Zusammenfassung: Vererbung in Java

Dieses Kapitel führt in eines der wichtigsten Konzepte der objektorientierten Programmierung ein: `Vererbung`. Vererbung hilft dabei, Code zu strukturieren, zu vereinfachen und wiederzuverwenden.

◇ 1. Vererbung – Grundidee

Eine `Klasse` kann eine andere `Klasse` erweitern.

Die erweiterte Klasse heisst `Superklasse`, die neue Klasse `Subklasse`.

Subklassen erben:

- alle `Datenfelder`
- alle `Methoden` (ausser private)

`Objekt` einer `Subklasse` besitzt:

- eigene Eigenschaften & `Methoden` plus
- alle Eigenschaften & `Methoden` der `Superklasse`

Beispiel:

```
Auto ist ein Fahrzeug
Hund ist ein Tier
Student ist eine Person
```

Das nennt man eine **Ist-ein-Beziehung**.

◇ 2. Vererbungshierarchie

`Klassen` können über viele Stufen vererbt werden.

Beispiel:

```
Object → Fahrzeug → Auto → Elektroauto
```

Jede `Klasse` ohne explizite `Superklasse` erbt automatisch von Object.

3. Codewiederverwendung

Vererbung ermöglicht:

- kein doppelter Code (keine Duplizierung)
- gemeinsame Funktionen in der `Superklasse`
- spezifische Erweiterungen in der `Subklasse`

Beispiel:

```
class Fahrzeug {
    public void fahren() {
        System.out.println("Fahrzeug faehrt");
    }
}

class Auto extends Fahrzeug {
// erbt fahren()
}
```

4. Konstruktor der Superklasse

Eine `Subklasse` muss immer zuerst den Konstruktor ihrer `Superklasse` aufrufen.

```
class Auto extends Fahrzeug {
    public Auto() {
        super();  // muss als erste Zeile stehen
    }
}
```

Falls man keinen `super(...)`-Aufruf schreibt, fügt Java automatisch `super()` ein.

◇ 5. Subtypen & polymorphe Variablen

✔ `Subtyp`

Eine `Subklasse` definiert automatisch einen `Subtyp` ihrer `Superklasse`.

✔ `Polymorphe Variable`

Eine Variable vom Typ einer `Superklasse` kann Objekte von jeder `Subklasse` halten.

Beispiel:

```
Fahrzeug f = new Auto();          // erlaubt
Fahrzeug g = new Elektroauto();   // ebenfalls erlaubt
```

Das ist das Prinzip der Ersetzbarkeit:

Ein Objekt eines `Subtyps` kann überall eingesetzt werden, wo ein Objekt des `Supertyps` erwartet wird.

◇ 6. Verlust von Typinformation & Casting

Wenn ein Objekt über eine Variable vom `Supertyp` angesprochen wird, gehen spezifische Informationen nicht verloren, aber sie sind versteckt.

Beispiel:

```
Fahrzeug f = new Auto();
```

Die `Variable` weiss nur, dass es ein `Fahrzeug` ist → Zugriff nur auf `Methoden` von `Fahrzeug`.

Um wieder an Autotyp-spezifische `Methoden` zu kommen, braucht es ein Cast:

```
Auto a = (Auto) f;
```

◇ 7. Abstrakte Klassen (Grundidee)

- Eine `Klasse` kann abstrakt sein: Sie kann nicht instanziiert werden.
- Sie dient als Grundtyp für andere Klassen.
- Oft enthält sie abstrakte `Methoden`, die `Subklassen` definieren müssen.

Beispiel:

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

▧ Beispiel: Vererbung und Polymorphie

```
class Tier {
    public void geraeusch() {
        System.out.println("ein Tier macht ein Geraeusch");
    }
}

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

class Katze extends Tier {
    @Override
    public void geraeusch() {
        System.out.println("Miau!");
    }
}

public class Demo {
    public static void main(String[] args) {
        Tier t1 = new Hund();   // polymorph
        Tier t2 = new Katze();  // polymorph

        t1.geraeusch();  // Wuff!
        t2.geraeusch();  // Miau!
    }
}
```

🧠 Merksätze (Schweizer Rechtschreibung)

Vererbung: `Subklasse` erweitert `Superklasse`.

Ist-ein-Beziehung: Ein Auto ist ein Fahrzeug.

`Subklassen` erben `Methoden` & `Felder` ihrer `Superklassen`.

`Konstruktor` der `Superklasse` muss zuerst aufgerufen werden.

`Polymorphie`: `Variablen` vom `Supertyp` können `Subtypen` halten.

Ersetzbarkeit ermöglicht flexiblen, erweiterbaren Code.

Jede `Klasse` ohne `Superklasse` erbt automatisch von `Object`.