

Slide-2.1: Tổng quan Java 8, Interfaces, Method References

Giảng viên: Nguyễn Bá Minh Đạo



Nội dung

1. Tổng quan về lập trình với Java 8
2. Default Method, Static Method
3. Functional Interface trong Java 8
4. Method References trong Java 8



Tổng quan về lập trình với Java 8

Giới thiệu Java 8

- ❑ Oracle đã phát hành một phiên bản **Java 8** vào ngày 18/03/2014.
- ❑ Là một phiên bản mang tính cách mạng của Java cho phát triển phần mềm.
- ❑ Bao gồm các nâng cấp khác nhau cho lập trình Java, JVM, Tools, các thư viện.

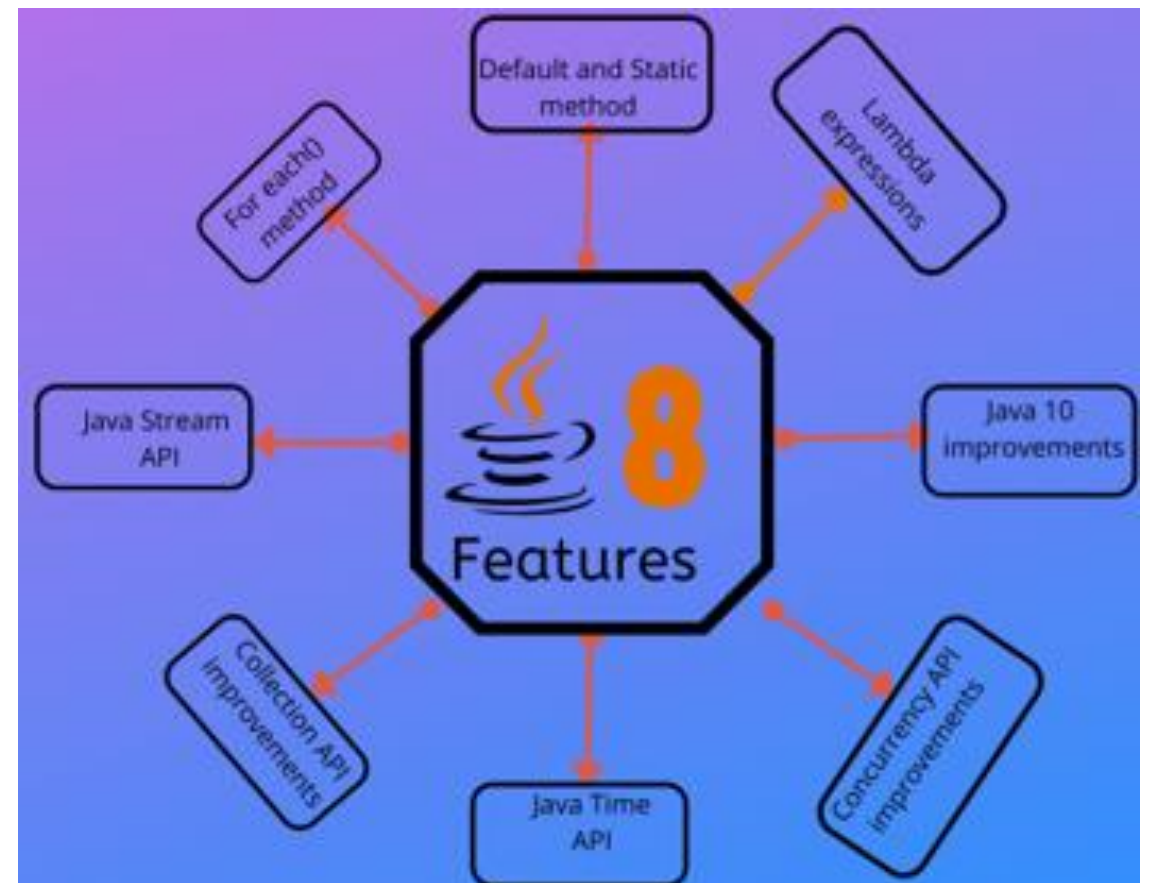




Tổng quan về lập trình với Java 8

Một số tính năng chủ yếu của Java 8

- ❑ **Default Method:** Cung cấp phương thức mặc định cho **Interface**.
- ❑ **Lambda Expression:** Thêm khả năng xử lý function cho Java.
