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

Lambda, Method references

Nguyễn Anh Tuấn

KTECH COLLEGE

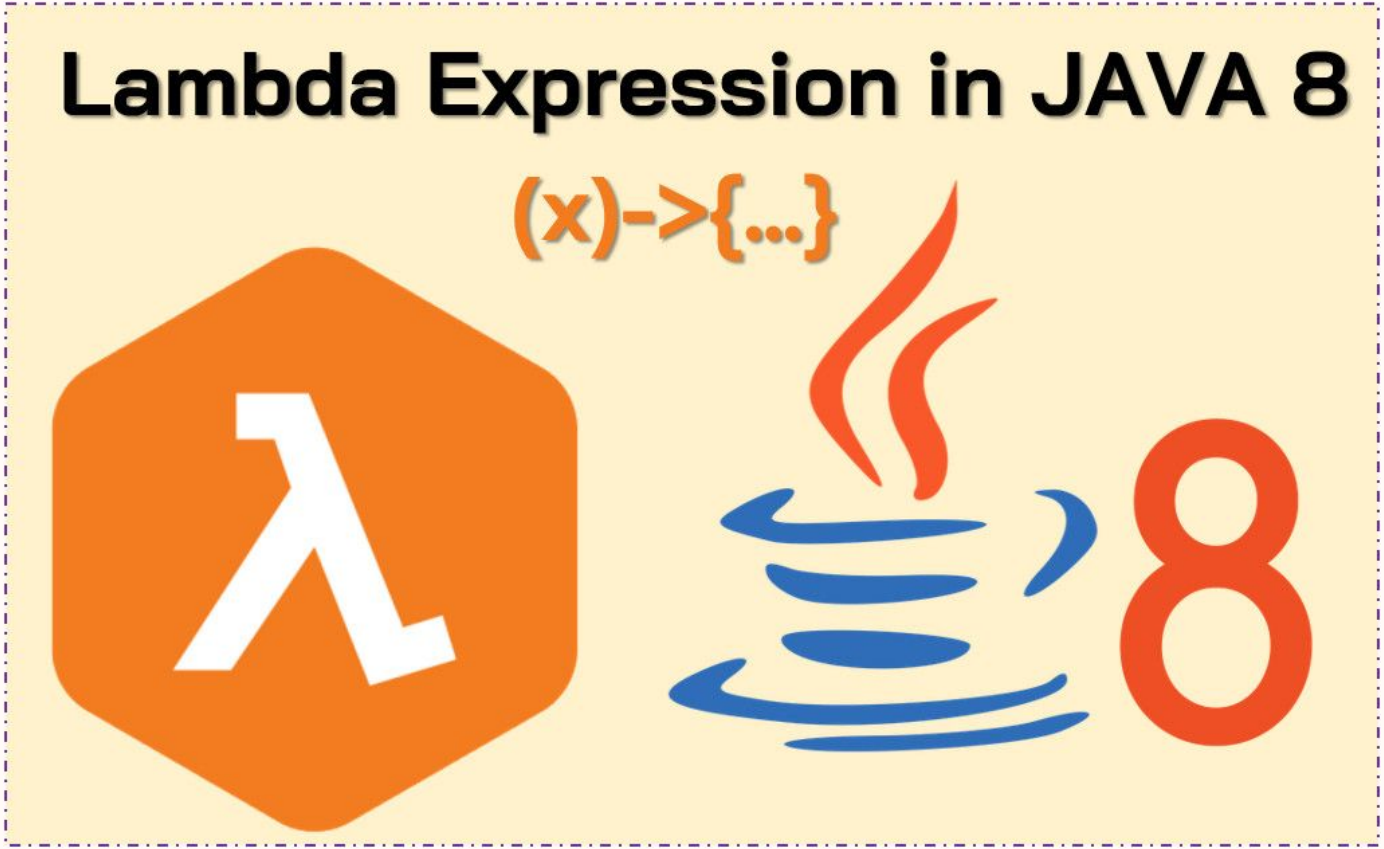
Nội dung bài giảng

1 Biểu thức Lambda

2 Method references

Biểu thức Lambda

Biểu thức Lambda - Lambda Expression

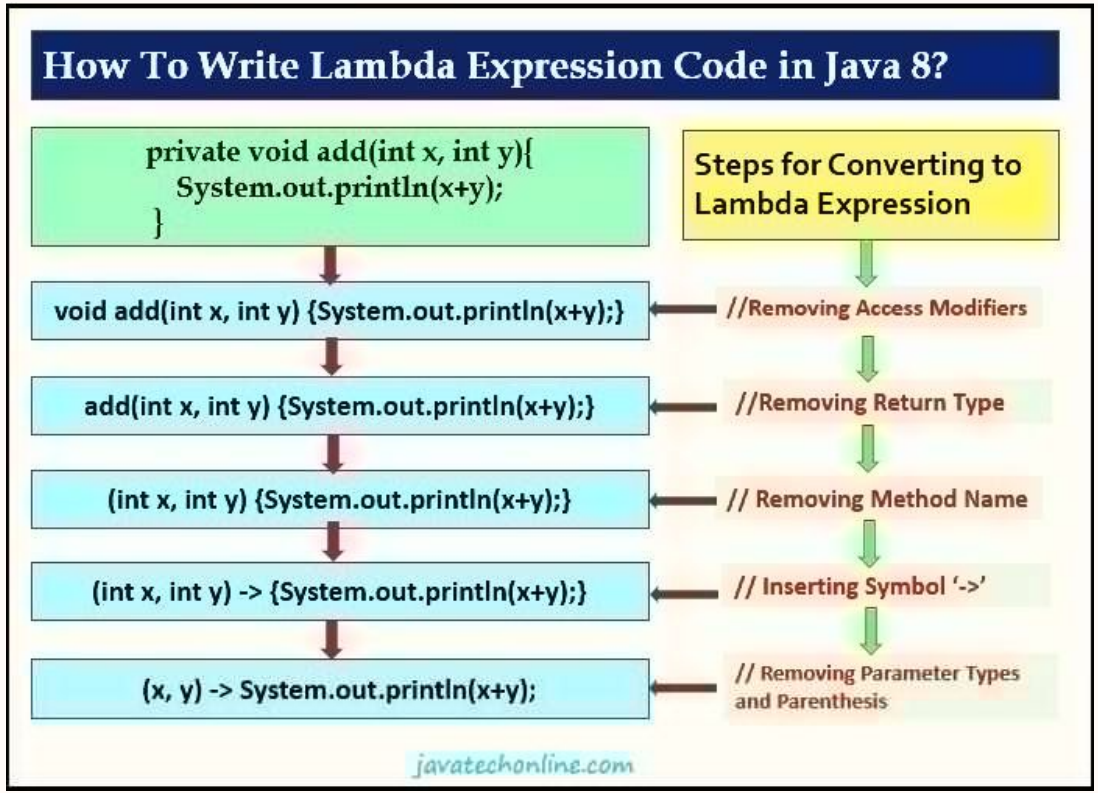


Biểu thức Lambda - Lambda Expression

- **Định nghĩa**:
  - Biểu thức Lambda cung cấp cách thức implement cho method được định nghĩa ở functional interface.
  - Lambda Expression là một hàm ẩn danh, không có tên, không thuộc bất kỳ lớp nào, không có phạm vi truy cập (private, public hoặc protected), không khai báo kiểu trả về.

- **Ý nghĩa**:
  - Lambda cũng cung cấp các thư viện giúp cải tiến cách thức làm việc với Collection như duyệt, filter, và truy xuất dữ liệu
  - hỗ trợ thực hiện tuần tự (Sequential) và song song (Parallel) hiệu quả hơn thông qua Stream API
  - Hỗ trợ viết ít code hơn

Biểu thức Lambda - Lambda Expression



Biểu thức Lambda - Lambda Expression

- **Cú pháp của Lambda: (argument-list) -> {body}**
  - **Argument-list**: Danh sách tham số, có thể không có, có một hoặc nhiều tham số.
  - **Arrow-token**: Toán tử mũi tiên được sử dụng để liên kết danh sách tham số và body của biểu thức.
  - **Body**: Nội dung thực thi, là 1 khối lệnh hoặc 1 biểu thức.

Biểu thức Lambda - Lambda Expression

- **Không có argument**

```
@FunctionalInterface
interface Hello {
        void sayHello();
}

public static void main(String[] args) {
        // Sử dụng biểu thức Lambda để triển khai phần thân cho phương thức sayHello()
        Hello h = () -> {
                System.out.println(“Hello World”);
        };
        h.sayHello();
}
```

Biểu thức Lambda - Lambda Expression

- **Có 1 argument**

```
@FunctionalInterface
interface Hello {
        void sayHello(String name);
}

public static void main(String[] args) {
        // Sử dụng biểu thức Lambda để triển khai phần thân cho phương thức sayHello()
        // Nếu body chỉ có 1 dòng thì không cần dùng { }
        Hello h = (name) -> System.out.println(“Hello World, tôi là “ + name);
        h.sayHello(“Tuan");
}
```

