**Das Objektorientierte Paradigma in Java**

*Dokumentation begleitend zu den praktischen Beispielen*

**Abstraktion:**

***„****Abstraktion ist ein theoretisches Konzept zur Beschreibung von Objekten und Klassen.****“***

Weil ein Objekt sowohl Daten als auch Methoden beinhaltet, kann die Abstraktion sowohl die Daten selbst als auch deren Attribute konsistent behandeln. Die abstrakte Beschreibung von Objekten und Klassen dient im Wesentlichen zur begrifflichen Definition derselben.

Sprich, eine Klasse kann sowohl vollständige Methoden als auch nur Signaturen dieser enthalten, also ohne eigentliche Implementierung. Abstrakte Klassen/Methoden sind quasi ein Zeichen, ‚dass hier eine Methode vorhanden ist, die es eigentlich nicht gibt‘. Eine sinnvolle Einsetzung findet bei der Vererbung statt, wo diese mitvererbt und implementiert werden (können).

**Kapselung:**

***„****The physical location of features (properties, behaviors) into a single black box abstraction that hides their implementation behind a public interface.****“***

(Daten-) Kapselung ist auch bekannt unter dem Namen ‚data hiding‘ oder auch Geheimnisprinzip, und ermöglicht Daten oder Informationen vor dem Zugriff von außen zu schützen. Die direkte Kommunikation erfolgt über eine genau festgelegte Anwendungsschnittstelle, und kann daher als ‚Black Box‘ bezeichnet werden.

Durch die Access Modifier

* + / public, von überall aus zugreifbar (auch andere Klassen)
* - / private, kann nur von der eigenen Klasse aus verwendet werden
* # / protected, für Subklassen
* ~ / default (od. no modifier) kann nur aus dem eigenen Package zugegriffen werden

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modifier | Klasse | Package | Subklasse | Welt |
| public | Ja | Ja | Ja | Ja |
| private | Ja | Nein | Nein | Nein |
| protected | Ja | Ja | Ja | Nein |
| default | Ja | Ja | Nein | Nein |

Das Prinzip des *Information Hiding* besagt, dass …

Anwendern nur die Informationen zur Verfügung stehen sollen, die zur Anwendungsschnittstelle gehören, sodass alle anderen Informationen für ihn verborgen und nicht zugreifbar sind.

Dies hat folgende Gründe:

* Vermeiden unsachgemäßer Anwendung durch den User/Manipulation
* Reduktion der Abhängigkeiten in der Software
* Austausch von Teilen der Implementierung

*Einfaches Beispiel:*

In den unteren Jahrgängen in Softwareentwicklung wurde uns diese Herangehensweise mithilfe einer Kokosnuss erklärt, ohne die richtigen Werkzeuge (hier: Parameter) gibt es keinen Weg ins Innere, wo die leckere Kokosmilch (hier: Variablen, Informationen, Daten) verborgen ist.

**Vererbung:**

***„****Die Vererbung (inheritance) in der objektorientierten Programmierung ermöglicht es, neue Klassen aus bereits existierenden Klassen abzuleiten.****“***

Einige wichtige Stichworte:

* Inheritance (die Vererbung)
* derived class (die abgeleitete Klasse)
* base class (Basisklasse)
* subclass (Unterklasse)
* superclass (Oberklasse)

Bei einer abgeleiteten Klasse, die dank der Vererbung alle Funktionen ihres sogenannten Elternteils hat ohne Manipulation der Implementierung, müssen nur Funktionen hinzugefügt werden um diese Klasse zusätzlich zu erweitern, jedoch können auch geerbte Methoden überschrieben werden.

Ein ‚Elternobjekt‘ beschreibt, wenn auch unvollständig, prinzipiell das Verhalten seiner ‚Kinder‘.

Vererbung spart vor allem eines: *Schreibaufwand*, sprich redundanten Code

Außerdem wird ein späteres Nachbessern im Code des Elternteils vereinfacht, weil der geänderte Code sich somit auf alle geerbten Klassen auswirken kann. Wie oben erwähnt, gibt es zwei Möglichkeiten, um ein ‚Kind‘ einer Klasse zu erweitern, hier näher beschrieben:

*Erweitern:*

Erweitert die Oberklasse um neue Variablen, Methoden sowie Konstruktoren.

*Überschreiben:*

Methoden der Oberklasse werden durch neue Methoden überschrieben, die diejenige dann ersetzen. Meist gibt es zusätzlich eine Routine, auf die Ursprungs-Klasse der Oberklasse zuzugreifen.

Diese Möglichkeiten sind beliebig kombinierbar.

In Java lässt sich mit dem Ausdruck ‚extends Klassenname‘ verwirklichen, es kann immer nur von einer einzelnen Klasse geerbt werden.

**Polymorphie:**

***„****Polymorphismus (griech. Vielgestaltigkeit) bezeichnet die Möglichkeit, gleichnamige Methodenbezeichner bei unterschiedlicher und/oder gleicher Funktionalität in verschiedenen Klassen zu verwenden, wobei der Aufruf der gleichnamigen Methoden differieren kann.****“***

In der Regel dient sie dazu, einem Methodenbezeichner erst zur Laufzeit die konkrete Ausprägung der Methode mitzuteilen und dadurch die Mehrdeutigkeit ähnlicher Prozesse beschreiben als auch steuern zu können. Diese späte Zuweisung nennt man daraus resultierend auch ‚late binding‘ oder dynamisches Binden. Auch Polymorphismus kann polymorph sein, sowie ist zu bedenken, dass sie nicht zwangsläufig auch Abstraktion beinhalten muss.

*Universelle Polymorphie – Generizität:*

Auch parametrisierte Polymorphie genannt, eine Methode soll dynamisch für Daten verschiedener Typen einsetzbar sein. Durch formale Typparameter kann diese Vielgestaltigkeit erreicht werden.