- ❑ **Method References:** Các hàm tham chiếu theo tên của phương thức thay vì gọi trực tiếp. Sử dụng các function làm tham số.
- ❑ **Stream API:** bao gồm các class, interface, enum để cho phép các hoạt động kiểu function trên các element của một Collection, Array.

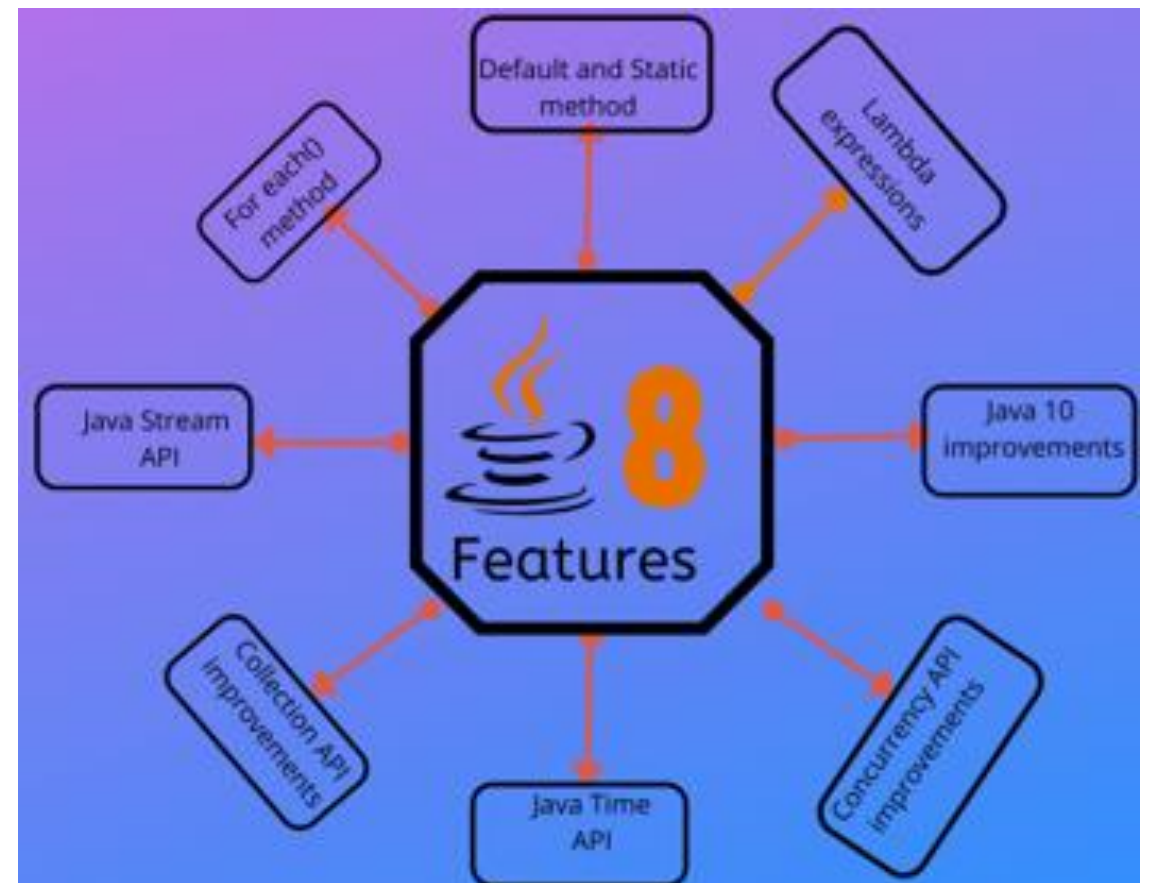




Tổng quan về lập trình với Java 8

Một số tính năng chủ yếu của Java 8

- ❑ **Collection API Enhancements:** một số cải tiến về Collections như stream, parallel stream, map, compute,...
- ❑ **Annotations:**
 - **Repeating Annotation** – cho phép các annotation giống nhau có thể được khai báo nhiều lần cùng một vị trí.
 - **Type Annotation** – có thể được áp dụng cho bất kỳ kiểu dữ liệu nào (Type), bao gồm: new operator, type casts, implements clauses, throws clauses.

