



ITWissen

Das große Online-Lexikon
für Informationstechnologie

Glossar

Objektorientierung

- **Abstrakter Datentyp**
- **Abstraktion**
- **Attribut**
- **Automatisches Objekt**
- **Datenkapselung**
- **Delegation**
- **Dependency Injection**
- **Destruktor**
- **Erweiterung**
- **Getypte objektorientierte Sprache**
- **Gültigkeitsbereich**
- **Hierarchie**
- **Imperative Programmierung**
- **Instanz**
- **Instanziierung**
- **Instanzmethode**
- **Instanzobjekt**
- **Instanzvariable**
- **IoC, inversion of control**
- **Klassendefinition**
- **Klassenmethode**
- **Klassenobjekt**
- **Klassenvariable**
- **Komplexe Klasse**
- **Komposition**
- **Konstruktor**
- **Lebensdauer**
- **Mehrfachvererbung**
- **Merkmal**
- **Metaklasse**

- **Methode**
- **Oberklasse**
- **Objekt**
- **Objektbasierte Programmierung**
- **Objekthierarchie**
- **Objektmodell**
- **Objektorientierte Programmierung**
- **Objektorientierte Software-Metrik**
- **Objektorientierung**
- **Objektreferenz**
- **Objekttyp**
- **Persistenz**
- **Polymorphie**
- **Prototyp**
- **Prozedurale Programmierung**
- **Spezialisierung**
- **Subobjekt**
- **Subtyp**
- **Typhierarchie**
- **Unterklasse**
- **Vererbung**
- **Impressum**

Abstrakter Datentyp

ADT, abstract data type

Ein Abstrakter Datentyp ist die Beschreibung eines Datentyps durch eine Zusammenfassung von Wertebereichen und anwendbaren Operationen. Dabei werden nur syntaktische und semantische Eigenschaften, die sogenannte Signatur und Algebra eines Abstrakten Datentyps, definiert. Von der internen Datenstruktur und der Implementierung der Operationen wird abstrahiert, so dass zu einem Abstrakten Datentyp mehrere Implementierungen möglich sind, die aber alle der Signatur und Algebra genügen müssen. In objektorientierten Sprachen können Abstrakte Datentypen durch Klassen beschrieben werden, meistens jedoch nur unter Berücksichtigung syntaktischer Eigenschaften, wie z.B. Eindeutigkeit von Bezeichnern und Definition der Typen von Attributen und Argumenten der *Methoden*.

Abstraktion

abstraction

Abstraktion ist ein in der *objektorientierten Programmierung* (OOP) verwendeter Begriff. Unter Abstraktion versteht man die Beschränkung auf die im Sinne einer Anwendung wesentlichen Eigenschaften von Objekten. Bei der Abstraktion werden nur die Eigenschaften eines Objektes betrachtet, die es aus konzeptioneller Sicht von anderen Objekten unterscheidet. Dabei wird von der internen Objektstruktur und der Implementierung der *Methoden* abstrahiert.

Attribut

attribute

Allgemein handelt es sich bei Attributen um Beschreibungsmerkmale, die einer Datei, einem Verzeichnis oder einem Datenfeld in einer Datenbank zugewiesen werden können. So können Attribute Parameter bezeichnen, die zur korrekten Verwaltung von Ressourcen erforderlich sind. Den Begriff Attribut gibt es in unterschiedlichem Kontext; in der Dateiverwaltung, der Internet-Terminologie, der Verschlüsselung, bei Grafiken, X.500 und bei Programmiersprachen. Die Bezeichnung Attribut wird auch in der *objektorientierten Programmierung* (OOP) verwendet. Darunter versteht man eine Variable zur Speicherung von objektspezifischen Daten, die genau einem *Objekt* zugeordnet ist. Die Menge der Attribute eines Objektes

Objektorientierung

definiert die Objektstruktur; die Menge der aktuellen Attributwerte den Objektzustand. Attribute von *Instanzenobjekten* heißen auch *Instanzvariablen*, Attribute von *Klassenobjekten* dementsprechend *Klassenvariablen*.

Automatisches Objekt

Die Bezeichnung automatische *Objekte* steht im Zusammenhang mit der *objektorientierten Programmierung*. Der *Gültigkeitsbereich* von automatischen Objekten erstreckt sich von der Deklaration bis zum Ende des umgebenden Blocks, z.B. Anweisungsblock, Funktion, Programm. Bei geschachtelten Deklarationen des gleichen Identifikators gilt die Bindung eines Identifikators stets zur nächstumfassenden Deklaration. Ihre *Lebensdauer* geht von der Abarbeitung der Definition bis zum Verlassen des Gültigkeitsbereichs. Der *Konstruktor* wird automatisch aufgerufen. Dieses Verhalten ist das in allen blockstrukturierten Programmiersprachen übliche Verhalten. Da hier Deklaration stets mit der Objekterzeugung verbunden ist, besteht bei dieser Form keine Gefahr Objekt-Identifikatoren ohne assoziiertes Objekt zu verwenden oder erzeugte Objekte nicht mehr identifizieren zu können. Eine weitere Form sind die anwendungskontrollierten Objekte. Neben diesen beiden Formen gibt es noch Mischformen in objektorientierten Sprachen, die aus der Programmiersprache C entwickelt wurden.

Datenkapselung

Mit Datenkapselung bezeichnet man die Kapselung einer Datenstruktur und der Implementierung der auf diese anwendbaren Operationen. Eine gekapselte Datenstruktur besitzt eine Schnittstelle mit den von außen zugreifbaren Operationen. Alle Zugriffe auf die gespeicherten Daten erfolgen ausschließlich über die Operationen dieser Schnittstelle. Gekapselte Datenstrukturen werden oft als *Instanzen Abstrakter Datentypen* erzeugt. In der *objektorientierten Programmierung* bilden die *Objekte* die Einheiten der Datenkapselung.

