Inhaltsverzeichnis

[1. Objektorientierung 1](#_Toc66641376)

[2. Was ist Objektorientierte Programmierung? 1](#_Toc66641377)

[3. Ursprung 1](#_Toc66641378)

[4. Kernbestandteile der OOP 1](#_Toc66641379)

[4.1. Klassen und Objekte 1](#_Toc66641380)

[4.2. Methoden 2](#_Toc66641381)

[4.3. Konstruktoren 3](#_Toc66641382)

[4.4. Klassenvariablen und Klassenmethoden 4](#_Toc66641383)

[4.5. Vererbung 4](#_Toc66641384)

[4.6. Abstrakte Klassen 5](#_Toc66641385)

[4.7. Interfaces 5](#_Toc66641386)

[4.8. Modifizierer 6](#_Toc66641387)

[5. Fazit 7](#_Toc66641388)

[6. Literaturverzeichnis 8](#_Toc66641389)

# Objektorientierung

In dieser Hausarbeit wird das Thema Objektorientierung behandelt, dazu wird zunächst darauf Eingegangen was Objektorientierung überhaupt ist, danach springen wir in die Zeit zurück und schauen uns die Anfänge der Objektorientierten Programmierung an. Anschließend wird nochmal im Detail eingegangen was die Kernbestandteile der Objektorientierten Programmierung sind.

# Was ist Objektorientierte Programmierung?

„Die objektorientierte Programmierung (kurz OOP) ist ein auf dem Konzept der Objektorientierung basierendes Programmierparadigma. Die Grundidee besteht darin, die Architektur einer Software an den Grundstrukturen desjenigen Bereichs der Wirklichkeit auszurichten, der die gegebene Anwendung betrifft. Ein Modell dieser Strukturen wird in der Entwurfsphase aufgestellt. Es enthält Informationen über die auftretenden Objekte und deren Abstraktionen, ihre Typen. Die Umsetzung dieser Denkweise erfordert die Einführung verschiedener Konzepte, insbesondere Klassen, Vererbung, Polymorphie und spätes Binden (dynamisches Binden).“ (Objektorientierte Programmierung 2021)

# Ursprung

Aufgrund von Fehlender Flexibilität, schwieriger Lesbarkeit und aufwändiger Wartung kam die Überlegung, wie man dies verbessern könne. Mit Simula 67 kamen schon die ersten Ansätze der objektorientierten Programmierung. Wenig später kam Smalltalk, welches die Konzepte verfeinerte und konsequenter umsetzte. Dadurch kamen immer mehr sogenannte hybride Programmiersprachen. Dabei wurden bestehende Programmiersprachen genommen und das objektorientierte Programmierkonzept wurde versucht mit einzubauen. Dabei hatte C++ in den 1980er Jahren seinen Durchbruch erlangt, welche heute immer noch sehr beliebt ist. 1990 wurde die Programmiersprache Java von Microsystems entwickelt, welche eine reine objektorientierte Programmiersprache ist, zudem ist sie auch plattformunabhängig und kann somit ohne große Anpassung auf verschiedenen Betriebssystemen sowie Computer laufen. Das liegt daran das Java in einer Virtuellen Maschine läuft, welche den Bytecode ausführt. (vgl. OOP und UML o.D.)

# Kernbestandteile der OOP

Um die Objektorientierung und deren Grundlagen besser zu verstehen, wird zunächst auf die wesentlichen Begrifflichkeiten der objektorientierten Programmierung eingegangen.

# Klassen und Objekte

„Java-Programme bestehen aus Klassendefinitionen. Eine Klasse ist eine allgemeingültige Beschreibung von Dingen, die in verschiedenen Ausprägungen vorkommen können, aber alle eine gemeinsame Struktur und ein gemeinsames Verhalten haben. Sie ist ein Bauplan für die Erzeugung von einzelnen konkreten Ausprägungen. Diese Ausprägungen bezeichnet man als Objekte oder Instanzen der Klasse.“ (Abst 2015, S. 35)

Methoden, Konstruktoren und Variablen die auch Attribute genannt werden sind Bestandteile einer Klasse. In den Attributen können Werte gespeichert werden, welche den Zustand eines Objektes aussagen. In den Methoden wird der ausführbare Code reingeschrieben. Diese Methoden befinden sich in einer Klasse und beschreiben somit das Verhalten. (vgl. Abst 2015, S. 36)

„Der allgemeine Aufbau einer Klasse ist:

[Modifizierer] class Klassenname [extends Basisklasse][implements Interface-Liste] {

Attribute

Konstruktoren

Methoden

...