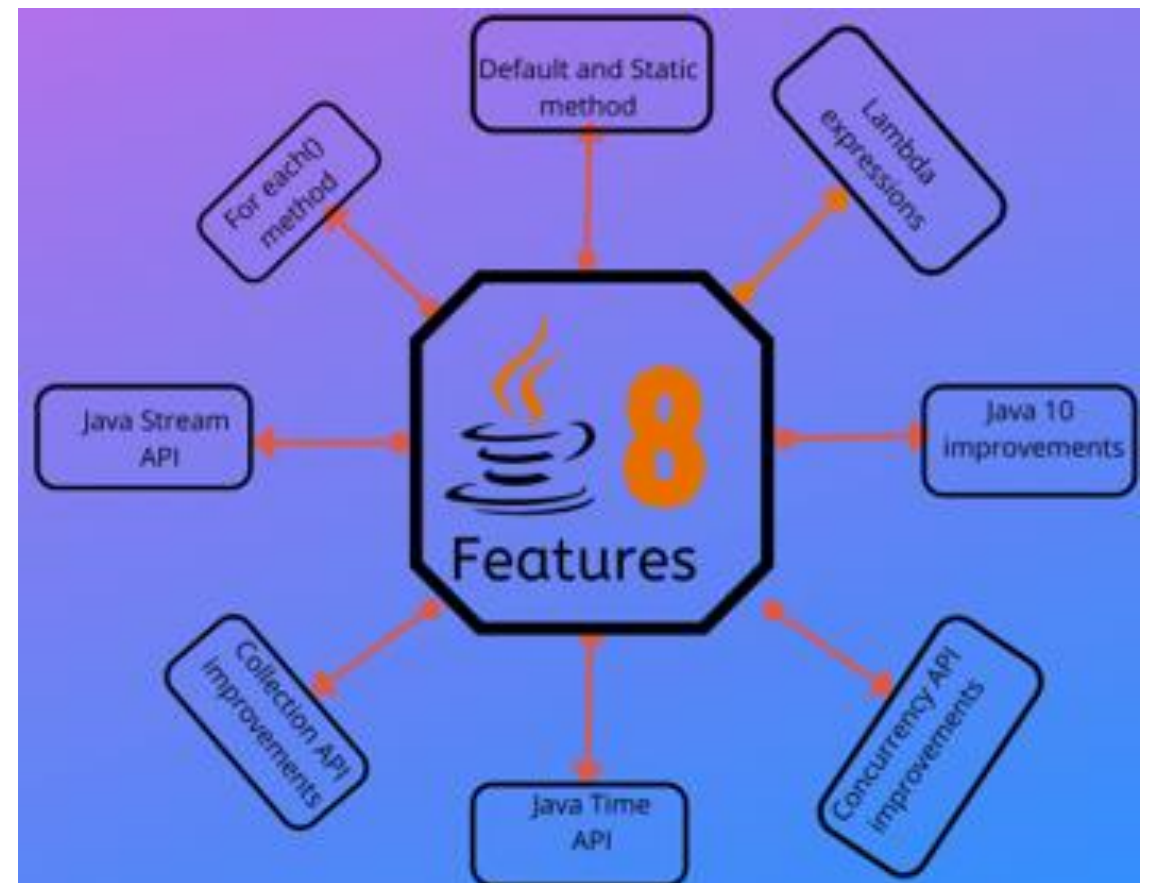




Tổng quan về lập trình với Java 8

Một số tính năng chủ yếu của Java 8

- ❑ **Date Time API:** cung cấp một số lớp mới trong gói java.time cùng với định dạng thời gian Joda.
- ❑ **Optional:** là một lớp được sử dụng để hạn chế với lỗi **NullPointerException** trong ứng dụng Java.
- ❑ **New tools:** các công cụ, tiện ích mới cho trình biên dịch được thêm vào như **jdeps**.
- ❑ **Nashorn, JavaScript Engine:** là JS Engine cho phép chạy JS trên JVM. Tương tự engine V8 (trên chrome)





Default Method, Static Method

Khái niệm Interface trong Java 8

- ❑ Một trong những thay đổi lớn trong Java 8 là khái niệm về **Interface**.
- ❑ Trong những phiên bản Java trước, interface chỉ cho phép chúng ta khai báo các phương thức **public abstract methods** bên trong nó.
- ❑ Nhưng trong Java 8, chúng ta sẽ có thêm 2 khái niệm mới đối với **interface** là phương thức **default** (default method) và phương thức **static** (static method).





Default Method, Static Method

Phương thức Default trong Interface (Default Method)

- ❑ Default Method là khái niệm mới trong Java 8. Nó cho phép chúng ta thêm vào các chức năng cho interface mà không làm phá vỡ các lớp implement từ interface này.
- ❑ Phương thức **setColor(String color)** chính là phương thức **default** của Shape. Khi class được implements từ Shape nó **không bắt buộc** phải implement phương thức default.

```
public interface Shape {  
  
    void draw();  
  
    default void setColor(String color) {  
        System.out.println("Draw this Shape with color: " + color);  
    }  
}
```

1

```
public class Rect implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Draw Rectangle");  
    }  
}
```