Delegation *delegation*

Der Begriff Delegation wird in der *objektorientierten Programmierung* (OOP) verwendet. Unter Delegation versteht man die Verarbeitung von Nachrichten durch Weiterleitung an andere *Objekte*. Die Delegation von Nachrichten wird z.B. in komplexen Objekten verwendet, in dem empfangene Nachrichten an *Subobjekte* - das sind Teile eines komplexen Objekts - weitergeleitet werden.

Dependency Injection *DI, dependency injection*

Dependency Injection (DI) ist ein Begriff, der im Zusammenhang mit der *objektorientierten Programmierung* steht. Damit wird ausgedrückt, dass Module (*Objekte, Klassen*) ihre Abhängigkeiten - Informationen oder ein bestimmtes Verhalten durch den Aufruf von *Methoden* - von einer anderen, externen *Instanz* zugewiesen bekommen, was dann auch als Injektion bezeichnet wird. Diese Injektion wird von einem DI-Framework - wie etwa dem Spring oder Unity-Framework - durchgeführt. Eng verknüpft mit dem Begriff der Dependency Injection ist das Prinzip *Inversion of Control* - die Umkehrung des Kontrollflusses. Es lassen sich mit Setter Injection, Constructor Injection und Interface Injection drei verschiedene Arten von Dependency Injection unterscheiden. Damit werden auch die Ebenen deutlich, auf denen die Injektion erfolgt. Das Ziel der Verwendung von DI ist es, Abhängigkeiten zwischen Modulen zu minimieren. Dependency Injection ist auch ein Konzept, was als Bestandteil einer Software-Architektur verwendet werden kann. Der Begriff der Dependency Injection wurde durch Martin Fowler geprägt und eingeführt. Unter dem u.g. Link ist eine ausführliche Beschreibung von Martin Fowler zu diesem Thema verfügbar.

Bereits seit Beginn der *Objektorientierung* ist das Prinzip des Inversion of Control (IoC) ein in diesem Zusammenhang diskutierter Aspekt. Aber erst im Verlauf der weiteren Entwicklung von objektorientierten Programmiersprachen und Konzepten wie beispielsweise im Umfeld von C++, Java, Python u.a. wurde dieses Thema wieder populärer; obwohl das Prinzip keineswegs

Objektorientierung

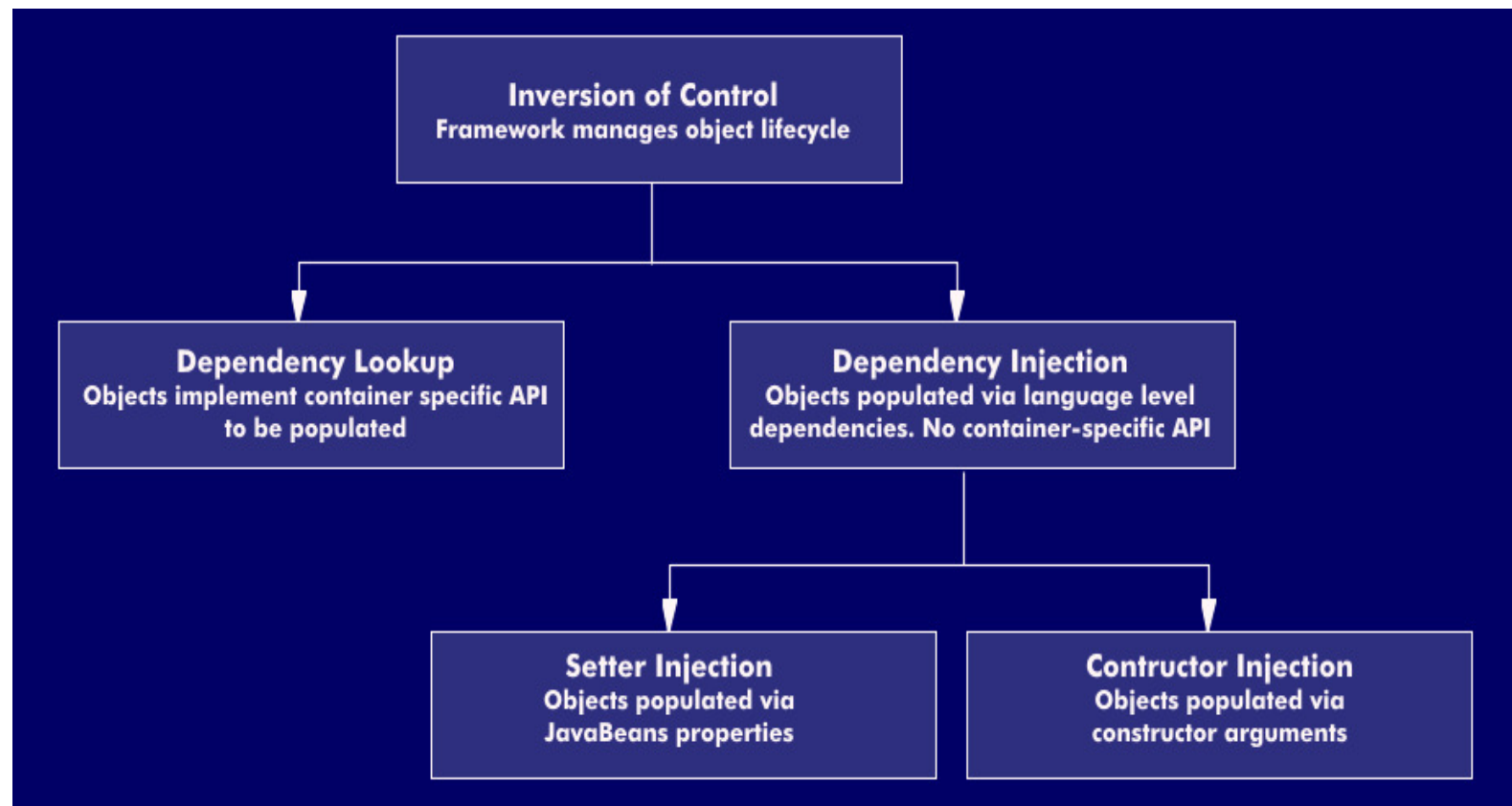
eine neue Erfindung ist. Nunmehr ist Inversion of Control als ein allgemeines Merkmal von Frameworks zu verstehen, was man im allgemeinem Sprachgebrauch auch als das sogenannte Hollywood-Prinzip „Don't call us, we'll call you!“ bezeichnet. Damit liegt die Verantwortlichkeit bei der Programmausführung auf Seiten des Frameworks - was auch als umgekehrter Kontrollfluss bezeichnet wird. Damit liegt die Verantwortlichkeit ausdrücklich nicht bei den Komponenten, die auf Grundlage dieses Frameworks entwickelt wurden. Dazu müssen die entsprechenden Module sogenannte Callback-Methoden (Callbacks) implementieren. Dabei wird die Laufzeit-Umgebung für das Framework durch einen Container zur Verfügung gestellt. Die Callbacks geben dem Framework während der Laufzeit die Möglichkeit der Übergabe - der Injektion - von Informationen an diese Module oder auch durch Injektion ein bestimmtes Verhalten von Module durch den Aufruf ihrer Methoden zu realisieren. So kann auch der *Lebenszyklus* eines kompletten Moduls von außen durch die Callbacks beeinflusst werden.

