„2. Open/Closed Principle (OCP) „

Das **Open/Closed Principle (OCP)** oder auf Deutsch das **Offen-Geschlossen-Prinzip** ist eines der grundlegenden Prinzipien der SOLID-Prinzipien in der objektorientierten Programmierung (OOP). Es besagt, dass **Software-Entitäten (wie Klassen, Module, Funktionen) für Erweiterungen offen, aber für Änderungen geschlossen sein sollten.**

Verwendung:

- Interfaces
- Abstrakte Klassen

Detaillierte Erklärung des OCP:

- Das Open/Closed Principle sagt, dass **bestehender Code nicht verändert werden sollte, wenn sich die Anforderungen ändern**, sondern dass der Code so gestaltet werden sollte, dass er **durch Hinzufügen von neuem Code erweitert** werden kann. Dies hilft, die Stabilität des Codes zu gewährleisten, denn Änderungen an bestehenden Klassen oder Modulen können oft zu unerwarteten Fehlern führen.

- Wenn der Code gut strukturiert ist und dem OCP folgt, können neue Anforderungen durch das **Erstellen neuer Klassen, Module oder Funktionen** erfüllt werden, ohne dass der bestehende Code modifiziert werden muss. Dies führt zu weniger Bugs und besser wartbarem Code.

Beispiel: Bankkonto:

Nehmen wir an, wir haben eine Klasse, die verschiedene Transaktionen auf einem Bankkonto verarbeitet:

```
class Bankkonto {
    public double balance;

    public Bankkonto(double balance) {
        this.balance = balance;
    }

    public void bearbeiteTransaktion(String transaktionsArt, double betrag) {
        if (transaktionsArt.equals("Einzahlung")) {
            balance += betrag;
        } else if (transaktionsArt.equals("Auszahlung")) {
            balance -= betrag;
        }
    }
}
```

In dieser Klasse gibt es eine Methode `bearbeiteTransaktion`, die zwischen zwei Arten von Transaktionen unterscheidet: **Einzahlung** und **Auszahlung**. Sie überprüft die Transaktionsart mit

einer Bedingung (in diesem Fall mit `if-else`), und je nach Ergebnis wird der Betrag entweder hinzugefügt oder abgezogen.

Problem:

Das funktioniert, aber was passiert, wenn wir eine neue Transaktionsart hinzufügen müssen, sagen wir **Überweisung**? Dann müssten wir die Methode `bearbeiteTransaktion` ändern, um eine neue Bedingung hinzuzufügen:

```
public void bearbeiteTransaktion(String transaktionsArt, double betrag) {
    if (transaktionsArt.equals("Einzahlung")) {
        balance += betrag;
    } else if (transaktionsArt.equals("Auszahlung")) {
        balance -= betrag;
    } else if (transaktionsArt.equals("Überweisung")) {
        balance -= betrag;  // für eine Überweisung ziehen wir den Betrag ab
    }
}
```

Das ist ein klarer **Verstoß gegen das OCP**, weil wir den **bestehenden Code ändern müssen**, um die neue Funktion hinzuzufügen. Jede zukünftige Erweiterung wird erfordern, dass wir den Code immer wieder ändern – und das kann zu Fehlern führen.

Lösung nach dem Open/Closed Principle:

Um das OCP zu erfüllen, sollten wir die Klasse so gestalten, dass sie für Erweiterungen offen ist, aber keine Modifikationen am bestehenden Code erfordert. Ein Weg, dies zu erreichen, ist die **Verwendung von Vererbung(Abstrakte Klassen) oder Schnittstellen (Interfaces)**, sodass wir die Funktionalität erweitern können, ohne den bestehenden Code zu ändern.

```
interface Transaktion {
    void bearbeite(Bankkonto konto, double betrag);
}

class Einzahlung implements Transaktion {
    public void bearbeite(Bankkonto konto, double betrag) {
        konto.balance += betrag;
    }
}

class Auszahlung implements Transaktion {
    public void bearbeite(Bankkonto konto, double betrag) {
        konto.balance -= betrag;
    }
}

class Überweisung implements Transaktion {
    public void bearbeite(Bankkonto konto, double betrag) {
        konto.balance -= betrag;  // Beispiel für Überweisung
    }
}

class Bankkonto {
    public double balance;

    public Bankkonto(double balance) {
        this.balance = balance;
    }

    public void bearbeiteTransaktion(Transaktion transaktion, double betrag) {
        transaktion.bearbeite(this, betrag);
    }
}
```

Erklärung:

- Jetzt haben wir eine **Schnittstelle `Transaktion`**, die eine Methode `bearbeite` hat. Jede neue Transaktionsart (wie **Einzahlung**, **Auszahlung**, **Überweisung**) implementiert diese Schnittstelle auf ihre eigene Weise.
- Die Methode `bearbeiteTransaktion` in der Klasse `Bankkonto` ändert sich nicht, egal wie viele neue Transaktionsarten wir hinzufügen. Sie arbeitet einfach mit dem abstrakten Typ `Transaktion`, und wir können beliebig viele neue Transaktionsarten durch **Erstellen neuer Klassen** hinzufügen, ohne den bestehenden Code zu ändern.

Warum ist das OCP wichtig?

1. **Stabilität und weniger Fehler:** Wenn wir bestehenden Code nicht ändern müssen, um neue Funktionalität hinzuzufügen, vermeiden wir potenzielle Fehler, die durch Änderungen in funktionierendem Code entstehen könnten. Änderungen am funktionierenden Code sind riskant und führen oft zu unerwarteten Problemen.
2. **Bessere Erweiterbarkeit:** Wenn das System so gestaltet ist, dass es leicht erweitert werden kann, ohne bestehende Module oder Klassen zu ändern, wird das Hinzufügen neuer Funktionalität einfacher und schneller. Dies führt zu einer flexibleren und robusteren Softwarearchitektur.
3. **Klare Verantwortlichkeiten:** Da der bestehende Code nicht geändert wird, bleibt jede Klasse oder Komponente in ihrer ursprünglichen Funktion bestehen. Neue Anforderungen führen zu neuen Klassen oder Modulen, was den Code lesbarer und verständlicher macht.

Zusammenfassung:

- Das **Open/Closed Principle (OCP)** besagt, dass Software-Entitäten **für Erweiterungen offen**, aber **für Änderungen geschlossen** sein sollten.
- **Bedeutung:** Der bestehende Code sollte **nicht geändert** werden müssen, wenn neue Funktionalität hinzugefügt wird. Stattdessen sollte die Funktionalität durch das Hinzufügen neuer Klassen oder Module erweitert werden.
- **Vorteile:** Dies führt zu **stabilerem, weniger fehleranfälligem** Code, der leichter zu **erweitern** und zu **warten** ist.