2

```
public class DefaultMethodExample01 {  
  
    public static void main(String[] args) {  
        Rect rect = new Rect();  
        rect.draw();  
        rect.setColor("Blue");  
  
        Circle circle = new Circle();  
        circle.draw();  
        circle.setColor("Red");  
    }  
}
```

4

```
public class Circle implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Draw Circle");  
    }  
}
```

3



Default Method, Static Method

Phương thức Default trong Interface (Default Method)

- ❑ Tính năng này giúp chúng ta dễ dàng mở rộng các phương thức bổ sung phát sinh sau này mà không ảnh hưởng đến các class liên quan.
- ❑ Khi muốn mở rộng một **interface** mà không ảnh hưởng các lớp liên quan, chúng ta chỉ cần viết thêm các **phương thức default** (default method) trong **interface** đó.

```
public interface Shape {  
  
    void draw();  
  
    default void setColor(String color) {  
        System.out.println("Draw this Shape with color: " + color);  
    }  
}
```

1

```
public class Rect implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Draw Rectangle");  
    }  
}
```

2

```
public class DefaultMethodExample01 {  
  
    public static void main(String[] args) {  
        Rect rect = new Rect();  
        rect.draw();  
        rect.setColor("Blue");  
  
        Circle circle = new Circle();  
        circle.draw();  
        circle.setColor("Red");  
    }  
}
```

4

```
public class Circle implements Shape {  
  
    @Override  
    public void draw() {  
        System.out.println("Draw Circle");  
    }  
}
```

3

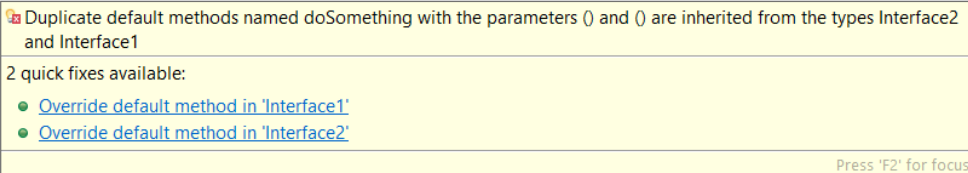


Default Method, Static Method

Điều gì xảy ra khi có đa thừa kế với 2 default method cùng tên?