}“ (Abst 2015, S. 35)

Im obigen Beispiel sehen wir die Grundbestandteile einer Klasse und deren Bezeichnung. Dabei ist das was in den Eckigen Klammer angegeben ist nur optional und muss nicht angegeben werden. Eine Klassendefinition befindet sich in einer Quelldatei, dabei sollte sich nur eine Klassendefinition darin befinden. Wenn aber mehrere vorhanden sind, darf höchstens eine Klasse den Modifizierer public haben. Wenn diese Klasse den Modifizierer public hat, muss diese den gleichen Namen wie der von der Datei haben. Groß- und Kleinschreibung ist dabei auch zu beachten. Um mit einer Klasse zu arbeiten, müssen wir zunächst ein Objekt von der Klasse erzeugen. Dafür legen wir eine neue Variable an, die von dem Typ der Klassen ist, anschließend muss noch mit dem Operator new das Objekt zugewiesen werden. Dabei enthält die Variable selbst nur einen Adressverweis auf den Speicherplatz, dies wird auch Referenzvariable genannt. Wenn wir auf ein Attribute oder eine Methode von einem Objekt zugreifen wollen, müssen wir die Punktnotation verwenden. Dazu geben wir zuerst unsere Referenz an, danach mit einem Punkt getrennt unser Attribute oder die Methode.

(vgl. Abst 2015, S. 36-40)

# Methoden

„Operationen auf Objekten werden mit Methoden realisiert. Innerhalb einer Methode können die Attribute und andere Methoden der eigenen Klasse ohne ausdrückliche Objektreferenz benutzt werden.

[Modifizierer] Rueckgabetyp Methodenname([Parameter-Liste]) [throws Exception-Liste] {

Anweisungen

}“ (Abst 2015, S. 40)

Um Methoden zu verwenden, brauchen wir zuerst ein Hintergrundwissen, wie diese Aufgebaut sind. Eine Methode besteht aus dem Methodenkopf und dem Methodenrumpf, welcher die Implementierung beinhaltet. Zudem kann eine Methode einen Wert zurückgeben. Dieser Rückgabewert, kann ein einfacher Datentyp oder eine Referenz sein. Wird die return Anweisung ausgeführt, wird die Methode anschließend beendet. Wenn eine Methode keinen Rückgabetyp hat, handelt es sich um eine void Methode. Im Methodenkopf werden die Parameter in runden Klammern angegeben, wenn welche angegeben sind, müssen diese beim Aufruf der Methode übergeben werden. Erst kommt der Datentyp und danach der Name, mehrere Argumente werden mit Komma getrennt. Wenn wir einen Wert übergeben, wird innerhalb dieser Methode ein neuer Speicherplatz erzeugt, da nur der Wert kopiert wird. Dies wird auch call by value genannt. Das hat den Vorteil, dass unsere Variablen nach den Methodenaufruf nicht verändert werden. Wenn aber hingegen ein Objekt übergeben, wird hier nur eine Referenz übergeben, da eine Kopie von einem Objekt nur ein Verweis auf dieses Objekt ist. Dies wird call by reference bezeichnet.

Wenn Variablen innerhalb einer Methode, oder innerhalb eine Blockes definiert, so werden diese Lokale Variablen bezeichnet. Sie werden erst angelegt, wenn die Methode oder der Block beginnt. Nach dem beenden einer Methode oder Verlassen eines Blockes, werden diese zerstört. Beginnt eine Deklaration mit final, sprechen wir von einer Konstante. Hier können wir nur bei der Initialisierung oder beim ersten Zugriff, einen Wert zuweisen.