Dependency Injection (DI) ist eine spezielle Form des Inversion of Control-Prinzips, das von Martin Fowler definiert und eingeführt wurde. Der Hintergrund war die vorwiegende Nutzung des Inversion of Control-Prinzips für die Injektion von Referenzen auf benötigte Ressourcen durch leichtgewichtige Frameworks wie Spring und Container wie PriceContainer oder Excalibur. Durch DI können Dienste sogenannter Service-Fabriken nach außen hin verborgen werden. Als Fabriken werden in der Objektorientierung Objekte bezeichnet, die wiederum andere Objekte erzeugen können. So sollen durch die Nutzung von DI keine Abhängigkeiten zwischen Modulen und Klassen entstehen. Die Verantwortlichkeit für das Zusammenwirken von Modulen liegt in jedem Fall beim Framework, das dazu weitergehende Informationen bezüglich der Konfigurationen dieser Module benötigt. Diese werden entweder direkt in den Code integriert oder aber stehen separat in einer Konfigurationsdatei - häufig eine XML-Datei - zur Verfügung.

Objektorientierung

Die Übergabe von genutzten Objekten an zu nutzende Objekte wird also immer auf Basis dieser Konfigurationsinformationen erfolgen. Die Wahl der Konfiguration ist abhängig vom jeweils vorliegenden Einzelfall.

Es gibt drei grundlegend verschiedene Formen von Dependency Injection bei der Übergabe von



Verschiedene Arten von IoC

Modulen:

Setter Injection. Das DI-Framework sorgt über die Setter-Methode dafür, dass ein benötigtes Modul injiziert wird.

Constructor Injection. Beim Constructor Injection werden die benötigten Module als Argument des *Konstruktors* einer Klasse übergeben.

Interface Injection. Hier erfolgt die Injektion über explizit implementierte Schnittstellen (interfaces) sowohl auf Seiten des genutzten Moduls als auch vom nutzenden Modul. Diese Möglichkeit der Injektion wird in der Praxis jedoch vermieden, da sich durch die Implementierung der Interfaces wiederum Abhängigkeiten ergeben.

Vorteilhaft anzuwenden ist DI bei der Zusammenführung einer Anwendung aus verschiedenen Modulen. Dabei können durch das Verfahren verschiedene Arten der Konfiguration unterstützt werden. Durch die Zuordnung von Modulen zu einer Konfiguration können durch das Framework beliebige Formen der Konfiguration beispielsweise spezielle Server-Konfigurationen oder Test-Konfigurationen aufgebaut werden. Dabei liegt der Schwerpunkt von Dependency Injection auf der Entkopplung von Modulen, die ansonsten keinerlei Berührung haben. Dependency Injection ist nicht als Ersatz für Fabriken anzusehen. Bei der Konfiguration der Module über separate Dateien gibt es aber auch einen großen Nachteil für den Entwickler - die Abläufe einer Applikation sind ohne genaue Kenntnis der Konfigurationsdatei nicht mehr nachvollziehbar. Für unterschiedliche Programmiersprachen sowie Plattformen gibt es die verschiedensten Frameworks, die Dependency Injection unterstützen:

- Für Java: Spring, EJB 3.0, Seam, Guice.
- Für .NET: Unity, Ninject, ObjectBuilder, LightCore, PicoContainer, Structuremap.
- Für Python: PyContainer.
- Für Ruby: Needle, Copland.

Objektorientierung

- Für C++: DSM.
- Für PHP5: Garden, Symfony Components, Stubbles IoC.

Eine alternative Möglichkeit zur Dependency Injection sind sogenannte Service Locator - damit bezeichnet man ein übergeordnetes Verzeichnis von Diensten. Der Service Locator kennt für alle Dienste, die eine Applikation benötigt, eine entsprechende Realisierung und stellt sie dieser auf Anfrage zur Verfügung. Da hier jedoch nichts geschieht ohne dass das nutzende Modul explizit anfragt, kann nicht von einer Umkehrung des Kontrollflusses - der Voraussetzung für Dependency Injection - gesprochen werden.

<http://www.martinfowler.com/articles/injection.html>

Destruktor

destructor

Der Begriff Destruktor wird in der *objektorientierten Programmierung* (OOP) verwendet.

Darunter versteht man eine *Methode*, die zuerst die *Attribute* eines Objektes und dann das *Objekt* selbst löscht.

Der zum Destruktor gegensätzliche Mechanismus ist der *Konstruktor*.

Erweiterung

expansion

Der Begriff Erweiterung wird in der *objektorientierten Programmierung* (OOP) verwendet. Unter Erweiterung versteht man die Definition zusätzlicher *Merkmale* in einer *Unterklasse*. Bei der Erweiterung wird die Menge der Merkmale einer Unterklasse, die aufgrund der *Vererbung* alle Merkmale der Oberklasse(n) enthält, um zusätzliche Merkmale ergänzt. Durch die Erweiterung wird eine Klasse zur *Spezialisierung* ihrer Oberklassen.