Biểu thức Lambda - Lambda Expression

- **Có nhiều hơn 1 arguments**

```
@FunctionalInterface
interface Hello {
        int timX(int x, int y);
}

public static void main(String[] args) {
        // Các arguments cách nhau bởi dấu phẩy
        // Nếu body có return thì bắt buộc phải sử dụng { }
        Hello h = (x, y) -> {
                return x * y;
        };
        System.out.println(h.timX(2, 4));
}
```

Biểu thức Lambda - Lambda Expression

- **Biểu thức lambda với forEach**

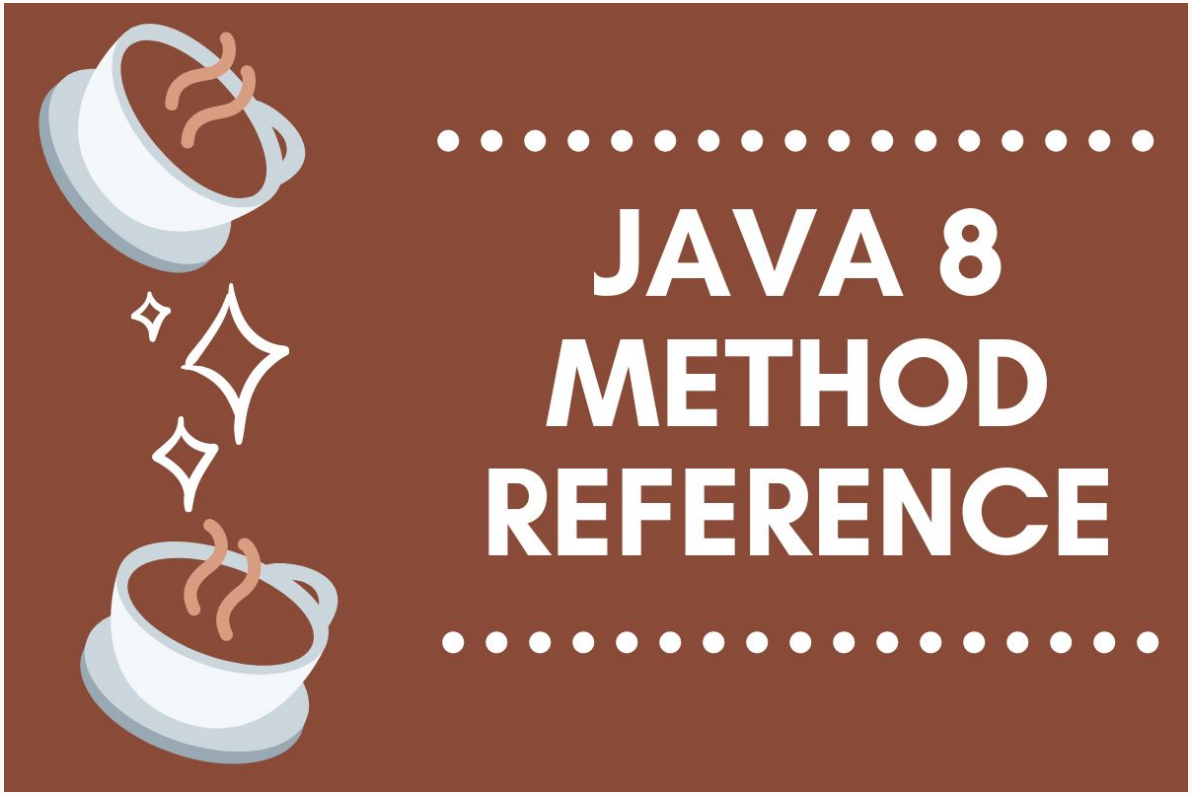
```
public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Java");
        list.add("PHP");
        list.add("C++");
        list.add("Python");

        list.forEach((element) -> {
                System.out.println(element);
        });
}
```

Tại sao lại sử dụng được biểu thức Lambda với forEach?

