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

Functional Interface

Nguyễn Anh Tuấn

K TECH COLLEGE

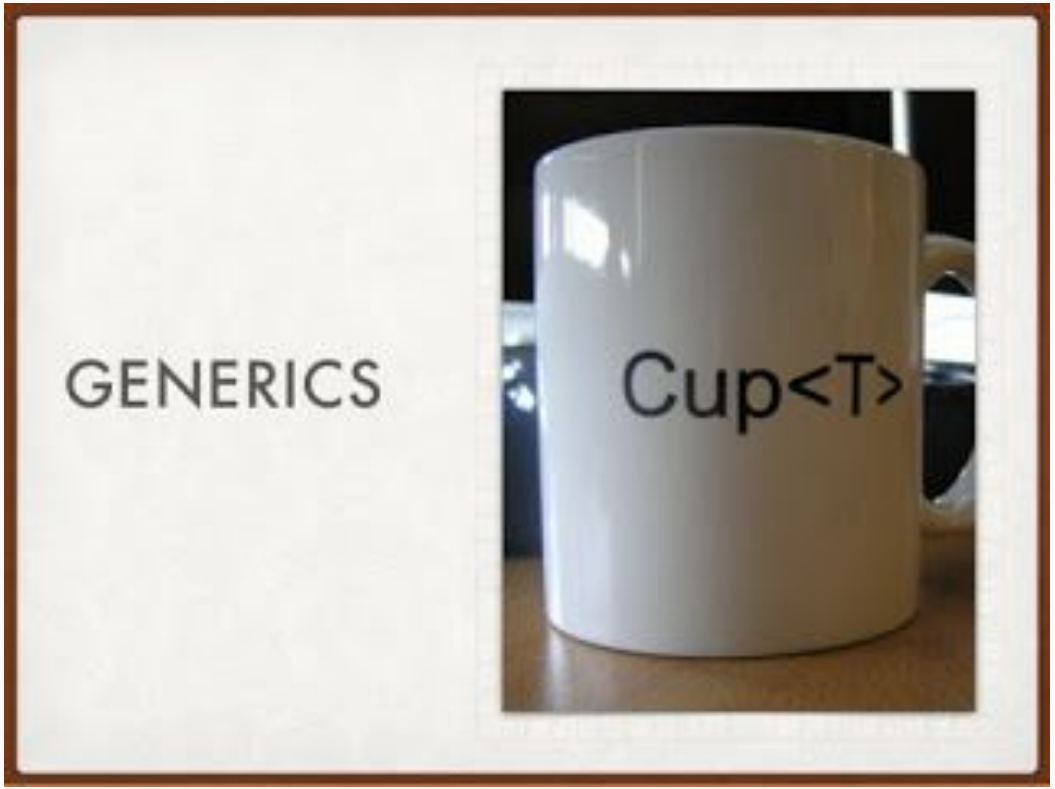
Nội dung bài giảng

1 Generic trong Java

2 Functional Interface

Generic trong Java

Generic trong Java



Class generic

```
// Sử dụng <> sau tên class để tạo ra 1 class generic
// Bên trong <> có thể để 1 chữ viết hoa bất kỳ
public class Dog<T> {
        T age;
        public Dog(T age1) {
                age = age1;
        }
}

public static void main(String[] args) {
        // Khai báo 1 đối tượng generic dog phải ghi
        // rõ kiểu dữ liệu mong muốn
        Dog<Integer> dog = new Dog(15);
        System.out.println(dog.age);
}
```

```
// Sử dụng <> sau tên class để tạo ra 1 class generic
// Bên trong <> có thể để 1 chữ viết hoa bất kỳ
public class Dog<T> {
        T age;
        public Dog(T age1) {
                age = age1;
        }
}

public static void main(String[] args) {
        // Khai báo 1 đối tượng generic dog phải ghi
        // rõ kiểu dữ liệu mong muốn
        Dog<String> dog = new Dog("muoi lam");
        System.out.println(dog.age);
}
```

Method generic

```
public class Dog {
        // <T> là kiểu dữ liệu giữ chỗ cho param act, nên act cũng có kiểu dữ liệu là T
        // T là kiểu dữ liệu trả về
        public <T> T action(T act) {
                return act;
        }
}

public static void main(String[] args) {
        Dog dog = new Dog();
        String actStr = dog.action(“nam xuong");
        int actInt = dog.action(123);
        System.out.println(actStr);
        System.out.println(actInt);
}
```

Interface generic

```
// Sử dụng <> sau tên class để tạo ra 1 class generic
public interface Father<T> { void doSomeThing(T action); }

// Khi ghi đè method doSomeThing, class Son cũng phải định nghĩa là class generic với kiểu dữ liệu
// chưa xác định là <T> để khi khởi tạo đối tượng Son, ta có thể quyết định kiểu dữ liệu cho T
public class Son<T> implements Father<T> {
        @Override
        public void doSomeThing(T action) { System.out.println(action); }
}

public static void main(String[] args) {
        Son<Integer> son = new Son<>();
        son.doSomeThing(123);

        Son<String> son1 = new Son<>();
        son1.doSomeThing("test");
}
```

Generic trong Java

- **Định nghĩa:**
  - **Generic** là một cơ chế cho phép các class, interface và method làm việc với các kiểu dữ liệu khác nhau mà không cần phải chỉ định cụ thể kiểu dữ liệu cho đến khi các class, interface và method đó được sử dụng
  - **Generic** được giới thiệu từ Java 5

