📓 Zusammenfassung: Abstrakte Klassen, Interfaces & bessere Abstraktionen

Dieses Kapitel stellt Werkzeuge vor, mit denen wir flexiblere und besser strukturierte Anwendungen entwerfen können. Besonders wichtig sind dabei `abstrakte Klassen`, `Interfaces` und `Simulationen` als Anwendungsfall für gute Abstraktionen.

◇ 1. Abstrakte Klassen

Eine `abstrakte Klasse` ist eine `Klasse`, die nicht direkt instanziert werden soll. Sie dient als `Superklasse`, von der konkrete Klassen erben.

Eigenschaften:

- Kann `abstrakte Methoden` enthalten (nur Signatur, kein Rumpf)
- Kann konkrete `Methoden` enthalten (mit Implementierung)
- Wird mit `abstract` markiert

Beispiel:

```
abstract class Tier {
    abstract void geraeusch();   // abstrakte Methode

    public void bewegen() {      // konkrete Methode
        System.out.println("Tier bewegt sich");
    }
}
```

◇ 2. Abstrakte Methoden

Haben nur einen Methodenkopf, keinen Rumpf.

Müssen in einer konkreten `Subklasse` zwingend überschrieben werden.

Beispiel:

```
abstract class Fahrzeug {
    public abstract void bewegeDich();
}
```

3. Konkrete und abstrakte Subklassen

Konkrete `Subklassen` müssen alle `abstrakten Methoden` implementieren.

Wenn eine Subklasse eine abstrakte Methode nicht implementiert → ist sie selbst abstrakt.

Beispiel:

```
class Auto extends Fahrzeug {

    @Override
    public void bewegeDich() {
        System.out.println("Auto faehrt");
    }
}
```

4. Interfaces

Ein `Interface` ist eine rein abstrakte Typdefinition.

Eigenschaften:

- Definiert Methodenköpfe, aber meistens keine Implementierungen
- Wird mit `interface` deklariert
- Eine `Klasse` kann mehrere `Interfaces` gleichzeitig implementieren (multiple Vererbung von Interfaces)
- `Interfaces` definieren `Typen`, die `Variablen` zugewiesen werden können

Beispiel:

```
interface Beweglich {
    void bewegeDich();
}

class Auto implements Beweglich {
    public void bewegeDich() {
        System.out.println("Auto faehrt");
    }
}
```

◇ 5. Multiple Vererbung (nur für Interfaces)

Java erlaubt keine multiple Vererbung von `Klassen`, aber:

Eine `Klasse` kann beliebig viele `Interfaces` implementieren

Komplexität entsteht, wenn mehrere `Interfaces Default-Methoden` definieren → Konflikte müssen manuell gelöst werden

◇ 6. Superklassen-Methodenaufrufe

Auch bei `abstrakten Klassen` gilt:

`Methoden` der `Superklasse` werden basierend auf dem `dynamischen Typ` ausgeführt.

▧ Beispiel: Abstrakte Klasse + Interface kombiniert

```
abstract class Tier {
    abstract void geraeusch();
}

interface Beweglich {
    void bewegen();
}

class Hund extends Tier implements Beweglich {
    @Override
    void geraeusch() {
    System.out.println("Wuff!");
    }

    @Override
    public void bewegen() {
        System.out.println("Hund rennt");
    }
}
```

Dieses Beispiel zeigt:

- abstrakte Klasse (Tier)
- Interface (Beweglich)
- konkrete Klasse (Hund)
- Implementieren abstrakter Methoden
- mehrere Typen für dasselbe Objekt (Tier, Beweglich, Hund)

🧠 Merksätze (Schweizer Rechtschreibung)

Abstrakte Klassen definieren gemeinsame Basis, aber keine vollständige Implementierung.

Abstrakte Methoden müssen in konkreten `Subklassen` implementiert werden.

`Interfaces` definieren Typen ohne Implementierung → ideal für flexible Entwürfe.

`Klassen` können nur eine `Superklasse`, aber viele `Interfaces` haben.

Gute Abstraktionen ermöglichen flexiblere, wiederverwendbare Software.