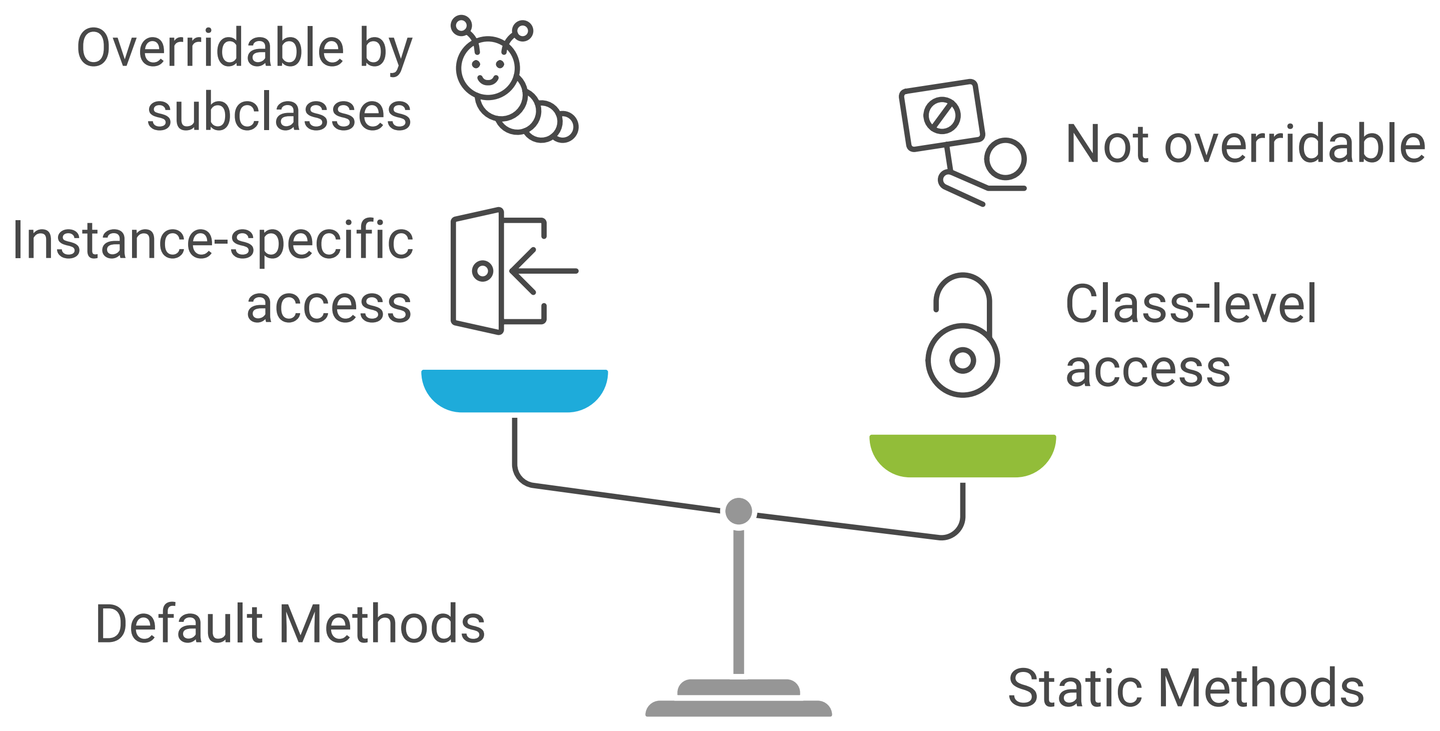
Default and Static Methods in Functional Interfaces

In Java, functional interfaces are interfaces that contain exactly one abstract method, and they can have multiple default and static methods. This document provides a summary of default and static methods in functional interfaces, including their implementation, differences, and real-time usage scenarios.

Abstract

Functional interfaces are a key component of Java's functional programming capabilities introduced in Java 8. They allow for the use of lambda expressions and method references, making code more concise and readable. Default and static methods enhance the functionality of these interfaces, providing additional behaviour without breaking existing implementations. This document explores how to implement these methods, the differences between them, and when to use them in real-world applications.



Default Methods

Implementation

A default method is defined in an interface using the **default** keyword. It provides a default implementation that can be overridden by implementing classes. Here's an example:

```
@FunctionalInterface
interface MyFunctionalInterface {
    void abstractMethod();

    default void defaultMethod() {
        System.out.println("This is a default method.");
    }
}

class MyClass implements MyFunctionalInterface {
    @Override
    public void abstractMethod() {
        System.out.println("Implementing abstract method.");
    }
}
```

Usage

Default methods are useful when you want to add new functionality to an interface without breaking existing implementations. They allow for backward compatibility and can be used to provide common behaviour across multiple classes.

Static Methods

Implementation

Static methods in a functional interface are defined using the **static** keyword. They belong to the interface itself rather than any instance of the interface. Here's an example:

```
@FunctionalInterface
interface MyFunctionalInterface {
    void abstractMethod();

    static void staticMethod() {
        System.out.println("This is a static method.");
    }
}
```

Usage

Static methods are typically used for utility functions that are related to the interface but do not require an instance of the interface to be invoked. They can be called directly on the interface, making them convenient for operations that are common to all implementations.

Differences Between Default and Static Methods

1. **Invocation Context**:

    - Default methods can be called on instances of the implementing class.
    - Static methods are called on the interface itself and cannot be overridden.

2. **Purpose**:

    - Default methods provide a way to add new functionality to existing interfaces without breaking existing implementations.
    - Static methods are used for utility functions that do not depend on instance state.

3. **Inheritance**:
    - Default methods can be inherited and overridden by implementing classes.
    - Static methods cannot be inherited and are not part of the instance's behaviour.

Real-Time Usage Scenarios

- **Default Methods**: When you want to evolve an interface by adding new methods while maintaining backward compatibility. For example, in a library where multiple classes implement the same interface, adding a default method allows you to introduce new functionality without forcing all classes to implement it.

- **Static Methods**: When you need utility methods that are relevant to the interface but do not require an instance. For instance, a static method can be used to perform calculations or transformations related to the interface's purpose, such as a factory method for creating instances.

In conclusion, default and static methods in functional interfaces provide powerful tools for enhancing interface functionality while maintaining flexibility and backward compatibility. Understanding their differences and appropriate use cases is essential for effective Java programming.