- ❑ Một class có thể implements nhiều interface cùng 1 lúc. **Vậy điều gì sẽ xảy ra khi các interface này có các default methods cùng tên?**
- ❑ Khi 1 class triển khai 2 interface chứa default method cùng tên -> xảy ra xung đột (conflict) vì lúc này Java sẽ không biết phải sử dụng default method nào cho phù hợp. Khi đó, trình biên dịch của Java sẽ thông một lỗi tương tự như sau:

```
interface Interface1 {  
    default void doSomething() {  
        System.out.println("Do something 1...");  
    }  
}  
  
interface Interface2 {  
    default void doSomething() {  
        System.out.println("Do something 2...");  
    }  
}  
  
public class MultiInheritance implements Interface1, Interface2 {
```



Duplicate default methods named doSomething with the parameters () and () are inherited from the types Interface2 and Interface1

2 quick fixes available:

- [Override default method in 'Interface1'](#)
- [Override default method in 'Interface2'](#)

Press 'F2' for focus



Default Method, Static Method

Điều gì xảy ra khi có đa thừa kế với 2 default method cùng tên?

- ❑ Để giải quyết vấn đề này, chúng ta có thể sử dụng cách sau để giải quyết:
 - Override lại phương thức doSomething từ lớp con.
 - Gọi default method của một Interface cụ thể bằng cách sử dụng từ khóa super.

```
interface Interface1 {  
    default void doSomething() {  
        System.out.println("Do something 1...");  
    }  
}  
  
interface Interface2 {  
    default void doSomething() {  
        System.out.println("Do something 2...");  
    }  
}  
  
public class MultiInheritance implements Interface1, Interface2 {  
  
    @Override  
    public void doSomething() {  
        Interface1.super.doSomething();  
    }  
}
```



Default Method, Static Method

Khi có một super class có phương thức (abstract method hoặc non-abstract method) cùng tên với default method, phương thức nào sẽ thực thi?

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
}  
  
class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
}  
  
public class MultiInheritance02 extends Parent implements Interface3 {  
  
    public static void main(String[] args) {  
        MultiInheritance02 m = new MultiInheritance02();  
        m.doSomething(); //Execute in Parent  
    }  
}
```

- ❑ Nếu một lớp con thừa kế một phương thức (abstract hoặc non-abstract) từ một super class và một phương thức cùng tên trong các super interface, thì lớp con sẽ thừa kế phương thức của super class và các phương thức của interface sẽ bị bỏ qua.



Default Method, Static Method

Một số đặc điểm quan trọng về Default Methods trong Interface

- ❑ Giúp dễ dàng mở rộng **interface** mà không phá vỡ các **class** được **implements** từ nó
- ❑ Giúp tránh dùng các **class** tiện ích, ví dụ như tất cả phương thức của class **Collections** có thể được cung cấp ngay bên trong interface của nó.
- ❑ Giúp **tháo gỡ** các **class** **cơ sở** (base class), chúng ta **có thể tạo default method** và trong **class** được **implement** có thể chọn phương thức để **override**.





Default Method, Static Method

Một số đặc điểm quan trọng về Default Methods trong Interface

- ❑ Một trong những lý do xuất hiện của **default method** là để nâng cấp **Collection API** trong **Java 8** hỗ trợ cho **Lambda Expression**.
- ❑ Nếu bất kỳ **class** nào kế thừa những **default method** giống nhau, thì nó sẽ không còn hiệu lực.
- ❑ Một **default method** sẽ không thể override một phương thức từ **java.lang.Object** vì **Object** là **base class** của tất cả các class trong Java.





Default Method, Static Method

Một số đặc điểm quan trọng về Default Methods trong Interface

- ❑ **Default Method** cũng có thể được gọi là phương thức **Defender** (**Defender Methods**) hay là **phương thức Virtual mở rộng** (**Virtual Extension Methods**).
- ❑ **Default Method** cho phép các Java interface đã tồn tại phát triển thêm mà không gây lỗi trong quá trình biên dịch. Ví dụ như bổ sung các phương thức vào interface **java.util.Collection**: `stream()`, `parallelStream()`, `forEach()`, ...





Default Method, Static Method

Phương thức static trong Interface (Static Method)

- ❑ Phương thức **static** cũng giống như phương thức **default**, ngoại trừ việc nó **không thể được override** chúng trong class được implements.

