Direkte Methodenimplementierung in Interfaces (Default-Methoden) in Java

Seit Java 8 können Interfaces **Default-Methoden** enthalten. Diese ermöglichen es, **Methoden direkt im Interface** mit einer Standardimplementierung zu definieren. Dadurch entfällt in vielen Fällen der Bedarf an Adapterklassen, da du nur noch die Methoden überschreiben musst, die du wirklich anpassen möchtest.

Was sind Default-Methoden?

Default-Methoden:

- Werden direkt im Interface definiert.
- Haben eine **Standardimplementierung**, die von den implementierenden Klassen übernommen werden kann.
- Können von der implementierenden Klasse **überschrieben** werden, wenn eine andere Implementierung benötigt wird.

Beispiel: Ein Interface mit Default-Methoden

Stell dir ein Interface `AutoAktionen` vor, das Standardmethoden für ein Auto definiert.

```
public interface AutoAktionen {
    void motorStarten(); // Muss von allen implementierenden Klassen definiert
werden

    default void beschleunigen() {
        System.out.println("Das Auto beschleunigt mit Standardgeschwindigkeit.");
    }

    default void bremsen() {
        System.out.println("Das Auto bremst standardmäßig.");
    }

    default void hupen() {
        System.out.println("Die Hupe ertönt standardmäßig.");
    }
}
```

Implementierung in einer Klasse

Jetzt kannst du das Interface implementieren und nur die Methoden überschreiben, die du anpassen möchtest.

```
public class ElektroAuto implements AutoAktionen {
    @Override
    public void motorStarten() {
```

```
        System.out.println("Der Elektromotor startet lautlos.");
    }

    // Keine Überschreibung von `beschleunigen()`, `bremsen()` oder `hupen()`
nötig,
    // da die Standardimplementierung aus dem Interface verwendet wird.
}
```

Ausgabe:

```
public class Main {
    public static void main(String[] args) {
        AutoAktionen auto = new ElektroAuto();
        auto.motorStarten(); // "Der Elektromotor startet lautlos."
        auto.beschleunigen(); // "Das Auto beschleunigt mit
Standardgeschwindigkeit."
        auto.bremsen(); // "Das Auto bremst standardmäßig."
        auto.hupen(); // "Die Hupe ertönt standardmäßig."
    }
}
```

Überschreiben einer Default-Methode

Wenn eine Klasse eine andere Implementierung für eine Standardmethode benötigt, kann sie die Default-Methode überschreiben.

```
public class Sportwagen implements AutoAktionen {
    @Override
    public void motorStarten() {
        System.out.println("Der Sportwagen-Motor startet mit einem aggressiven
Röhren.");
    }

    @Override
    public void beschleunigen() {
        System.out.println("Der Sportwagen beschleunigt extrem schnell.");
    }
}
```

Ausgabe:

```
public class Main {
    public static void main(String[] args) {
        AutoAktionen auto = new Sportwagen();
        auto.motorStarten(); // "Der Sportwagen-Motor startet mit einem
```

```
aggressiven Röhren."
        auto.beschleunigen(); // "Der Sportwagen beschleunigt extrem schnell."
        auto.bremsen(); // "Das Auto bremst standardmäßig."
        auto.hupen(); // "Die Hupe ertönt standardmäßig."
    }
}
```

Vorteile von Default-Methoden

1. **Flexibilität:**
    - Implementierende Klassen müssen nur die Methoden überschreiben, die sie ändern wollen.
    - Es bleibt eine Standardimplementierung verfügbar, wenn keine spezifische benötigt wird.
2. **Weniger Boilerplate-Code:**
    - Du musst keine Adapterklassen oder leere Implementierungen für ungenutzte Methoden schreiben.
3. **Abwärtskompatibilität:**
    - Neue Methoden können zu einem bestehenden Interface hinzugefügt werden, ohne alte Implementierungen zu brechen.

Grenzen von Default-Methoden

1. **Ambiguitäten:**
    - Wenn eine Klasse von mehreren Interfaces mit gleichen Default-Methoden erbt, muss die Klasse eine eigene Implementierung der Methode bereitstellen.

```
public interface InterfaceA {
    default void methode() {
        System.out.println("Default in InterfaceA");
    }
}

public interface InterfaceB {
    default void methode() {
        System.out.println("Default in InterfaceB");
    }
}

public class Konflikt implements InterfaceA, InterfaceB {
    @Override
    public void methode() {
        System.out.println("Eigene Implementierung in Konflikt.");
    }
}
```

2. **Keine Zustände:**
    - Default-Methoden können keinen Zustand (Instanzvariablen) speichern, da Interfaces in Java keine Instanzvariablen haben dürfen.

Zusammenfassung

Default-Methoden in Interfaces	**Adapterklassen**
Einführung seit Java 8.	Seit Beginn von Java.
Ermöglicht Standardimplementierungen direkt im Interface.	Adapterklassen bieten leere Implementierungen für alle Methoden eines Interfaces.
Bessere Flexibilität und weniger Boilerplate-Code.	Besonders hilfreich bei älteren Interfaces mit vielen Methoden.
Ideal für neue und moderne APIs.	Geeignet für ältere APIs wie AWT/Swing.

Default-Methoden machen Adapterklassen in vielen Fällen überflüssig, vor allem bei modernen Java-Programmen. Sie bieten eine elegante und direkte Lösung für Standardimplementierungen.