Interfaces und Adapterklassen in Java

Du kannst **Interfaces** in Java entweder direkt oder mit Hilfe von **Adapterklassen** verwenden. Welche Variante du wählst, hängt von der Anzahl der Methoden im Interface und von deinem spezifischen Anwendungsfall ab.

---

1. Interfaces direkt verwenden

Wenn du ein Interface direkt implementierst, musst du **alle Methoden** des Interfaces in deiner Klasse definieren, selbst wenn du einige davon nicht benötigst. Dies ist der Standardweg und funktioniert gut, wenn das Interface nur wenige Methoden hat.

```
public interface AutoAktionen {
    void motorStarten();
    void beschleunigen();
    void bremsen();
    void hupen();
}
```

Wenn du dieses Interface implementierst:

```
public class ElektroAuto implements AutoAktionen {
    @Override
    public void motorStarten() {
        System.out.println("Elektromotor startet lautlos.");
    }

    @Override
    public void beschleunigen() {
        System.out.println("Elektroauto beschleunigt.");
    }

    @Override
    public void bremsen() {
        // Nicht benötigt
    }

    @Override
    public void hupen() {
        // Nicht benötigt
    }
}
```

Hier musst du auch die Methoden `bremsen()` und `hupen()` definieren, selbst wenn sie leer bleiben.

---

2. Interfaces mit Adapterklassen verwenden

Wenn ein Interface **viele Methoden** hat und du nur wenige davon benötigst, kannst du eine **Adapterklasse** verwenden. Diese Klasse implementiert das Interface und stellt leere Implementierungen für alle Methoden bereit. Dadurch kannst du nur die Methoden überschreiben, die du wirklich benötigst.

Wie funktioniert das?

1. Du erstellst eine **Adapterklasse**, die das Interface implementiert und alle Methoden leer lässt.
2. Du erstellst dann eine **spezifische Klasse**, die von der Adapterklasse erbt, und überschreibst nur die benötigten Methoden.

Adapterklasse:

```
public abstract class AutoAdapter implements AutoAktionen {
    @Override
    public void motorStarten() {}
    @Override
    public void beschleunigen() {}
    @Override
    public void bremsen() {}
    @Override
    public void hupen() {}
}
```

Spezifische Klasse:

```
public class ElektroAuto extends AutoAdapter {
    @Override
    public void motorStarten() {
        System.out.println("Elektromotor startet lautlos.");
    }

    @Override
    public void beschleunigen() {
        System.out.println("Elektroauto beschleunigt.");
    }
}
```

Wann benutzt man Adapterklassen?

Du benutzt Adapterklassen, wenn:

1. Ein **Interface viele Methoden** hat, aber du nur wenige davon verwenden möchtest.
2. Du Boilerplate-Code vermeiden möchtest (leere Methoden in deiner Klasse).

Beispiel: `MouseListener` **direkt vs. mit Adapterklasse**

Ohne Adapterklasse:

```
public class MausBeispiel implements MouseListener {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Maus geklickt.");
    }

    @Override
    public void mousePressed(MouseEvent e) {}
    @Override
    public void mouseReleased(MouseEvent e) {}
    @Override
    public void mouseEntered(MouseEvent e) {}
    @Override
    public void mouseExited(MouseEvent e) {}
}
```

Mit Adapterklasse (`MouseAdapter`):

```
public class MausBeispiel extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Maus geklickt.");
    }
}
```

Zusammenfassung: Wann was verwenden?

Direkte Verwendung des Interfaces	Adapterklasse verwenden
Wenn das Interface nur wenige Methoden hat.	Wenn das Interface viele Methoden hat.
Du benötigst alle oder fast alle Methoden des Interfaces.	Du benötigst nur einige wenige Methoden.
Kein zusätzlicher Codeaufwand für eine Adapterklasse.	Du sparst Boilerplate-Code für leere Methoden.

In der Praxis werden Adapterklassen vor allem bei älteren APIs (z. B. AWT/Swing) oder bei sehr großen Interfaces eingesetzt. In modernen Java-Versionen kannst du oft auch **Default-Methoden** in Interfaces verwenden, was den Bedarf an Adapterklassen verringert.