```
interface Vehicle {  
    default void print() {  
        if (isValid())  
            System.out.println("Vehicle printed");  
    }  
  
    static boolean isValid() {  
        System.out.println("Vehicle is valid");  
        return true;  
    }  
  
    void showLog();  
}  
  
public class Car implements Vehicle {  
    @Override  
    public void showLog() {  
        print();  
        Vehicle.isValid();  
    }  
}
```

- ❑ Trong đoạn code trên, trong lớp **Car** chúng ta có thể sử dụng phương thức **static isValid()** của **interface Vehicle**.



Default Method, Static Method

Một số đặc điểm quan trọng về Static Methods trong Interface

- ❑ **Static Method** là một thành phần của **interface**, chúng ta có thể sử dụng nó trong **class** được **implements** từ nó.
- ❑ Phương thức static rất hữu ích trong việc cung cấp các phương thức tiện ích.
- ❑ Lớp con có thể định nghĩa một phương thức cùng tên với **static method** của **interface** cha. Nhưng, chúng ta không thể thêm annotation **@Override** cho phương thức này.





Default Method, Static Method

Một số đặc điểm quan trọng về Static Methods trong Interface

- ❑ **Không thể** định nghĩa **static method** của các phương thức thuộc class **Object**, chúng ta sẽ gặp lỗi “**This static method cannot hide the instance method from Object**”.
- Vì **Object** là **base class** cho tất cả các **class** và chúng ta không thể có một phương thức **static** và một phương thức khác cùng định dạng.

So sánh Abstract Classes và Interface trong Java 8

Abstract Classes	Interface trong Java 8
Có thể có hàm tạo (constructor)	Không thể có hàm tạo (constructor)
Mục đích của abstract class là cung cấp một phần trừu tượng hóa (<100% abstract)	Mục đích của interface là cung cấp sự trừu tượng đầy đủ (100% abstract)



Functional Interface trong Java 8

So sánh Object-Oriented Programming và Functional Programming

❑ Object-Oriented Programming (OOP)

- Là một mô hình lập trình dựa trên khái niệm **đối tượng (object)**
- Là kết hợp giữa **data** thể hiện dưới dạng các trường (field) hoặc thuộc tính (attribute) và **code** thể hiện dưới dạng thủ tục (procedure) hoặc phương thức (method).

❑ Functional Programming (FP)

- Là một mô hình lập trình, một kiểu xây dựng cấu trúc và các phần tử của các chương trình, xử lý tính toán dựa trên các function và nó tránh thay đổi trạng thái, dữ liệu có thể thay đổi được (mutable data).





Functional Interface trong Java 8

So sánh Object-Oriented Programming và Functional Programming

- ❑ Trong 1 chương trình, có 2 thành phần chính: dữ liệu và hành vi (behavior hoặc code).
 - **OOP** phát biểu rằng tập hợp dữ liệu và hành vi liên quan của nó thành một “đối tượng” giúp dễ hiểu hơn về cách thức hoạt động của một chương trình.
 - **FP** phát biểu rằng dữ liệu và hành vi là khác biệt rõ rệt và nên được giữ riêng biệt để rõ ràng.





Functional Interface trong Java 8

Functional Interface là gì?

- ❑ Java 8 gọi các Interface có duy nhất một method trừu tượng là các **Functional Interface**
- ❑ Cũng có thể được gọi là **Single Abstract Method** (SAM) -> đôi khi chúng ta sẽ bắt gặp.
- ❑ Java 8 cũng giới thiệu một annotation mới là **@FunctionalInterface**
 - **@FunctionalInterface** có thể sử dụng cho **mục đích bắt lỗi ở thời điểm biên dịch** nếu vô tình thêm một method trừu tượng khác nữa vào interface có đánh dấu bởi annotation này mà vi phạm quy tắc của **Functional Interface**.
- ❑ Lợi ích chính của **functional interface** là chúng ta có thể sử dụng **Lambda Expression** để tạo ra thể hiện (instance) cho **interface** đó.



Functional Interface trong Java 8

Quy tắc khai báo Functional Interface

- ❑ Một **Functional Interface** **hợp lệ** chỉ có duy nhất một method trừu tượng.

