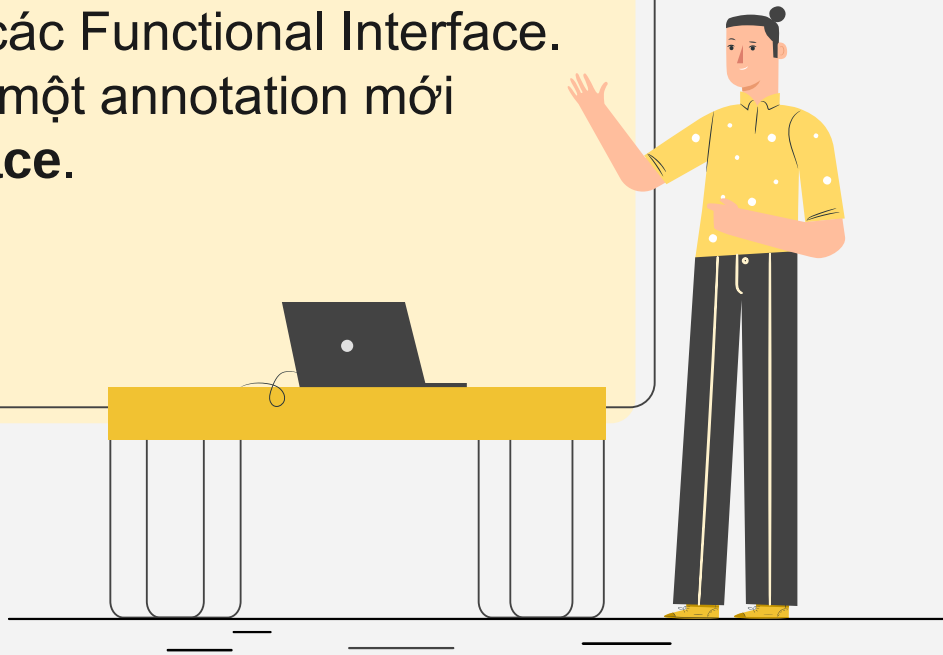
Lambda Expression & Functional Interface