Thì giữ Ctrl + Click vào forEach để xem chứ làm slide mà cứ hỏi hoài

Method references

Phương thức tham chiếu - Method references

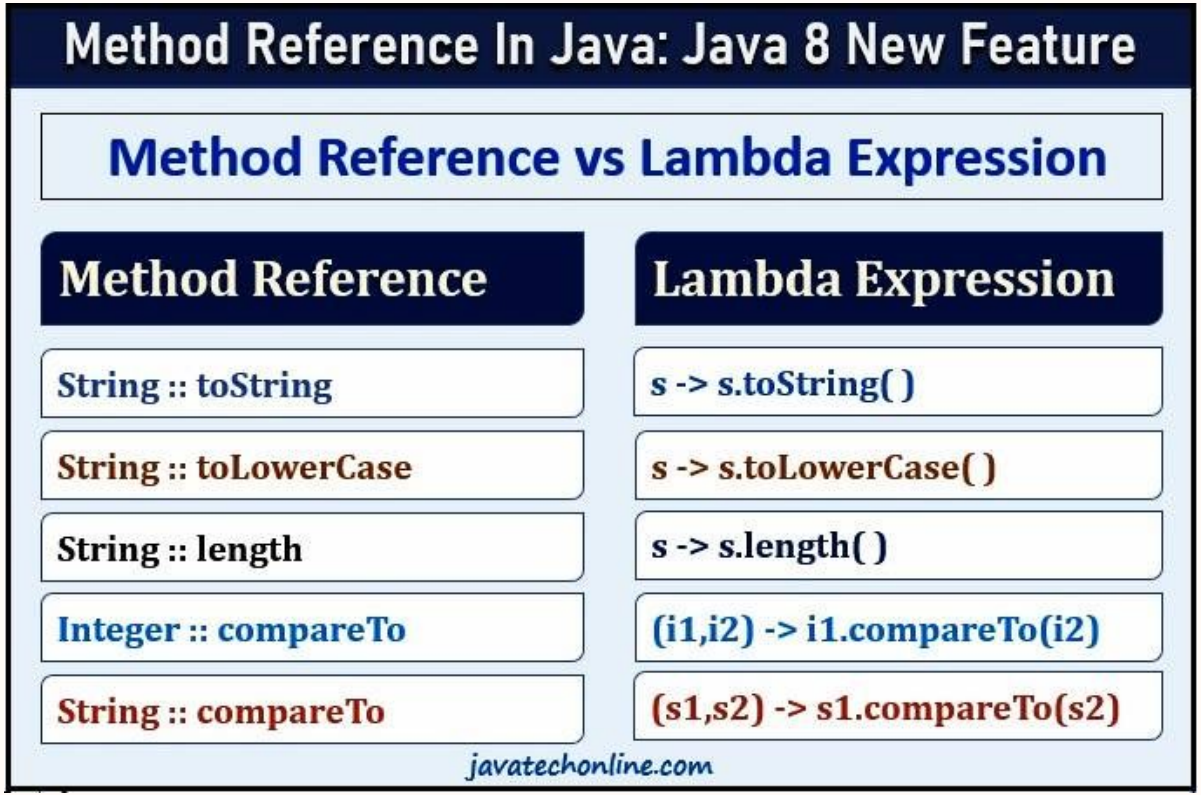


Phương thức tham chiếu - Method references

- **Định nghĩa**:
  - **Method references** cung cấp các cú pháp (syntax) hữu ích để truy cập trực tiếp tới các constructor hoặc method đã tồn tại của các lớp hoặc đối tượng trong Java mà không cần thực thi chúng.
  - Một method reference là cú pháp ngắn của lambda expression giúp thực thi một method.

- **Lưu ý**:
  - Method reference không thể sử dụng cho nhiều method.
  - Method reference được tạo ra để thay thế 1 lambda expression sử dụng 1 method.

Phương thức tham chiếu - Method references



Phương thức tham chiếu - Method references

- **Cú pháp chung của Method references: Object :: methodName**

- **Có 4 loại Method references:**
  - Tham chiếu đến một static method – **Class::staticMethod**
  - Tham chiếu đến một instance method của một đối tượng cụ thể – **object::instanceMethod**
  - Tham chiếu đến một instance method của một đối tượng tùy ý của một kiểu cụ thể – **Class::instanceMethod**
  - Tham chiếu đến một constuctor – **Class::new**

Phương thức tham chiếu - Method references

Tham chiếu đến một static method – Class::staticMethod

```
@FunctionalInterface
public interface MRInterface { int timXY(int x, int y); }

public class Service {
        public static int tinhTong(int x, int y) { return x + y; }

public static void main(String[] args) {
        int x = 10; int y = 5;
        int z = toDo(x, y, Service::tinhTong);
}

public static int toDo(int x, int y, MRInterface s) {
  return s.timXY(x, y);
}
```