```
@FunctionalInterface
public interface FunctionalInterfaceExample01 {

    void doSomething();

}
```

- ❑ Các **functional Interface** **không hợp lệ** nếu không có bất kỳ phương thức trừu tượng nào hoặc có nhiều hơn một method trừu tượng.

```
@FunctionalInterface
public interface FunctionalInterface_Invalid1 {

}
```

```
@FunctionalInterface
public interface FunctionalInterface_Invalid2 {

    void doSomething1();

    void doSomething2();

}
```



Functional Interface trong Java 8

Quy tắc khai báo Functional Interface

- ❑ Một **Functional Interface** có thể có các phương thức của lớp **java.lang.Object**

```
@FunctionalInterface
public interface FunctionalInterfaceExample02 {

    void doSomething();

    int hashCode();

    String toString();

    boolean equals(Object obj);

}
```

- ❑ **Functional Interface** không hợp lệ nếu nó chỉ khai báo các phương thức của lớp **java.lang.Object** mà không có bất kỳ abstract method nào của riêng nó.

```
@FunctionalInterface
public interface FunctionalInterface_Invalid3 {

    int hashCode();

    String toString();

    boolean equals(Object obj);

}
```



Functional Interface trong Java 8

Quy tắc khai báo Functional Interface

❑ **Functional Interface** dưới đây **không hợp lệ** vì:

➤ Phương thức **clone()** của lớp **java.lang.Object** có phạm vi truy cập là **protected**. Trong khi tất cả các **abstract method** của interface đều là **public abstract** nên phương thức **clone()** này không phải là **override** lại phương thức **clone()** của lớp **Object**.
=> Do đó interface này được xem như khai báo 2 phương thức **doSomething()** và **clone()**, nên vi phạm nguyên tắc của **Functional Interface**.

```
@FunctionalInterface
public interface FunctionalInterface_Invalid4 {

    void doSomething();

    Object clone();
}
```



Functional Interface trong Java 8

Default/Static Methods không phá vỡ quy tắc của Functional Interface

```
@FunctionalInterface
public interface FunctionalInterfaceExample03 {

    void doSomething();

    default void defaultMethod1() {

    }

    default void defaultMethod2() {

    }

    static void staticMethod() {

    }

}
```

Một Functional Interface có thể mở rộng một Interface khác chỉ khi nó không có bất kỳ phương thức trừu tượng nào

```
interface BaseInterface1 {
    void base();
}

interface BaseInterface2 {
    void base();
}

@FunctionalInterface
public interface FunctionalInterfaceExample04 extends BaseInterface1, BaseInterface2 {

}
```



Functional Interface trong Java 8

Một Functional Interface có thể mở rộng một Interface khác chỉ khi nó không có bất kỳ phương thức trừu tượng nào

- ❑ **Functional Interface** bên dưới đây **không hợp lệ** vì có Interface con có một phương thức trừu tượng **doSomething()** của riêng nó và kế thừa 1 phương thức **base()** từ **Interface Base**.

