsumer = math1.andThen(math2).andThen(math3);
        combineConsumer.accept(TEST_NUMBER);
}
```

Supplier<T>

```
public static void main(String[] args) {
        // Tạo Supplier interface kiểu cổ ngữ
        Supplier<String> supplier = new Supplier<String>() {
                @Override
                public String get() { return "Hello World"; }
        }
        System.out.println("Kieu co ngu: " + supplier.get());

        // Tạo Supplier interface với lambda expression
        Supplier<String> supplier = () -> "Hello";
        System.out.println("Kieu lambda expression: " + supplier.get());

        // Tạo Supplier interface với method reference
        Supplier<String> supplier = () -> "hello KTC";
        System.out.println("Kieu method reference: " + supplier.get());
}
```

Ứng dụng thực tế F-P-C-S

Ứng dụng Function, Predicate, Consumer và Supplier

- **Function**:
  - **Chuyển đổi dữ liệu**: Chuyển đổi một danh sách các đối tượng này thành một danh sách các đối tượng khác, ví dụ như chuyển đổi một danh sách các chuỗi thành danh sách các độ dài của chuỗi đó.
  - **Xử lý dữ liệu phức tạp**: Thực hiện các phép tính hoặc biến đổi phức tạp trên dữ liệu, ví dụ như tính toán giá trị sau thuế của một danh sách các sản phẩm.
  - **Lập trình hàm**: Kết hợp với các phương thức của Stream như map để thực hiện các chuyển đổi trên tập hợp dữ liệu một cách dễ dàng.

Ứng dụng Function, Predicate, Consumer và Supplier

- **Predicate**:
  - **Lọc dữ liệu**: Lọc các phần tử của một danh sách dựa trên một điều kiện nào đó, ví dụ như lọc ra các số chẵn trong một danh sách các số nguyên.
  - **Xác thực dữ liệu**: Kiểm tra tính hợp lệ của dữ liệu đầu vào, ví dụ như kiểm tra xem một chuỗi có phải là email hợp lệ hay không.
  - **Quản lý quyền truy cập**: Kiểm tra quyền truy cập của người dùng trong các ứng dụng bảo mật, ví dụ như kiểm tra xem người dùng có quyền truy cập vào một tài nguyên cụ thể hay không.

Ứng dụng Function, Predicate, Consumer và Supplier

- **Consumer**:
  - **Xử lý sự kiện**: Xử lý các sự kiện hoặc thực hiện các hành động trên các đối tượng, ví dụ như in ra thông tin của từng phần tử trong một danh sách.
  - **Thao tác trên đối tượng**: Thực hiện các thay đổi trên các đối tượng, ví dụ như cập nhật trạng thái của các đối tượng trong một danh sách.
  - **Lập trình hàm**: Kết hợp với các phương thức của Stream như forEach để thực hiện các hành động trên từng phần tử của tập hợp dữ liệu.

Ứng dụng Function, Predicate, Consumer và Supplier

- **Supplier**:
  - **Tạo đối tượng khi cần thiết**: Tạo đối tượng khi cần thiết, đặc biệt khi việc tạo đối tượng đó tốn kém tài nguyên, ví dụ như tạo kết nối cơ sở dữ liệu hoặc đối tượng cấu hình.
  - **Cung cấp giá trị mặc định hoặc thay thế**: Cung cấp các giá trị mặc định hoặc thay thế trong các tình huống đặc biệt, ví dụ như cung cấp giá trị mặc định nếu không có giá trị nào khác được cung cấp.
  - **Stream API**: Tạo các nguồn dữ liệu mới, ví dụ như tạo một stream vô hạn các số ngẫu nhiên hoặc các giá trị tính toán.

ROAD TO KOREA

Nếu có bất kỳ thắc mắc nào, hãy đặt câu hỏi qua

mail@mail.com hoặc Zalo 0xxx xxx xxx