Wenn wir sichergehen wollen, dass wir auf ein Objekt welches sich in unserer Klasse befindet, zugreifen wollen, verwenden wir this. Ein Beispiel wäre hier eine setter Methode:

public void setJahr(int jahr) {

this.jahr = jahr;

}

Hier wird von der lokalen Variable das Attribute unserer Klasse verdeckt. Mit this können wir dies aber umgehen und greifen so direkt auf unser Attribut zu.

Eine Signatur wird aus dem Namen der Methode und deren Parameterliste gebildet. Dabei gehört der rückgabetyp nicht dazu. Wenn wir nun eine Methode aufrufen wollen, benötigen wir solch eine Signatur. Dabei geben wir nur den Methodennamen an und in den runden Klammern unsere Parameter.

Es können auch mehrere Methoden denselben Namen haben. Dafür muss aber jede Methode die den gleichen Namen trägt unterschiedliche Signaturen haben. Dies wird Überlagern genannt. Ein Beispiel könnte folgendes sein:

public array getField() {

return this.field;

}

public int getField(int y, int x) {

return this.field[y][x];

}

Wobei wir im ersten Fall das gesamte Feld zurückgeben und im zweiten nur einen bestimmten Eintrag. (vgl. Abst 2015, S. 40-43)

# Konstruktoren

Bei einem Konstruktor handelt es sich um eine Spezielle Methode, denn diese werden bei der Erzeugung eines Objektes aufgerufen. Dieser wird wie eine normale Methode deklariert, nur das hier der Konstruktor genauso benannt wird, wie die Klasse selbst. In der Regel werden sie für die Initialisierung verwendet.

Ist kein Konstruktor in einer Klasse angegeben, wird der Standardkonstruktor ausgeführt. Ein Konstruktor kann auch einen anderen Konstruktor mit this aufrufen. Dabei muss die Anweisung als erstes erfolgen.

Im Stack werden die Variablen verwaltet und im Heap liegen die erzeugten Objekte. Um diesen Speicher zu verwalten, wird der Garbage Collector verwendet. Dieser gibt automatisch den Heap frei, wenn dieser nicht mehr gebraucht wird.

(vgl. Abst 2015, S. 44-46)

# Klassenvariablen und Klassenmethoden

Die Klassenvariablen tragen das Schlüsselwort „static“ und werden beim Laden der Klasse erzeugt. Dazu kann von einer Variablen nur ein Exemplar existieren, unabhängig von der Anzahl der erzeugten Instanzen. Wenn Attribute verwendet werden die kein „static“ haben, reden wir von Instanzvariablen. Möchten wir von außerhalb der Klasse auf die Klassenvariable zugreifen, müssen wir die Punktnotation verwenden. Dazu geben wir zuerst den Klassennamen an und danach einen Punkt und jetzt den Variablennamen. Andernfalls können wir auch über das angelegte Objekt von der Klasse darauf zugreifen.

Ähnlich sieht es auch bei den Methoden aus. Wenn eine Methode mit dem Schlüsselwort „static“ versehen wurde, handelt es sich um eine Klassenmethode. Diese können wiederum von außen über den Klassennamen aufgerufen werden. Wenn das „static“ wegfällt, handelt es sich um eine Instanzmethode. Zudem dürfen Klassenmethoden nur auf Klassenvariablen und Klassenmethoden zugreifen.

Die Main Methode ist der Startpunkt des Programmes.

(vgl. Dietmar Abst 2015, S. 44-48)

# Vererbung

Bei der Vererbung können neue Klassen aus bereits vorhandenen Klassen erben, dies wird ermöglicht mit dem Schlüsselwort „extends“. Welcher im Kopf der Klassenbeschreibung zusammen mit der Klasse, von der abgeleitet werden soll, angegeben wird. Die neu erzeugte Klasse wird als Subklasse bezeichnet und die andere als Superklasse. Die Subklasse kann auf alle Attribute und Methode der Superklasse zugreifen, mit Ausnahme von private gekennzeichneten Elementen. Zudem kann die Subklasse die Methoden auch überschreiben. Dabei bleiben der Name und die Parameterliste unverändert. Das einzige was sich ändert, sind die Anweisungen im Methodenrumpf.