```
interface BaseInterface {  
    void base();  
}  
  
@FunctionalInterface  
public interface FunctionalInterface_Invalid05 extends BaseInterface {  
  
    void doSomething();  
}
```




Method References trong Java 8

Method References là gì?

- ❑ Java cung cấp một tính năng mới gọi là **Method References** (phương thức tham chiếu) trong Java 8.
- ❑ **Method References** là một tính năng khá hay và liên quan đến việc sử dụng **Lambda Expression**.
- ❑ Nó 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.
- ❑ Nó làm cho việc viết code của chúng ta trở nên đơn giản hơn rất nhiều và nhìn chúng đẹp hơn.





Method References trong Java 8

Method References là gì?

- ❑ **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)`
- ❑ Chúng ta có thể viết lại theo cách của **Method References** như sau: `System.out::println`
- ❑ Java 8 **cho phép truyền một tham chiếu** của một **method** hoặc **constructor** thông qua việc sử dụng từ khóa **::**





Method References trong Java 8

Các loại Method References

- ❑ Tham chiếu đến 1 static method – **Class::staticMethod**
- ❑ Tham chiếu đến 1 instance method của 1 đối tượng cụ thể - **object::instanceMethod**
- ❑ Tham chiếu đến 1 instance method của 1 đối tượng tùy ý của một kiểu cụ thể - **Class::instanceMethod**
- ❑ Tham chiếu đến một constructor – **Class::new**





Method References trong Java 8

Tham chiếu đến một static method

```
@FunctionalInterface
interface ExecuteFunction {
    public int execute(int a, int b);
}

class MathUtils {
    public static int sum(int a, int b) {
        return a + b;
    }

    public static int minus(int a, int b) {
        return a - b;
    }
}

public class MethodReferencesExample01 {
    public static void main(String[] args) {
        int a = 10;
        int b = 7;

        int sum = doAction(a, b, MathUtils::sum);
        System.out.println(a + " + " + b + " = " + sum); // 10 + 7 = 17

        int minus = doAction(a, b, MathUtils::minus);
        System.out.println(a + " - " + b + " = " + minus); // 10 - 7 = 3
    }

    public static int doAction(int a, int b, ExecuteFunction func) {
        return func.execute(a, b);
    }
}
```



Method References trong Java 8

Tham chiếu đến một instance method của một đối tượng cụ thể

```
@FunctionalInterface
interface ExecuteFunction2 {
    public int execute(int a, int b);
}

class MathUtils2 {
    public int sum(int a, int b) {
        return a + b;
    }

    public int minus(int a, int b) {
        return a - b;
    }
}

public class MethodReferencesExample02 {
    public static void main(String[] args) {
        int a = 10;
        int b = 7;

        MathUtils2 obj = new MathUtils2();
        int sum = doAction(a, b, obj::sum);
        System.out.println(a + " + " + b + " = " + sum); // 10 + 7 = 17

        int minus = doAction(a, b, obj::minus);
        System.out.println(a + " - " + b + " = " + minus); // 10 - 7 = 3
    }

    public static int doAction(int a, int b, ExecuteFunction2 func) {
        return func.execute(a, b);
    }
}
```



Method References trong Java 8

Tham chiếu đến một constructor

```
@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 MethodReferencesExample04 {
    public static void main(String[] args) {
        SayHello ref = Hello::new;
        ref.display("Welcome to javacoder.com");
    }
}
```




Method References trong Java 8

Tham chiếu đến một instance method của một đối tượng tùy ý của một kiểu cụ thể

```
public class MethodReferencesExample03 {  
    public static void main(String[] args) {  
        String[] stringArray = { "Java", "C++", "PHP", "C#", "Javascript" };  
  
        Arrays.sort(stringArray, String::compareToIgnoreCase);  
        for (String str : stringArray) {  
            System.out.println(str);  
        }  
    }  
}
```

Một số lưu ý về Method References

- ❑ Có thể sử dụng **Method References** để thay thế cho các **Lambda Expression** khi **Lambda** gọi một phương thức nào đó đã được định nghĩa sẵn.
- ❑ Không thể truyền tham số cho các Method References, phải sử dụng đi kèm với **Functional Interfaces**.



Tổng kết nội dung bài học

- ☐ Tổng quan về lập trình với Java 8
- ☐ Default Method, Static Method
- ☐ Functional Interface trong Java 8
- ☐ Method References trong Java 8

Let's
Recap

