6. Appendix: die hohe Kunst der OOP

In den bisherigen Beispielen und Übungen haben wir immer die naheliegenden Lösungen für das Design von Beziehungen gewählt. So haben wir z.B. die Spezialisierung von `Kunde` und `Mitarbeiter` als Vererbung einer Oberklasse `Person` realisiert.

//Abb 6.11: //Vereinfachte, naheliegende Vererbungshierarchie

Diese Sichtweise hat aber Nachteile. So kann eine Person nicht gleichzeitig Kunde und Mitarbeiter sein. Wenn man den Sachverhalt genauer betrachtet, erkennt man auch, dass `Kunde` und `Mitarbeiter` ja nicht wirklich eine Spezialiserungen von `Person` darstellen, sondern `Rollen` sind, die eine Person einnimmt. Dass der vereinfachte Ansatz problematisch ist, fällt dann auf, wenn wir z.B. eine weiter Spezialisierung wie `Lieferant` zufügen.

Eine verbesserte Lösung im Sinne guten OO-Designs würde also wie folgt aussehen:

//Abb. 6.12: //Optimiertes Design mit einer Klasse Role (Rolle)

Was wollen wir Ihnen damit sagen?

Objektorientierung ist ein weites Feld. Es braucht ganz viel Erfahrung, um alle wichtigen Aspekte zu kenne und zu nutzen. So haben wir z.B. das Thema der Design-Patterns in keiner Weise angesprochen. Hier ein kleines Beispiel dazu.

Wenn Ihr Programm auf eine Ressource wie z.B. eine Datenbank zugreifen soll, so muss das „geordnet“ erfolgen. Sie können nicht an x-beliebigen Stellen in Ihrem Code DB-Zugriffsobjekte erzeugen. Das gleiche gilt auch für Kommunikationsschnitstellen. Wie aber kann man sicherstellen, dass es von einem Objekt immer nur eine Instanz gibt?

Dazu dient das Design-Pattern des `Singleton`.

//Abb. 6.13: //Singleton-Pattern

Aber was macht denn dieses `Singleton`? Ein Singleton fällt vorerst dadurch auf, dass sein Konstruktor private deklariert ist. Man kann also von ausserhalb der Klasse den Konstruktor gar nicht aufrufen.

Und wie erzeugt man nun eine Instanz dieser Klasse? Man ruft dazu die statische Methode `getInstance()` auf. Diese liefert als Ergebnis die Referenz auf das Singleton-Objekt und zwar immer ein und dieselbe Referenz.

Schauen wir uns dazu ein Stück Code an. Wir verwenden die Sprache Java, da sie in vielen Belangen die OO-spezifischen Gegebenheiten besser implementiert als Python.

```
public class Singleton{
    private Singleton instance;

    private Singleton(){
      // hier folgt eine allfällige Initialisierung von Attributen usw.
    }

    /**
       Liefert die Referenz auf das Singleton-Objekt.
       Das Objekt wird einmalig erzeugt. Bei jedem weiteren Aufruf
       wird die schon existierende Referenz geliefert.
    **/
    public Singleton getInstance(){
       if instance == null{
          instance = new Singleton()
```

```
        }
        return instance;
    }
}
```

Benötigt man eine Instanz der Klasse, wird das wie folgt bewerkstelligt.

```
  Singleton aSingleReferenceToWhatEver = Singleton.getInstance();
```

Und dieser Code kann an mehreren Orten einer Applikation implementiert sein. Es wird immer ein und dieselbe Referenz geliefert. D.h. dass alle Zugriff auf dieses eine Objekt erfolgen werden.
Die Kenntnis und Anwendung von Design-Patterns darf ohne Zweifel als die hohe Schule der OO-Programmierung bezeichnet werden. Sie haben also nich viel zu lernen.

© René Probst

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/archiv/m320/learningunits/lu06/theorie/lu07-kapitel_6?rev=1788091602

Last update: **2026/08/30 14:06**