Ein einfaches Beispiel wäre:

public class Rechteck {

public double getArea() {

return a\*b

}

}

public class Quadrat extends Rechteck{

@Override

public double getArea() {

return a^2

}

}

Hier erbt Quadrat von Rechteck und Quadrat spezialisiert die Flächenberechnung da, ein Rechteck zwei unterschiedlich lange Seiten hat und ein Quadrat vier gleich lange Seiten.

Dazu dürfen Instanzmethoden nur durch Instanzmethoden überschrieben werden, sowie Klassenmethoden durch Klassenmethoden.

Konstruktoren können wiederum nicht vererbt werden. Stattdessen kann die Subklasse mittels dem super Befehl, den Konstruktor aus der Superklasse aufrufen, oder mit this einen anderen Konstruktor aus der Subklasse aufrufen. Die super Anweisung muss aber als erstes angegeben werden.

Zudem kann eine Klasse höchstens eine Superklasse haben. Wenn eine Klasse nicht explizit von einer anderen Erbt, wird von Java automatisch von einer definierten Klasse abgeleitet.

Mit instanceof kann festgestellt werden zu welcher Klasse ein bestimmtes Objekt gehört. Wenn man also eine Referenzvariable mit einer Klasse überprüfen möchte, ob dies auf ein Objekt der Klasse oder Subklassen verweist. Kann man dies damit überprüfen.

(vgl. Abst 2015, S. 49-54)

# Abstrakte Klassen

Klassen die mit dem Schlüsselwort abstract beginnen, werden Abstrakte Klassen genannt. Hier dürfen nur abstrakte Methoden erstellt werden. Zudem besitzen sie kein Methodenrumpf und müssen von der Subklasse überschrieben werden. Andernfalls sind die abgeleiteten Klassen auch abstrakt und müssen mit dem Schlüsselwort abstract gekennzeichnet werden. Außerdem dürfen Abstrakte Methoden nicht private sein, da sie sonst nicht überschrieben werden können. Aber wofür brauchen wir Abstrakte Klassen? Sie sind dann hilfreich, wenn schon ein Teil in der Abstrakten Klasse implementiert wurde und der Subklasse überlassen wird, spezielle Methoden zu Implementieren. (vgl. Abst 2015, S. 55)

“Die Eigenschaft, dass der formal gleiche Aufruf einer Methode [...] unterschiedliche Reaktionen (je nach adressiertem Objekt) auslösen kann, wird als Polymorphie (Vielgestaltigkeit) bezeichnet.” (Abst 2015, S. 56)

# Interfaces

„Interfaces sind Schnittstellenbeschreibungen, die festlegen, was man mit der Schnittstelle machen kann. [..] Sie haben den folgenden Aufbau:

[public] interface Interfacename [extends Interface-Liste] {

Konstanten

abstrakt Methoden

geschachtelte Klassen und Interfaces

}

Die in eckigen Klammern angegebenen Einheiten sind optional. Interfacenamen unterliegen den gleichen Namenskonventionen wie Klassennamen. Die Konstanten haben implizit die Modifizierer public, static und final. Die Methoden sind alle implizit public und abstract. Diese automatisch vergebenen Modifizierer sollten nicht explizit gesetzt werden. Geschachtelte Klassen sind automatisch public und static.“ (Abst 2015, S. 59)

Im Gegensatz zu Klassen können Interfaces mehrere Interfaces abgeleitet werden. Dadurch können mehrere Schnittstellenbeschreibungen in einem zusammengefast werden.

Klassen können auch mehrere Interfaces implementieren, dazu wird das Schlüsselwort implements benötigt, welche mit Komma getrennt werden. Die Implementierung kann auch durch eine Superklasse geerbt werden. Es ist nicht möglich zwei Interfaces welche dieselbe Signatur haben, aber dessen Rückgabetyp unterschiedlich, zu implementieren.