Getypte objektorientierte Sprache

Mit getypter objektorientierter Sprache wird eine objektorientierte Programmiersprache bezeichnet, die auf einer Typisierung der *Objekte* basiert. In statisch getypten objektorientierten Sprachen kann die Typkonsistenz von Ausdrücken und Zuweisungen bereits

Objektorientierung

bei der Übersetzung überprüft werden, während in dynamisch getypten objektorientierten Sprachen diese Überprüfung teilweise erst zur Laufzeit möglich ist. Getypte objektorientierte Sprachen bieten eine eingeschränkte *Polymorphie*.

Gültigkeitsbereich

Die beiden Begriffe *Lebensdauer* und Gültigkeitsbereich müssen in Objektorientierten Programmier-sprachen explizit voneinander unterschieden werden.

Zum Umgang mit einem *Objekt* gehört neben der Erzeugung auch die Deklaration eines Identifikators, der mit dem Objekt assoziiert wird; die Objektdefinition setzt sich also aus der Deklaration eines Identifikators, ggf. mit Typinformationen, und der Erzeugung des Objekts zusammen. In objektorientierten Programmiersprachen werden Objekte beispielsweise durch Variable oder formale Parameter identifiziert, die das erzeugte Objekt aufnehmen. Im Gültigkeitsbereich ihres Identifikators ist das Objekt durch diesen Identifikator erreichbar. Der Gültigkeitsbereich ist also eine statische, d.h. eine vom Programmtext abhängige, Eigenschaft und bezeichnet den Teil des Programms, in dem genau dieses Objekt unter seinem Identifikator und gleicher Bedeutung verwendet werden kann.

Die blockstrukturierten objektorientierten Programmiersprachen unterscheiden verschiedene Arten der Objektdefinition, wie die der automatischen Objekte und der anwendungskontrollierten Objekte, die sich auf den Gültigkeitsbereich und die Lebensdauer der Objekte auswirken.

Hierarchie *hierarchy*

Die Bezeichnung Hierarchie wird in der Netzwerktechnik für die die Gliederung von Strukturen verwendet. Dieser Ordnungsbegriff wird auch in der *objektorientierten Programmierung* verwendet.

In der objektorientierten Programmierung versteht man unter Hierarchie die partielle Ordnung

Objektorientierung

von Objekten oder Klassen entsprechend einer zwischen ihnen definierten Beziehung. In den meisten objektorientierten Sprachen werden verschiedene, orthogonale Hierarchien verwendet: die *Objekthierarchie*, der die *Komposition* „part-of“-Beziehung zugrunde liegt und die *Klassenhierarchie*, die auf der *Vererbung* der Implementierung basiert. In einigen Sprachen wird davon die *Typhierarchie* unterschieden.

Imperative Programmierung

imperative programming

Bei der imperativen Programmierung werden Programme als aufeinander folgende Befehle formuliert. Die Befehle verändern während der Programmausführung in Variablen gespeicherte Werte und können ermitteln so Berechnungsergebnisse.

Imperative Programmierung ist ein Programmierparadigma und wird beispielsweise in der *prozeduralen Programmierung* und in der *objektorientierten Programmierung* umgesetzt. Ein anderes Paradigma ist die deklarative Programmierung.

Instanz entity

Eine Entität bezeichnet in der Informatik das abstrakte, individuelle *Objekt*. Dabei können die Objekte materiellen oder auch immateriellen Ursprungs sein. Vergleichbar ist Entität mit dem Begriff des Objektes oder der Instanz in der *objektorientierten Programmierung*.

Häufig wird Entität im Zusammenhang mit der Modellierung der datenorientierten Sichtweise mit Hilfe der ER-Modellierung (Entity-Relationship-Model) angewendet. Entitäten können in Beziehung zu sich selbst und auch zu anderen Entitäten bestehen. Entitäten wird ein spezifischer Entitätstyp zugewiesen. Die Eigenschaften von Entitäten werden durch *Attribute* gekennzeichnet. Dadurch können Entitäten gleichen Entitätstyps voneinander unterschieden werden. Entitätsmengen kennzeichnen die Zusammenfassung individueller Entitäten eines Entitätstyps. Entitätstypen können wiederum Entitätsklassen zugeordnet werden. Dies entspricht der Sichtweise in der objektorientierten Programmierung.

Instanziierung

Unter Instanziierung, die in Zusammenhang mit der *objektorientierten Programmierung* steht, versteht man speziell das Erzeugen eines *Objekts* als *Instanz* einer Klasse. Die Instanziierung erfordert evtl. die Angabe von objektspezifischen Parametern. Sie umfasst das Anlegen eines Speicherbereichs für die Instanz, sowie die Belegung von Attributen mit Default- oder Initialwerten.

Instanzmethode

Unter der Bezeichnung Instanzmethode, die in der *objektorientierten Programmierung* benutzt wird, versteht man speziell eine *Methode*, die von einer *Instanz* einer Klasse ausgeführt werden kann.

Eine Instanzmethode muss in der entsprechenden Klasse definiert oder von einer Oberklasse geerbt werden. Eine Instanz kann genau die Methoden ausführen, die Instanzmethoden ihrer Klasse sind.

Instanzobjekt

Unter einem Instanzobjekt versteht man speziell die *Instanz* einer Klasse, die im Gegensatz zu *Klassenobjekten* selbst keine Instanzen erzeugen kann. Die Bezeichnung Instanzobjekt kommt in der *objektorientierten Programmierung* vor.

