Die vier Grundpfeiler der objektorientierten Programmierung (OOP)

Die objektorientierte Programmierung basiert auf vier zentralen Konzepten:

Abstraktion

- **Definition**: Abstraktion bedeutet, nur die wesentlichen Eigenschaften eines Objekts offenzulegen und unnötige Details zu verbergen.
- **Beispiel**: Ein Auto hat wesentliche Eigenschaften wie **Farbe**, **Geschwindigkeit** und **maximale Geschwindigkeit**, aber die interne Funktionsweise des Motors bleibt verborgen.

Kapselung (Encapsulation)

- **Definition**: Kapselung bedeutet, dass die internen Details einer Klasse vor externem Zugriff geschützt werden.
- **Umsetzung**: Attribute (Daten) werden als `private` deklariert, und der Zugriff erfolgt über öffentliche Getter- und Setter-Methoden.

Vererbung (Inheritance)

- **Definition**: Mit Vererbung können Klassen Eigenschaften und Methoden einer übergeordneten Klasse übernehmen.
- **Beispiel**: Ein **Elektroauto** kann von der allgemeinen Klasse **Auto** erben und zusätzliche Eigenschaften wie **Batteriestand** hinzufügen.

Polymorphismus

- **Definition**: Polymorphismus ermöglicht es, dieselbe Methode in verschiedenen Kontexten unterschiedlich zu verwenden.
- **Beispiel**: Ein Auto könnte eine **beschleunigen**-Methode haben, die unterschiedlich implementiert wird, je nachdem ob es ein **Elektroauto** oder ein **Benzinauto** ist.

Kapselung (Encapsulation) im Detail

Kapselung schützt die internen Details einer Klasse und macht sie für andere Teile des Codes unzugänglich. Stattdessen erfolgt der Zugriff nur über definierte Schnittstellen wie Getter und Setter.

Problematik ohne Kapselung – Beispiel: Auto-Klasse

Wenn alle Attribute einer Klasse öffentlich (`public`) sind, können sie von externen Klassen direkt verändert werden, was problematisch ist.

Beispiel: Auto-Klasse ohne Kapselung

```
class Auto {
    public String farbe;
    public int maximaleGeschwindigkeit;
    public int momentaneGeschwindigkeit;
}
```

Verwendung der Klasse

```
public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();

        // Direkte Änderung der Attribute
        auto.farbe = "Rot";
        auto.maximaleGeschwindigkeit = 150;
        auto.momentaneGeschwindigkeit = -50; // Problematisch!
    }
}
```

Probleme ohne Kapselung

1. **Ungültige Werte können gesetzt werden**:
    - `momentaneGeschwindigkeit` wird auf `-50` gesetzt, was physikalisch unmöglich ist.
    - Es gibt keine Validierung, um solche Änderungen zu verhindern.
2. **Kein Schutz vor inkorrekten Änderungen**:
    - Jemand könnte die `maximaleGeschwindigkeit` auf `0` setzen, was das Verhalten der Klasse bricht.
3. **Unvorhersehbares Verhalten**:
    - Andere Teile des Codes könnten versehentlich Attribute ändern, was schwer nachvollziehbare Bugs verursacht.
4. **Keine Kontrolle über die Daten**:
    - Es ist nicht möglich, zusätzliche Logik (z. B. Protokollierung) auszuführen, wenn sich ein Attribut ändert.

Lösung: Kapselung mit privaten Attributen

Die Attribute der Klasse **Auto** werden als `private` deklariert und sind von außen nicht direkt zugänglich. Der Zugriff erfolgt ausschließlich über Getter- und Setter-Methoden.

Beispiel: Auto-Klasse mit Kapselung

```
class Auto {
    private String farbe;
    private int maximaleGeschwindigkeit;
    private int momentaneGeschwindigkeit;

    // Getter und Setter für Farbe
    public String getFarbe() {
        return farbe;
    }

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

    // Getter und Setter für maximale Geschwindigkeit
    public int getMaximaleGeschwindigkeit() {
        return maximaleGeschwindigkeit;
    }

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

    // Getter und Setter für momentane Geschwindigkeit
    public int getMomentaneGeschwindigkeit() {
        return momentaneGeschwindigkeit;
    }

    public void setMomentaneGeschwindigkeit(int geschwindigkeit) {
        if (geschwindigkeit >= 0) {
            this.momentaneGeschwindigkeit = geschwindigkeit;
        } else {
            System.out.println("Fehler: Geschwindigkeit kann nicht negativ
sein!");
        }
    }
}
```

Vorteile der Kapselung

1. **Datenvalidierung**:
    - Setter können sicherstellen, dass nur gültige Werte gesetzt werden (z. B. keine negativen Geschwindigkeiten).
2. **Bessere Wartbarkeit**:
    - Änderungen an der internen Implementierung betreffen nur die Klasse, solange die öffentlichen Getter- und Setter-Methoden gleich bleiben.
3. **Zusätzliche Logik**:

- Getter und Setter können zusätzliche Aktionen ausführen, wie z. B. Protokollierung oder Benachrichtigungen.

4. **Schutz vor versehentlichem Zugriff**:
   - Externe Klassen können Attribute nicht direkt manipulieren, was die Robustheit des Codes erhöht.

Beispiel mit Kapselung

Verwenden der Auto-Klasse

```
public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto();

        // Werte mit Setter setzen
        auto.setFarbe("Rot");
        auto.setMaximaleGeschwindigkeit(150);
        auto.setMomentaneGeschwindigkeit(100);

        // Werte mit Getter abrufen
        System.out.println("Farbe: " + auto.getFarbe());
        System.out.println("Maximale Geschwindigkeit: " +
auto.getMaximaleGeschwindigkeit());
        System.out.println("Momentane Geschwindigkeit: " +
auto.getMomentaneGeschwindigkeit());

        // Ungültige Geschwindigkeit setzen
        auto.setMomentaneGeschwindigkeit(-20); // Gibt eine Warnung aus
    }
}
```

Fazit

Die Kapselung ist ein essenzielles Prinzip der OOP, das:

- **Daten schützt** und **Fehler verhindert**,
- die **Wartbarkeit** und **Erweiterbarkeit** des Codes verbessert,
- sicherstellt, dass nur **gültige Werte** den Attributen zugewiesen werden.

Durch private Attribute und kontrollierten Zugriff über Getter und Setter wird der Code robuster, sicherer und leichter verständlich.