Sichtbarkeiten in Java: `private`, `public`, `protected` und Default

In Java werden Zugriffssichtbarkeiten (Access Modifiers) verwendet, um zu kontrollieren, wo und wie auf Klassen, Methoden, Variablen oder Konstruktoren zugegriffen werden kann. Sie spielen eine zentrale Rolle beim **Encapsulation**-Prinzip (Datenkapselung).

---

1. Definition der Sichtbarkeiten

1. `public`

- **Definition:** Mitglieder oder Klassen mit `public` sind von überall im Programm zugänglich.
- **Vorteile:**
  - Ermöglicht den Zugriff auf Klassen oder Methoden aus beliebigen Packages.
  - Ideal für Methoden oder Klassen, die eine öffentliche API darstellen.
- **Beispiel:**

```
public class Auto {
    public String marke; // Von überall zugänglich

    public void fahren() {
        System.out.println(marke + " fährt.");
    }
}
```

---

2. `private`

- **Definition:** Mitglieder oder Methoden mit `private` sind nur innerhalb der gleichen Klasse zugänglich.
- **Vorteile:**
  - Schützt sensible Daten vor unbefugtem Zugriff.
  - Erzwingt, dass der Zugriff auf die Daten nur über kontrollierte Methoden (z. B. Getter/Setter) erfolgt.
- **Beispiel:**

```
public class Auto {
    private String motorTyp = "V8"; // Nur innerhalb der Klasse zugänglich

    private void zeigeMotor() {
        System.out.println("Motor: " + motorTyp);
    }

    public void start() {
        zeigeMotor(); // Zugriff innerhalb der Klasse erlaubt
```

```
        System.out.println("Das Auto startet.");
    }
}
```

---

3. `protected`

- **Definition:** Mitglieder mit `protected` sind:
  - Zugänglich innerhalb der gleichen Klasse.
  - Zugänglich für Unterklassen (auch in anderen Packages).
  - Zugänglich für Klassen im gleichen Package.
- **Vorteile:**
  - Ermöglicht den Zugriff auf wichtige Funktionalitäten in Unterklassen.
  - Bietet einen Mittelweg zwischen `private` und `public`.
- **Beispiel:**

```
package com.vehicles;

public class Auto {
    protected int baujahr; // Zugänglich in Unterklassen und im gleichen Package

    protected void reparieren() {
        System.out.println("Das Auto wird repariert.");
    }
}
```

Verwendung in einer Unterklasse:

```
package com.garage;

import com.vehicles.Auto;

public class Werkstatt extends Auto {
    public void wartung() {
        reparieren(); // Zugriff auf geschützte Methode
        System.out.println("Wartung abgeschlossen.");
    }
}
```

---

4. Default (ohne Modifikator)

- **Definition:** Wenn kein Modifikator angegeben wird, ist der Zugriff nur innerhalb des gleichen Packages möglich.
- **Vorteile:**

    - Nützlich für Klassen oder Methoden, die nur für andere Klassen im gleichen Package sichtbar sein sollen.
- **Beispiel:**

```
class Motor {
    void starten() {
        System.out.println("Der Motor startet.");
    }
}
```

---

2. Vorteile und Verwendung von Getter und Setter

Getter und Setter sind Methoden, die den Zugriff auf `private` Variablen kontrollieren. Sie bieten eine kontrollierte Möglichkeit, auf Eigenschaften einer Klasse zuzugreifen oder sie zu ändern.

Warum Getter und Setter verwenden?

1. **Datenkapselung:** Direkter Zugriff auf private Variablen wird verhindert.
2. **Validierung:** Setter können Eingabewerte validieren, bevor sie einer Variablen zugewiesen werden.
3. **Flexibilität:** Getter und Setter ermöglichen Änderungen in der Implementierung, ohne die öffentliche API zu verändern.

Beispiel: Getter und Setter

```
public class Auto {
    private String marke; // Private Variable

    // Getter für 'marke'
    public String getMarke() {
        return marke;
    }

    // Setter für 'marke'
    public void setMarke(String marke) {
        if (marke != null && !marke.isEmpty()) { // Validierung
            this.marke = marke;
        } else {
            System.out.println("Ungültige Marke.");
        }
    }

    public void anzeigen() {
        System.out.println("Auto-Marke: " + marke);
    }
}
```

Verwendung der Getter und Setter:

```
public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();
        auto.setMarke("BMW"); // Setter verwenden
        System.out.println("Marke: " + auto.getMarke()); // Getter verwenden
    }
}
```

---

3. Zusammenfassung der Zugriffssichtbarkeiten

Modifikator	Gleiche Klasse	Gleiches Package	Unterklassen (anderes Package)	Überall
private	✔	✗	✗	✗
Default	✔	✔	✗	✗
protected	✔	✔	✔	✗
public	✔	✔	✔	✔

---

4. Praxisbeispiel: Anwendung aller Sichtbarkeiten

```
package com.vehicles;

public class Auto {
    private String motorTyp = "V8";    // Nur innerhalb der Klasse zugänglich
    protected int baujahr;             // Zugänglich in Unterklassen und im gleichen
Package
    public String marke;               // Von überall zugänglich

    // Getter und Setter für die private Variable
    public String getMotorTyp() {
        return motorTyp;
    }

    public void setMotorTyp(String motorTyp) {
        this.motorTyp = motorTyp;
    }

    protected void reparieren() {
        System.out.println("Das Auto wird repariert.");
    }

    public void starten() {
        System.out.println(marke + " startet.");
    }
}
```

Verwendung in einer Unterklasse:

```
package com.garage;

import com.vehicles.Auto;

public class Werkstatt extends Auto {
    public Werkstatt(String marke, int baujahr) {
        this.marke = marke;  // Zugriff auf `public` Feld
        this.baujahr = baujahr;  // Zugriff auf `protected` Feld
    }

    public void wartung() {
        reparieren(); // Zugriff auf geschützte Methode
        System.out.println("Wartung für " + marke + " abgeschlossen.");
    }
}
```

Fazit

- `private`**:** Maximale Kapselung, Zugriff nur innerhalb der Klasse.
- `protected`**:** Ermöglicht Zugriff für Unterklassen und im gleichen Package.
- `public`**:** Zugriff von überall, für öffentliche Schnittstellen geeignet.
- **Default:** Zugriff nur innerhalb des gleichen Packages.

Getter und Setter helfen, Daten sicher zu kapseln und bieten eine flexible Kontrolle über die Eigenschaften einer Klasse.