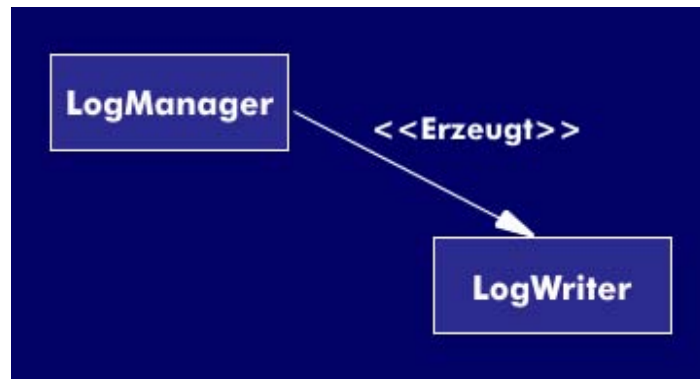
Instanzvariable

Instanzvariable ist ein in der *objektorientierten Programmierung* (OOP) verwendeter Begriff. Darunter versteht man eine *Variable*, die einer *Instanz* einer Klasse zugeordnet ist. Eine Instanzvariable ist also ein *Attribut* einer Klasse, das bei jeder *Instanziierung* dieser Klasse des neuen Objektes angelegt wird. Instanzvariable besitzen die gleiche *Lebensdauer* wie das *Objekt*, dem sie zugeordnet sind. Die aktuellen Werte der Instanzvariablen eines Objekts beschreiben den Objektzustand.

IoC, inversion of control Umkehrung des Kontrollflusses

Inversion of Control (IoC) ist ein Begriff, der im Zusammenhang mit der *objektorientierten Programmierung* ein Paradigma bezeichnet, bei dem ein mehrfach verwendbares Modul ein spezifisches Modul aufruft. Mit Modul werden abgegrenzte und eigenständige Teile einer Software bezeichnet - dabei kann es sich um *Objekte* oder Klassen handeln. Die Umkehr des Kontrollflusses ist nicht zu verwechseln mit dem in der objektorientierten Programmierung gleichermaßen bekannten Prinzip von der Umkehr der Abhängigkeiten (Dependency Inversion Principle). Vielmehr kann die Umkehr des Kontrollflusses eine Folge von der Umkehr der Abhängigkeiten sein. Inversion of Control wird im allgemeinen Sprachgebrauch auch als die Anwendung des Hollywood-Prinzips dargestellt: „Don´t call us, we´ll call you“. Für die Implementierung von IoC gibt es zwei Möglichkeiten - Dependency Lookup und *Dependency Injection*. IoC ist ein möglicher Weg, um Kopplungen zu verringern.

Sobald die Behandlung von Ereignissen von einem mehrfach verwendbaren Modul zur Verfügung gestellt werden soll, wird IoC verwendet. So rufen also nicht die spezifischen



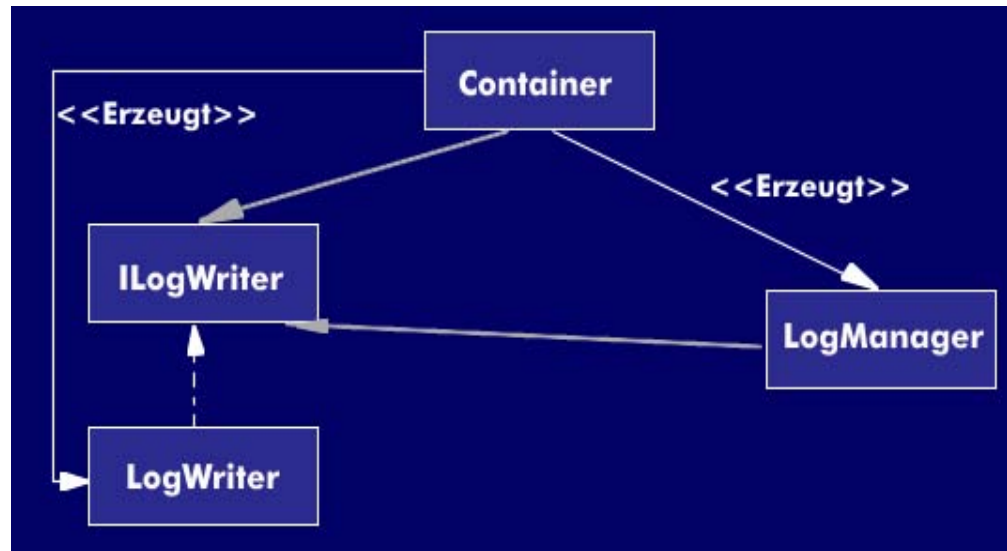
Die konkrete Implementierung eines Objektes erzeugt Abhängigkeiten

Module bei bestimmten Ereignissen die mehrfach verwendbaren Module auf, sondern umgekehrt ist das der Fall. Dabei kann das mehrfach verwendbare Modul auch als ein komplettes Framework ausgeführt sein, in dessen Rahmen dann die Klassen für spezifische und konkrete Anwendungen zusammengefasst sind. In IoC übernimmt dann das Framework die Rolle des Hauptprogramms und kümmert sich um den *Lebenszyklus* von Objekten. Dazu gibt das

Objektorientierung

Framework eine Schnittstelle vor, die zunächst von den spezifischen Modulen implementiert werden muss, so dass diese dann anschließend vom Framework aufgerufen werden kann - man bezeichnet dies auch als Callback-Mechanismus. Es gibt dabei zwei mögliche Formen der Implementierung von IoC:

- Dependency Lookup, dabei stellt ein Container Callbacks auf die Komponenten sowie einen Lookup-Kontext zur Verfügung. Verwendet wird dieser Ansatz beispielsweise bei Enterprise JavaBeans oder Apache Avalon. Dabei ist es Aufgabe der einzelnen Komponenten, den Lookup auf Ressourcen über entsprechende APIs des Containers zu realisieren.
- Dependency Injection ist eine spezielle Ausprägung des IoC.



Bei IoC erzeugt ein Container Instanzen und injiziert Abhängigkeiten

Nachfolgend wird erläutert, was IoC für die praktische Software-Entwicklung bedeuten kann.