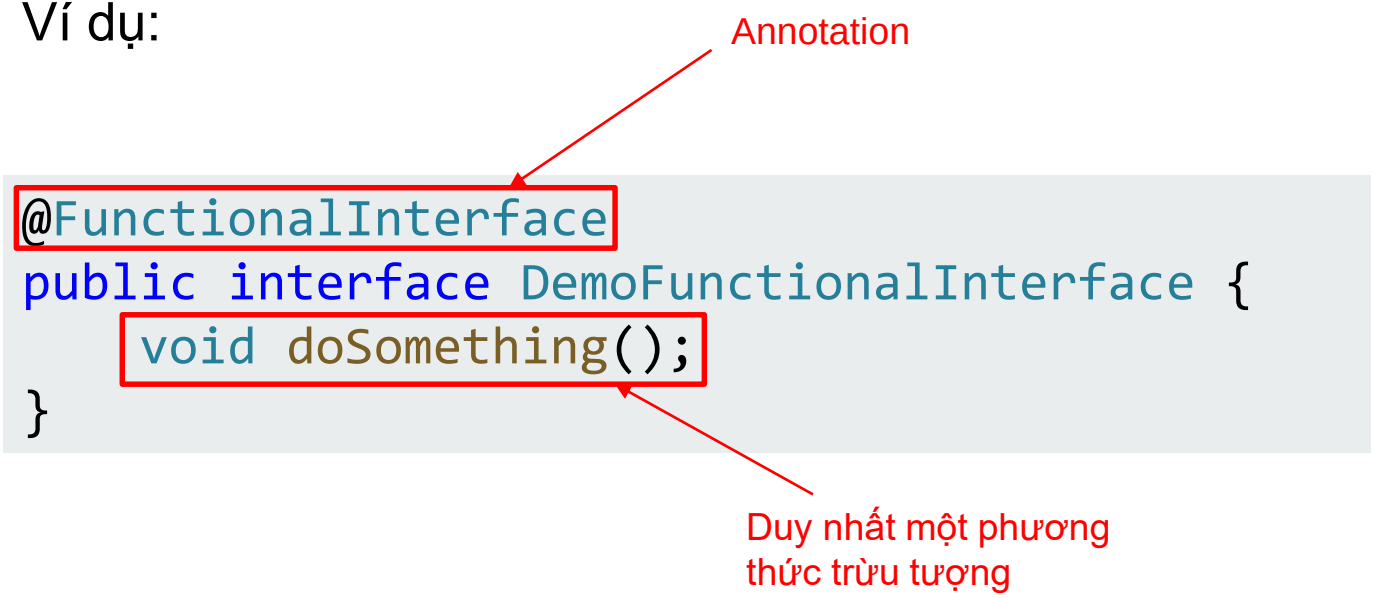
Functional Interface

- Java 8 gọi các interface có duy nhất một method trừu tượng là các Functional Interface.
- Java 8 cũng giới thiệu một annotation mới là **@FunctionalInterface**.



Functional Interfacec

Ví dụ:



Có thể thêm các phương thức không trừu tượng bằng từ khóa default và static

```
@FunctionalInterface
public interface DemoFunctionalInterface {
    void doSomething();
    default void defaultMethod1() {
    }

    default void defaultMethod2() {
    }

    static void staticMethod() {
    }
}
```

Từ khóa default

Từ khóa static

Anonymous Inner Class

Một lớp không có tên được gọi là lớp vô danh hay anonymous inner class. Nó nênđược sử dụng ếu bạn phải ghi đè phương thức của lớp hoặc interface.
Anonymous inner class có thể được tạo bằng hai cách:

01 Class

02 Interface

Khi nào nên sử dụng lớp vô danh

Lớp vô danh thường được sử dụng khi bạn không muốn phải khai báo cụ thể lớp con của một lớp nào đó, kể cả khi bạn không muốn khai báo cụ thể lớp triển khai của một interface nào đó , mà vẫn muốn sử dụng các đối tượng của chúng

Đặc điểm của lớp vô danh

- Lớp vô danh chỉ có thể triển khai từ duy nhất một interface
- Lớp vô danh chỉ có thể kế thừa hoặc triển khai một lớp khác hoặc một interface khác
- Lớp vô danh không có constructor

Lambda Expression

Lamba Expression là một hàm không có tên với các tham số và nội dung thực thi. Nội dung thực thi của LE có thể là một khối lệnh hoặc 1 biểu thức

```
// Khong có tham so, 1 cau lenh
() -> expression

// 1 tham so, 1 cau lenh
(parameters) -> expression

// cac tham so va noi dung khoi
(arg1, arg2, ...) -> {
    body-block
}

// cac tham so, noi dung khoi, du lieu tra ve
(arg1, arg2, ...) -> {
    body-block;
    return return-value;
}
```

Method Reference

Method References (Phương thức tham chiếu) cung cấp các cú pháp hữu ích để truy cập trực tiếp tới constructor hoặc method đã tồn tại của các lớp hoặc đối tượng trong java mà không cần thực thi chúng

Method Reference

Method references là cú pháp viết tắt của biểu thức lambda để gọi phương thức. Ví dụ, nếu biểu thức lambda được viết như sau:

```
str -> System.out.println(str);
```

Có thể viết lại theo các của Method reference như sau:

```
System.out::println;
```

Các loại Method Reference

- 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

Stream

Stream là gì?

Stream (luồng) là một đối tượng mới của Java được giới thiệu từ phiên bản Java 8, giúp cho 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).