Phương thức tham chiếu - Method references

Tham chiếu đến một instance method của một đối tượng cụ thể – object::instanceMethod

```
@FunctionalInterface
public interface MRInterface { int timXY(int x, int y); }

public class Service {
        public int tinhHieu(int x, int y) { return x - y; }

public static void main(String[] args) {
        int x = 10; int y = 5;
        Service sv = new Service();
        int z = toDo(x, y, sv::tinhHieu);
}

public static int toDo(int x, int y, MRInterface s) { return s.timXY(x, y); }
```

Phương thức tham chiếu - Method references

Tham chiếu đến một instance method của một đối tượng tùy ý của một kiểu cụ thể – **Class::instanceMethod**

```
// Khai báo thông qua lớp vô danh anonymous class
Function<Integer, String> convert = new Function<Integer, String>() {
        @Override
        public String apply(Integer integer) {
                return String.valueOf(integer);
        }
};

// Sử dụng Lambda expression
Function<Integer, String> convert = integer -> String.valueOf(integer);

// Sử dụng Method references
Function<Integer, String> convert = String::valueOf;
```

Phương thức tham chiếu - Method references

Tham chiếu đến một constuctor – Class::new

```
@FunctionalInterface
interface SayHello { void display(String say); }

class Hello implements SayHello {
    public Hello(String say) { System.out.print(say); }

@Override
public void display(String say) { System.out.println(say); }

public class Application {
    public static void main(String[] args) {
        SayHello ref = Hello::new;
        ref.display("Hello World!");
    }
}
```

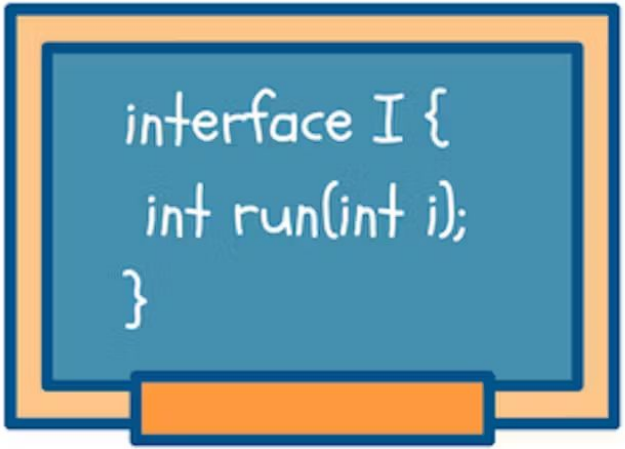
Stream API

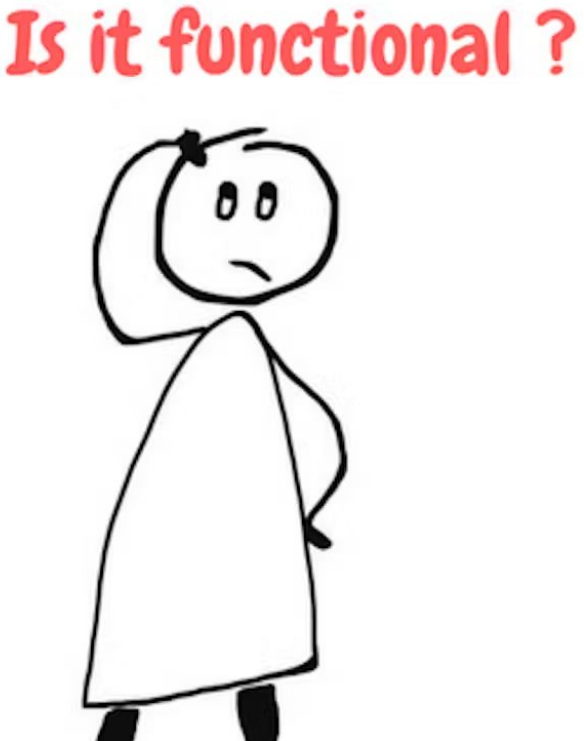
Stream API

- **Định nghĩa:**
  - **Stream** (luồng) hỗ trợ việc thao tác trên collection và array trở nên dễ dàng và tối ưu hơn.
  - Một Stream đại diện cho một chuỗi các phần tử hỗ trợ các hoạt động tổng hợp tuần tự (sequential) và song song (parallel).

- **Ý nghĩa:**
  - Stream hỗ trợ thực hiện các phép toán tổng hợp khác nhau trên dữ liệu được trả về từ các collection, array, các hoạt động Input/Output.

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"));
}
```

Predicate<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

You ko thoát được đâu son, to be continue…

ROAD TO KOREA

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

mail@mail.com hoặc Zalo 0xxx xxx xxx