Vererbung und Kapselung in Java - Erklärungen mit Auto-Beispiel

Kapselung in Java

Kapselung ist ein zentrales Prinzip der objektorientierten Programmierung, das die Daten einer Klasse schützt, indem die Felder einer Klasse als `private` deklariert werden. Der Zugriff auf diese Felder erfolgt ausschließlich über **Getter**- und **Setter-Methoden**, die als öffentliche Schnittstelle bereitgestellt werden.

Vorteile der Kapselung:

- Kontrollierter Zugriff auf Daten.
- Verhindert ungewollte Änderungen an den internen Zuständen.
- Erleichtert spätere Änderungen, da die interne Implementierung unabhängig von der öffentlichen Schnittstelle geändert werden kann.

Vererbung in Java

Vererbung ermöglicht es einer Klasse (Unterklasse), die Eigenschaften und Methoden einer anderen Klasse (Oberklasse) zu übernehmen. Durch Vererbung können Unterklassen vorhandenes Verhalten wiederverwenden und erweitern, ohne die Oberklasse zu verändern.

Vorteile der Vererbung:

- Wiederverwendbarkeit von Code.
- Ermöglicht die Spezialisierung durch Hinzufügen neuer Funktionen in Unterklassen.
- Fördert eine übersichtliche Klassenhierarchie.

Beispiel: Vererbung und Kapselung kombiniert

Das folgende Beispiel zeigt die Kapselung und Vererbung anhand der Klassen `Auto` (Oberklasse) und `Sportwagen` (Unterklasse).

```
// Oberklasse Auto mit Kapselung
public class Auto {
   // Private Felder (Kapselung)
   private String farbe;
   private int maximaleGeschwindigkeit;
   private int momentGeschwindigkeit;

   // Konstruktor
   public Auto(String farbe, int maximaleGeschwindigkeit) {
      this.farbe = farbe;
      this.maximaleGeschwindigkeit = maximaleGeschwindigkeit;
      this.momentGeschwindigkeit = 0;
   }

   // Getter und Setter (Kapselung)
   public String getFarbe() {
      return farbe;
```

```
    }

    public void setFarbe(String farbe) {
        this.farbe = farbe;
    }

    public int getMaximaleGeschwindigkeit() {
        return maximaleGeschwindigkeit;
    }

    public void setMaximaleGeschwindigkeit(int maximaleGeschwindigkeit) {
        this.maximaleGeschwindigkeit = maximaleGeschwindigkeit;
    }

    public int getMomentGeschwindigkeit() {
        return momentGeschwindigkeit;
    }

    public void setMomentGeschwindigkeit(int momentGeschwindigkeit) {
        if (momentGeschwindigkeit <= maximaleGeschwindigkeit &&
momentGeschwindigkeit >= 0) {
            this.momentGeschwindigkeit = momentGeschwindigkeit;
        } else {
            System.out.println("Ungültige Geschwindigkeit!");
        }
    }

    // Methode zum Beschleunigen
    public void beschleunigen(int wert) {
        setMomentGeschwindigkeit(this.momentGeschwindigkeit + wert);
    }

    // Methode zum Bremsen
    public void bremsen(int wert) {
        setMomentGeschwindigkeit(this.momentGeschwindigkeit - wert);
    }
}
```

Unterklasse `Sportwagen` mit zusätzlicher Funktionalität

```
// Unterklasse Sportwagen, erbt von Auto
public class Sportwagen extends Auto {
    private boolean turboModus; // Eigenschaft für den Turbo-Modus

    // Konstruktor
    public Sportwagen(String farbe, int maximaleGeschwindigkeit) {
        super(farbe, maximaleGeschwindigkeit); // Konstruktor der Oberklasse wird
aufgerufen
        this.turboModus = false;
    }
```

```
    // Methode zum Aktivieren des Turbo-Modus
    public void aktiviereTurbo() {
        this.turboModus = true;
        System.out.println("Turbo-Modus aktiviert!");
    }

    // Methode zum Deaktivieren des Turbo-Modus
    public void deaktiviereTurbo() {
        this.turboModus = false;
        System.out.println("Turbo-Modus deaktiviert!");
    }

    // Überschreiben der Methode beschleunigen
    @Override
    public void beschleunigen(int wert) {
        if (turboModus) {
            wert *= 2; // Verdoppelt die Beschleunigung im Turbo-Modus
        }
        super.beschleunigen(wert); // Ruft die beschleunigen-Methode der Oberklasse
auf
    }
}
```

Hauptprogramm zur Demonstration

```
public class Hauptprogramm {
    public static void main(String[] args) {
        // Erstellen eines normalen Autos
        Auto auto = new Auto("Rot", 150);
        auto.beschleunigen(50);
        System.out.println("Normales Auto Geschwindigkeit: " +
auto.getMomentGeschwindigkeit());

        // Erstellen eines Sportwagens
        Sportwagen sportwagen = new Sportwagen("Blau", 250);
        sportwagen.beschleunigen(50);
        System.out.println("Sportwagen Geschwindigkeit: " +
sportwagen.getMomentGeschwindigkeit());

        // Turbo-Modus aktivieren und erneut beschleunigen
        sportwagen.aktiviereTurbo();
        sportwagen.beschleunigen(30);
        System.out.println("Sportwagen Geschwindigkeit nach Turbo: " +
sportwagen.getMomentGeschwindigkeit());
    }
}
```

Erklärung

1. **Kapselung**:
    - Die Felder `farbe`, `maximaleGeschwindigkeit` und `momentGeschwindigkeit` der Klasse `Auto` sind als `private` deklariert.
    - Der Zugriff erfolgt ausschließlich über Getter- und Setter-Methoden, wodurch kontrolliert wird, dass nur gültige Werte zugewiesen werden.
2. **Vererbung**:
    - Die Klasse `Sportwagen` erbt von `Auto` mit dem Schlüsselwort `extends`.
    - `Sportwagen` fügt die Eigenschaft `turboModus` und Methoden zur Steuerung des Turbo-Modus hinzu.
    - Die Methode `beschleunigen` wird in der Unterklasse überschrieben, um das Verhalten im Turbo-Modus zu ändern.
3. **Polymorphismus**:
    - Ein `Sportwagen` kann wie ein `Auto` behandelt werden, da er alle Eigenschaften der Oberklasse `Auto` erbt. Gleichzeitig erweitert er das Verhalten durch Turbo-spezifische Funktionen.

Vorteile der Kombination von Kapselung und Vererbung

- **Sichere Daten**: Durch Kapselung bleiben die Felder geschützt und nur über definierte Methoden zugänglich.
- **Wiederverwendbarkeit**: Gemeinsame Funktionen werden in der Oberklasse definiert, und die Unterklassen spezialisieren diese.
- **Flexibilität**: Änderungen in der Oberklasse wirken sich automatisch auf die Unterklassen aus, was den Wartungsaufwand reduziert.