Một số phương thức của Stream

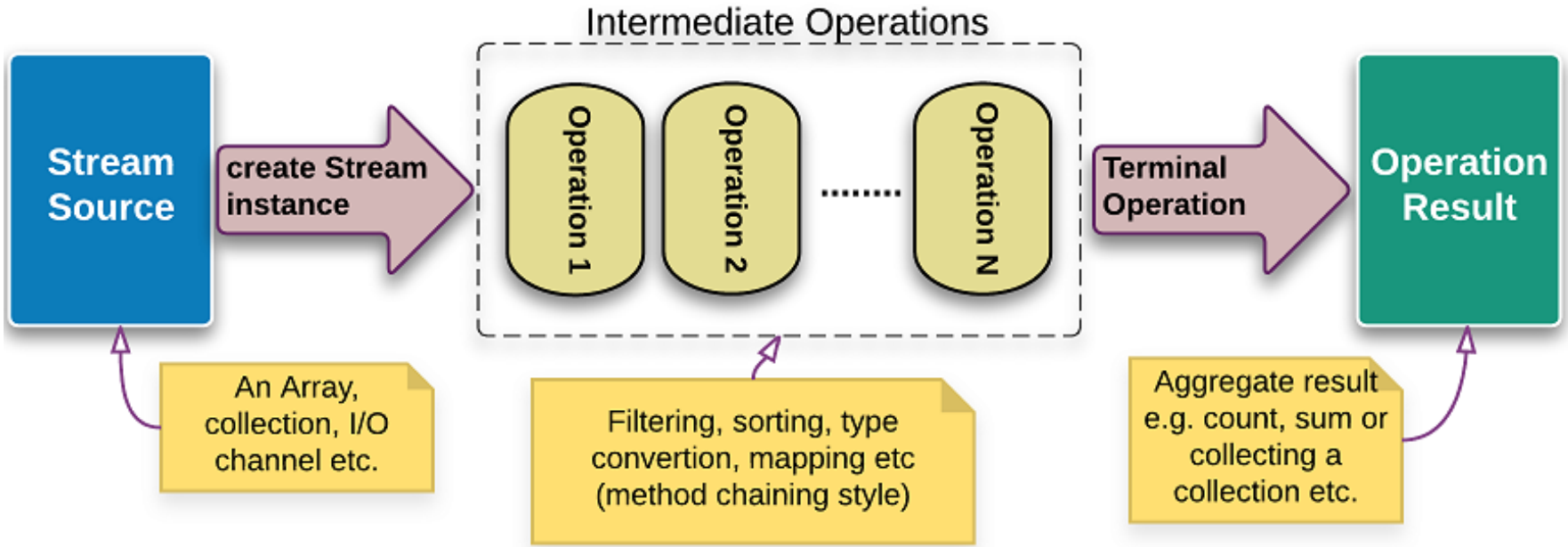
Trong Java 8, Collection interface được hỗ trợ 2 phương thức để tạo ra Stream bao gồm:

stream()

parallelStream()

```java
public class StreamExample  {
    List<Integer> numbers = Arrays.asList(7, 2, 5, 4, 2, 1);
    public void withoutStream() {
        long count = 0;
        for (Integer number : numbers) {
            if (number % 2 == 0) {
                count++;
            }
        }
        System.out.printf("There are %d elements that are even", count);
    }
    public void withStream() {
        long count = numbers.stream().filter(num -> num % 2 == 0).count();
        System.out.printf("There are %d elements that are even", count);
    }
}
```

Java Streams



Tạo Stream

Interface Stream trong package **java.util.stream** là interface đại diện cho một Stream. Interface này chỉ làm việc với kiểu dữ liệu là **Object**.
Với các kiểu primitive thì các bạn có thể sử dụng các đối tượng Stream dành cho những kiểu primitive đó, ví dụ như **IntStream**, **LongStream** hay **DoubleStream**.

Tạo Stream cho kiểu primitive

```
public class StreamExample  {
    public static void main(String[] args) {
        IntStream.range(1, 4).forEach(System.out::println);
        IntStream.of(1, 2, 3).forEach(System.out::println);
        DoubleStream.of(1, 2, 3).forEach(System.out::println);
        LongStream.range(1, 4).forEach(System.out::println);
        LongStream.of(1, 2, 3).forEach(System.out::println);
    }
}
```

Tạo Stream từ các cấu trúc dữ liệu khác

```
public class StreamExample  {
    public static void main(String[] args) {
        List<String> items = new ArrayList<>();
        items.add("Java");
        items.add("C#");
        items.add("C++");
        items.add("PHP");
        items.add("Javascript");

        items.stream().forEach(item -> System.out.println(item));
    }
}
```

```
public class StreamExample  {
    public static void main(String[] args) {
        String[] languages = { "Java", "C#", "C++", "PHP", "J
avascript" };

        // Get Stream using the Arrays.stream
        Stream<String> testStream = Arrays.stream(languages);
        testStream.forEach(x -> System.out.println(x));
    }
}
```