(vgl. Abst 2015, S. 59-60)

„Polymorphie lässt sich auch mittels Interfaces realisieren. Eine Klasse, die ein Interface implementiert, "erbt" nur die Methodensignatur (inkl. Rückgabetyp) und ist typkompatibel zum Interface. In einem Programm kann die Klasse leicht gegen eine andere, die dasselbe Interface implementiert, ausgetauscht werden. Die Klasse ist also nur lose an das Interface gekoppelt.“ (Abst 2015, S. 62)

Anders sieht es aus wenn die Subklasse eng mit der Superklasse gekoppelt ist. Da hier schon eine kleine Änderung im Code von der Superklasse gravierende auswirkunken haben kann.

Bei einem Interfaces und Abstrakten Klassen lassen sich abstrakte Methoden implementieren, aber was ist der Unterschied zwischen einem Interface und einer Abstrakten Klasse. Bei abstrakten Klassen können Konstruktoren und Instanzvariablen definiert werden, was bei Interfaces nicht geht.

(vgl. Abst 2015, S. 62-66)

# Modifizierer

Mithilfe von Modifizieren können bestimmte Eigenschaften von Klassen, Attributen und Methoden festgelegt werden. Zugriffsrechte werden mittels public, private und protected gesteuert. Wenn eine Klasse mit public gekennzeichnet wird, kann diese von überall verwendet werden. Gleiches gilt auch für Attribute, Methoden und Konstruktoren. Dafür muss aber zuerst ein Objekt von der Klasse erzeugt werden. Zudem darf nur höchstens eine public Klasse in einer Quelldatei deklariert werden. Auf Protected gekennzeichneten Attributen, Methoden und Konstruktoren kann nur in derselben Klasse und allen Klassen die sich in demselben Paket befinden, zugegriffen werden. Zudem können sie auch vererbt werden, wenn sie sich im selben Paket befinden. Die Zugriffsrechte von Methoden dürfen nicht reduziert werden. Das heißt, dass eine public methode nur mit public überschrieben werden kann und eine protected Methode kann mit protected oder public überschrieben werden. Bei Attributen, Methoden und Konstruktoren die private gekennzeichnet sind, kann nur innerhalb der Klasse zugegriffen werden. In der regel sollten Attribute immer private deklariert werden, damit der Zugriff nur über möglich gemacht wird. Dies wird auch Datenkapselung genannt.

Mit einer final gekennzeichneten Klasse, lässt sich keine Subklasse ableiten. Wenn ein Attribut mit final gekennzeichnet wird, kann der wert nur bei der Initialisierung zugewiesen werden und kann später nicht mehr geändert werden.

(vgl. Abst 2015, S. 57-58)

“Methoden, die mit native gekennzeichnet sind, werden wie abstrakte Methoden ohne Anweisungsblock definiert. Die Implementierung erfolgt extern in einer anderen Programmiersprache (Java Native Interface).” (Abst 2015, S. 58)

# Fazit

Aufgrund der Modularisierung erleichtert die objektorientierte Programmierung die Wiederverwendung von Codeblöcken. Das hat den großen Vorteil, dass einerseits Zeitaufwand gespart wird, da der Code nicht mehrmals an mehreren Stellen neu hingeschrieben werden muss und somit Fehler beim Kopieren auftreten. Selbst die Wartung wird leichter. Zum Beispiel eine Berechnung ändert sich, dann muss an einer zentralen Stelle der Code verändert werden und nicht an 10 verschiedenen Stellen. Dies war auch der Slogan der Entwickler von Django „Down’t Repeat Yourself“. Dies lässt sich gut mit der Vererbung realisieren. Dort wird eine Klasse erstellt für eine Berechnung oder Eigenschaften für ein Objekt und die Erbende Klasse kann dies übernehmen. (vlg. Augsten 2018)

# Literaturverzeichnis

Abst, D. (2015). *Grundkurs Java.* Wiesbaden: Springer Vieweg.

Augsten, S. (16. Januar 2018). Was ist OOP?. Abgerufen am 13. März 2021 unter https://www.dev-insider.de/was-ist-oop-a-677737/

Objektorientierte Programmierung. (25. Februar 2021). *Wikipedia*. Abgerufen am 13. März 2021 unter https://de.wikipedia.org/wiki/Objektorientierte\_Programmierung

OOP und UML. (o.D.). *Historische Entwicklung der OOP*. Abgerufen am 13. März 2021 unter http://www.oop-uml.de/oop-entwicklung.php