- **Ý nghĩa:**
  - Generic giúp tránh việc sử dụng các kiểu dữ liệu cụ thể và cho phép tái sử dụng mã nguồn
  - Không cần ép kiểu dữ liệu từ Object sang kiểu dữ liệu cụ thể, tránh phát sinh lỗi khi ép kiểu
  - Generic giúp xác định và kiểm tra kiểu dữ liệu tại thời gian biên dịch, tránh các lỗi do không tương thích kiểu dữ liệu

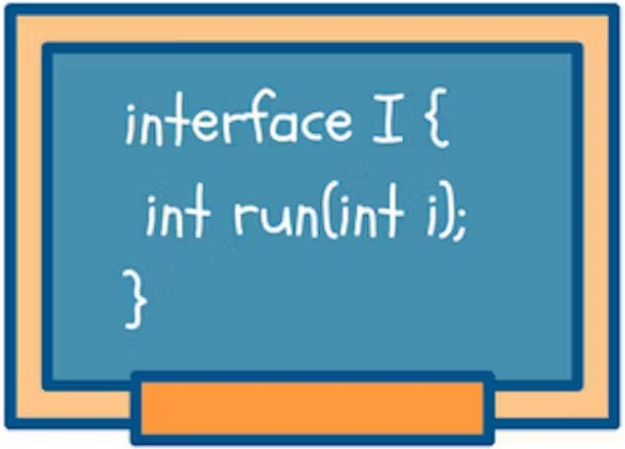
Functional interface

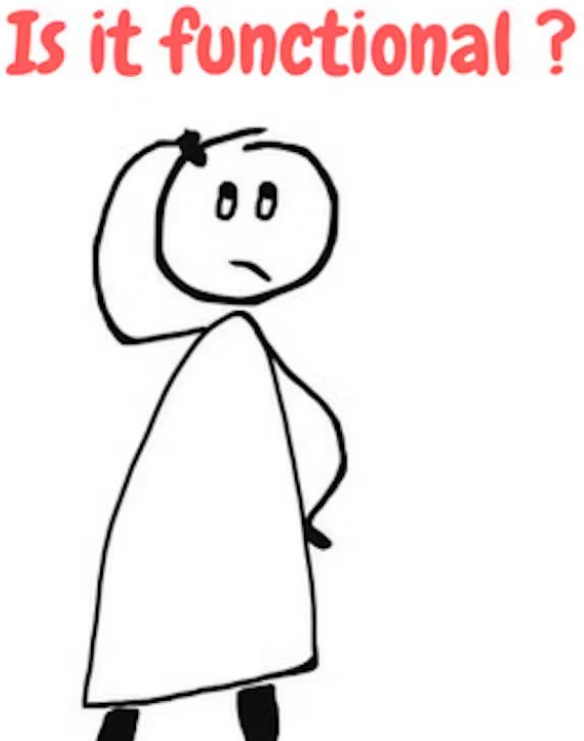
Function interface

- **Định nghĩa:**
  - **Functional interface** là interface có duy nhất 1 abstract method, có thể không có hoặc có nhiều default/static method hoặc các Object method (ở dạng abstract)
  - Functional interface có thể có các phương thức của lớp java.lang.Object
  - Annotation **@FunctionalInterface**: Được sử dụng để đánh dấu một interface là Functional Interface

- **Ý nghĩa:**
  - Giúp định nghĩa các hành vi chức năng rõ ràng
  - Tránh lặp lại code, chiếm nhiều tài nguyên hệ thống
  - Dễ dàng sử dụng trong các biểu thức lambda và method references

Function interface





Các Functional Interfaces có sẵn trong Java 8

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

Function<T, R>

- **Định nghĩa:**
  - Nhận một đối số kiểu T và trả về kết quả kiểu R, dùng để biến đổi một giá trị thành một giá trị khác

```
public static void main(String[] args) {
        // Khởi tạo Function interface để chuyển đổi kiểu dữ liệu
        Function<String, Integer> convert = new Function<String, Integer>() {
                @Override
                public Integer apply(String s) {
                        // Nhận đối số kiểu String, trả về kết quả kiểu Integer
                        return Integer.parseInt(s);
                }
        };
        System.out.println(convert.apply("123"));
}
```

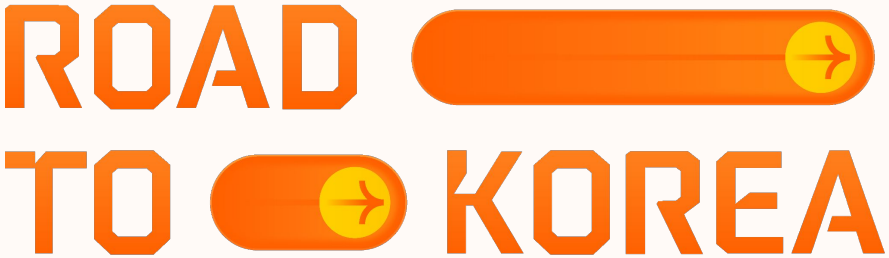
Function<T, R>

- **Định nghĩa:**
  - Nhận một đối số kiểu T và trả về kết quả kiểu boolean, dùng để check xem phần tử có thoả mãn điều kiện hay không

```
public static void main(String[] args) {
        // Khởi tạo Predicate interface kiểm tra kiểu dữ liệu có thoả mãn điều kiện không
        Predicate<String> convert = new Predicate<String>() {
                @Override
                public boolean test(String s) {
                        // Kiểm tra đối số có đúng kiểu String không
                        return true;
                }
        };
        System.out.println(convert.test("abc"));
}
```

Vậy thì Functional interface để làm cái gì?

Lợi ích chính của functional interface là chúng ta có thể sử dụng
Lambda Expression
để tạo ra thể hiện (instance) cho interface đó



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

mail@mail.com hoặc Zalo 0xxx xxx xxx