Bei einem klassischen Ansatz ist es üblich, Klassen und deren Abhängigkeiten fest zu codieren. Dabei wird mit konkreten Klassen gearbeitet, die dann über *Konstruktoren* oder *Properties* entsprechende Abhängigkeiten erhalten.

Die Abbildung verdeutlicht diese Vorgehensweise - dabei ist LogManager direkt abhängig von der konkreten Implementierung von LogWriter. Damit wird bereits mit der Compilierung festgelegt, welche Interaktionen mit anderen Objekten ausgeführt werden. So sind Änderungen zur Laufzeit gar nicht möglich. Änderungen anderer Art können nur nach vorheriger Modifizierung des Quellcodes und wiederum anschließender Compilierung eingebracht werden.

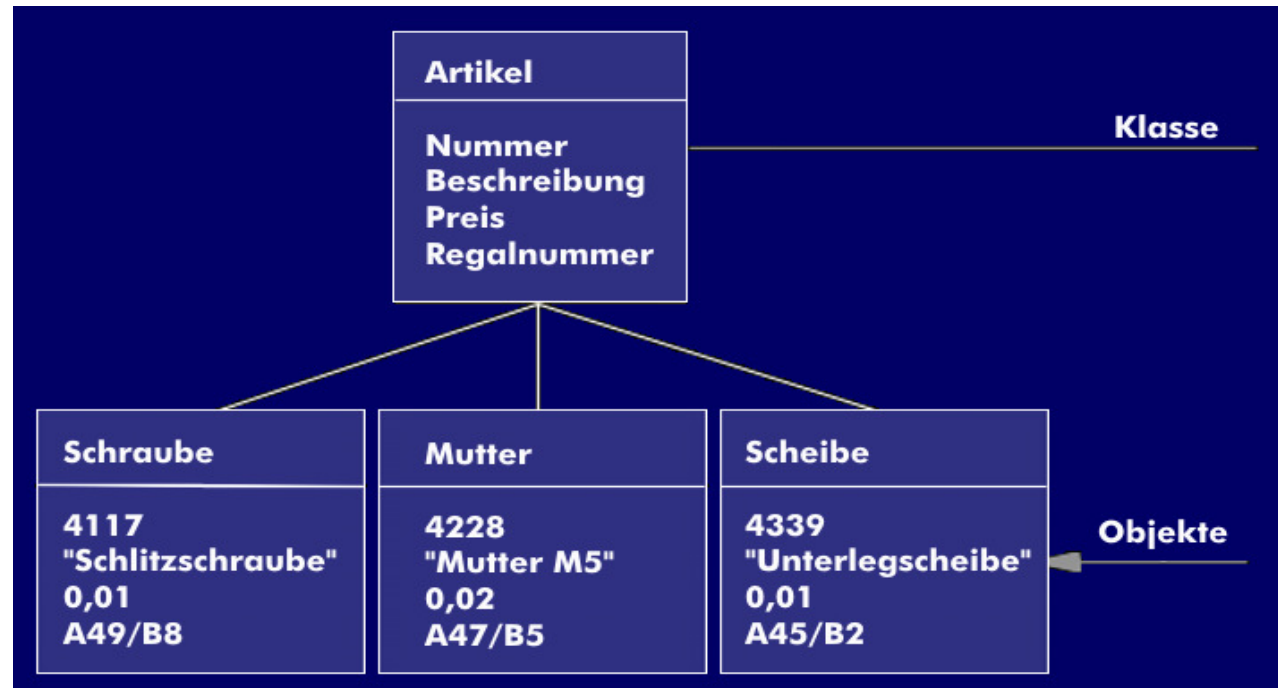
Vorteilhafter ist es, wenn diese Vorgehensweise umgekehrt wird. Dabei erzeugt eine Komponente - der aus der Abbildung ersichtliche Container - die Objekte und entscheidet dynamisch zur Laufzeit, wie und in welcher Form konkrete Implementierungen zu verwenden sind. Da damit der Container die Kontrolle übernimmt spricht, man auch von Inversion of Control. Die Verwendung dieses Prinzips bedingt *Abstraktionen* wie Interfaces und die Nutzung von Abstrakten Klassen, um die Möglichkeit der dynamischen Beeinflussung realisieren zu können.

Klassendefinition *class definition*

Eine Klassendefinition ist die Definition der *Merkmale* - das sind die charakteristischen Eigenschaften wie *Attribute* und *Methoden* - von Objekten. Eine Klassendefinition beschreibt die *Objekte* einer Klasse durch ein Schema, nach dem *Instanzen* dieser Klasse erzeugt und manipuliert werden. Dieses Schema besteht aus dem Klassennamen sowie den Attributen und Methoden der Klasse.

Falls *Vererbung* unterstützt wird, kann eine Klassendefinition zusätzlich die Namen der Oberklassen der neuen Klasse enthalten.

Außerdem können Klassendefinitionen die Zugriffsrechte auf Attribute und Methoden beschreiben und damit einer Teilmenge ihrer Merkmale (Attribute und Methoden) als Schnittstelle festlegen.



Klassen und Objekte der objektorientierten Programmierung (OOP)

Klassenmethode

Die Bezeichnung Klassenmethode steht in Zusammenhang mit der *objektorientierten Programmierung*. Unter einer Klassenmethode versteht man speziell eine *Methode*, die nur von einem *Klassenobjekt* und nicht von *Instanzobjekten* ausgeführt werden kann. Klassenmethoden werden nur von objektorientierten Programmiersprachen mit Klassenobjekten unterstützt. Sie können nur auf *Klassenvariablen* und andere Klassenmethoden, jedoch nicht auf *Instanzvariablen* und *Instanzmethoden* ihrer Klasse zugreifen.

Klassenobjekt

Der Ansatz der *Objektorientierten Programmierung* geht davon aus, dass jede Aktivität auf die Aktion eines *Objekts* zurückgeht. Insbesondere ist die Erzeugung von *Instanzen* einer bestimmten Klasse eine solche Aktivität. Man kann diese Aktivität einem speziellen „Instanzierungsobjekt“ zuschreiben, das dann aber Konstruktionsinformationen aus der *Klassendefinition* verwendet. Sauber im Sinne der Informationskapselung ist die Sicht, dass es zu jeder Klasse ein spezielles „Instanzierungsobjekt“ gibt, das die Klasseninformation kennt. Als solches kann man die Klasse selbst auffassen, die damit zum Klassenobjekt wird. Ein solches Klassenobjekt bezeichnet man in diesem Kontext dann auch als Fabrikobjekt, das durch Operationen wie „create“ und evtl. auch „destroy“ Objekte erzeugt bzw. wieder löscht. Fabrik- bzw. Klassenobjekte können aber auch weitere *Merkmale* haben. Die *Attribute* der Klassenobjekte werden auch als Klassenattribute (oder *Klassenvariablen*) bezeichnet, die *Methoden* dementsprechend als *Klassenmethoden*. Die *Konstruktoren* in einer jeden Klasse können in diesem Sinne auch als Klassenmethoden der Klassenobjekte angesehen werden.

Klassenvariable

Die Klassenvariable steht im Zusammenhang mit der *objektorientierten Programmierung*. Darunter versteht man speziell eine Variable, die ein *Attribut* von einem *Klassenobjekt* darstellt. Die aktuellen Werte von Klassenvariablen beschreiben den Zustand eines Klassenobjekts. Sie sind allen *Instanzen* der entsprechenden Klasse zugänglich. Da für jede Klasse genau ein Klassenobjekt existiert, wird jede Klassenvariable nur einmal angelegt.

Komplexe Klasse

Die Bezeichnung komplexe Klassen wird in der *objektorientierten Programmierung* (OOP) verwendet. Darunter versteht man eine Klasse, für die ein oder mehrere *Attribute* mit *Objekttyp* definiert sind. Ein *Objekt* einer komplexen Klasse, ein sogenanntes komplexes Objekt, besteht demnach aus ein oder mehreren Objekten anderer Klassen, sogenannten

Objektorientierung

Subobjekten. Ein komplexes Objekt kann beliebig viele *Subobjekte* besitzen, da durch einen Objekttyp u.a. auch Listen, Mengen oder Arrays von Objekten eines bestimmten Typs realisiert werden können. Als ein komplexes Objekt bezeichnet man auch die *Instanz* einer komplexen Klasse.

Komposition *composition*

Der Begriff Komposition wird in der *objektorientierten Programmierung* (OOP) verwendet. Unter Komposition versteht man die Definition komplexer Klassen durch die Definition von ein oder mehreren Attributen mit *Objekttyp*. In objektorientierten Sprachen werden komplexe *Objekte* als *Instanzen* einer komplexen Klasse erzeugt. Die Teilobjekte eines komplexen Objektes werden als Komponenten oder *Subobjekte* bezeichnet.

Konstruktor *constructor*

Die Bezeichnung Konstruktor wird in der *objektorientierten Programmierung* (OOP) verwendet. Unter einem Konstruktor versteht man eine *Methode*, die ein *Objekt* erzeugt und/oder seine *Attribute* initialisiert. Ein Konstruktor, der Objekte erzeugt, muss als *Klassenmethode* definiert werden, während der Konstruktor, der nur die Attribute eines Objektes initialisiert auch als *Instanzmethode* definiert werden kann.

Der zum Konstruktor gegensätzliche Mechanismus ist der *Destruktor*.

Lebensdauer *lifetime*

Der Begriff Lebensdauer wird hier im Kontext mit objektorientierten Programmiersprachen behandelt, wobei die beiden Begriffe Lebensdauer und *Gültigkeitsbereich* explizit voneinander unterschieden werden müssen.

Die Lebensdauer eines *Objekts* ist eine dynamische d.h. eine vom Programmablauf abhängige Eigenschaft und bezeichnet den Zeitraum vom Erzeugen eines Objekts bis zu seinem Beenden. Zum Umgang mit einem Objekt gehört aber neben der Erzeugung auch die

Objektorientierung

Deklaration eines Identifikators, der mit dem Objekt assoziiert wird; die Objektdefinition setzt sich also aus der Deklaration eines Identifikators, ggf. mit Typinformationen, und der Erzeugung des Objekts zusammen. Objekte werden in objektorientierten Programmiersprachen z.B. durch Variable oder formale Parameter identifiziert, die das erzeugte Objekt aufnehmen. Im Gültigkeitsbereich ihres Identifikators ist das Objekt durch diesen Identifikator erreichbar.

Die blockstrukturierten objektorientierten Programmiersprachen unterscheiden verschiedene Arten der Objektdefinition, wie die der automatischen Objekte und der anwendungskontrollierten Objekte, die sich auf den Gültigkeitsbereich und die Lebensdauer der Objekte auswirken.

Mehrfachvererbung

Der Begriff Mehrfachvererbung wird in der *objektorientierten Programmierung* (OOP) verwendet. Unter Mehrfachvererbung versteht man eine *Vererbung*, bei der eine Klasse mehrere direkte Oberklassen besitzt. Eine Klasse, die durch Mehrfachvererbung definiert wird, erbt die *Merkmale* aller Oberklassen. Dabei können Konflikte entstehen, falls ein Merkmal in verschiedenen Oberklassen definiert ist oder eine indirekte Oberklasse als mehrfache Oberklasse auftritt.

Merkmal

Der Begriff Merkmal wird in der *objektorientierten Programmierung* (OOP) verwendet. Merkmal ist ein Oberbegriff für die charakteristischen Eigenschaften eines *Objekts*, d.h. dessen *Attribute* und *Methoden*.

Metaklasse *meta class*

Eine Metaklasse ist eine Klasse, deren *Instanzen* wiederum Klassen sind. In objektorientierten Programmiersprachen mit Metaklassen werden Klassen als *Objekte*,

sogenannte *Klassenobjekte*, betrachtet. Diese Klassenobjekte sind dann wiederum Instanzen einer Klasse, die als Metaklasse bezeichnet wird. Da Metaklassen ebenfalls als Klassenobjekte betrachtet werden können, lassen sich Metaklassen und Klassenobjekte nicht eindeutig voneinander unterscheiden. Durch das Konzept der Metaklassen kann u.a. eine dynamische Manipulation von *Klassendefinitionen* realisiert werden.

Methode *method*

In der *objektorientierten Programmierung* (OOP) definiert eine Methode eine Operation, die ein *Objekt* ausführen kann und die durch die entsprechende *Klassendefinition* ihren Namen und ihre Argumenttypen charakterisiert wird.

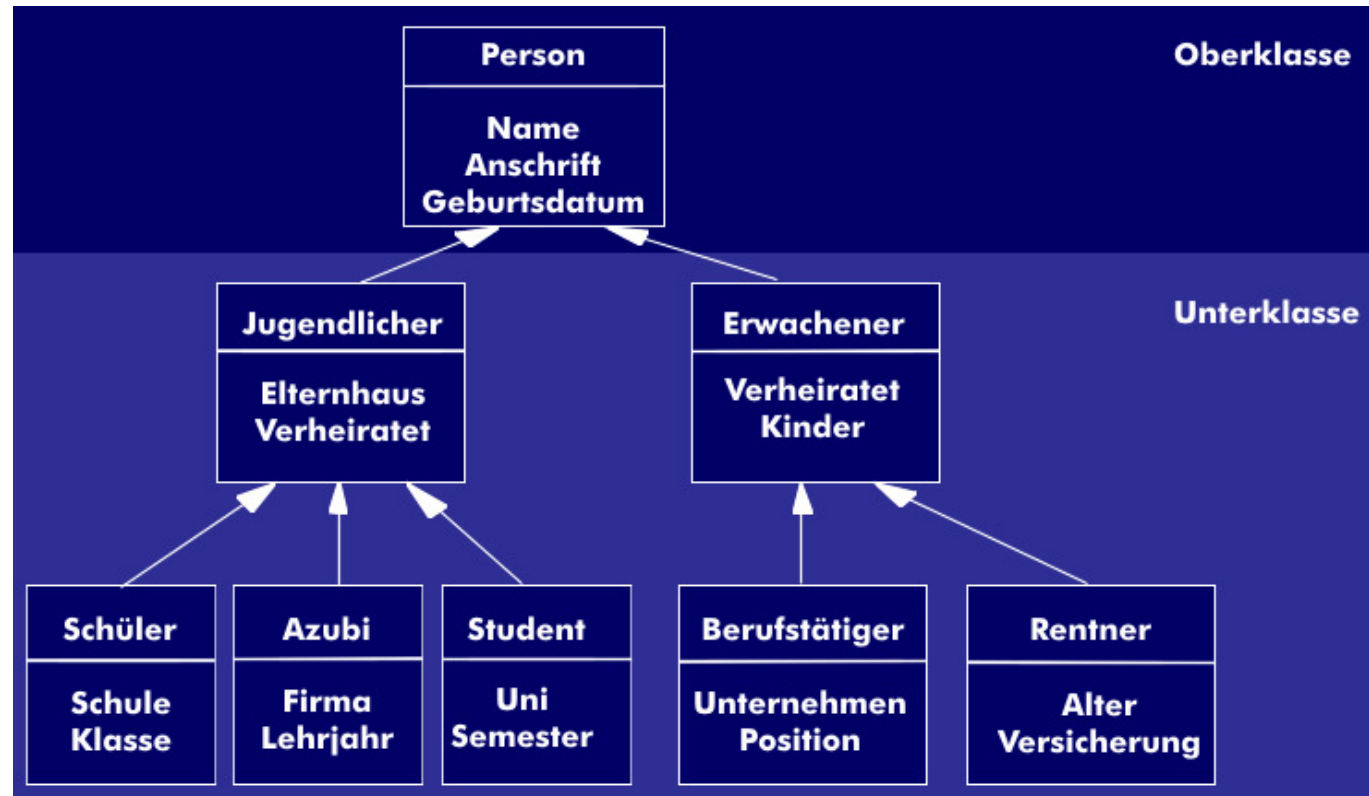
Die zulässigen Methoden eines Objekts werden in objektorientierten Sprachen durch die Klassendefinition festgelegt. Die Methoden realisieren die Dienste eines Objekts und legen das Objektverhalten fest.

Oberklasse

In der *objektorientierten Programmierung* (OOP) definiert eine Oberklasse eine Klasse, deren *Attribute* und *Methoden* durch *Vererbung* an abgeleitete Klassen, den *Unterklassen*, übertragen werden. In Abhängigkeit von der Anzahl der Vererbungsstufen, d.h. Anzahl der Klassen zwischen einer Unter- und einer Oberklasse, spricht man auch von den direkten oder indirekten Oberklassen einer Klasse. Darüber hinaus gibt es noch die mehrfache Oberklasse und die virtuelle Oberklasse.

Die mehrfache Oberklasse ist eine spezielle Klasse, die mehrfach als Oberklasse einer abgeleiteten Klasse auftritt. Eine abgeleitete Klasse besitzt eine mehrfache Oberklasse, wenn sie entweder direkt oder indirekt mehrfach von derselben Klasse abgeleitet wird. Im zweiten Fall besitzen mehrere Oberklassen der abgeleiteten Klasse eine gemeinsame Oberklasse. Durch mehrfache Oberklassen entsteht das Problem der wiederholten Vererbung.

Objektorientierung



Vererbung mit Unter- und Oberklassen

Unter einer virtuellen Oberklasse ist eine mehrfache Oberklasse zu verstehen, deren *Merkmale* jedoch nur einmal an jede abgeleitete Klasse, also die Unterklasse, vererbt werden. Virtuelle Oberklassen vermeiden das Problem der wiederholten Vererbung, indem das mehrfache Erben ihrer Merkmale durch eine indirekte Unterklasse unterdrückt wird.

Objekt

O, object

1. Ein Objekt ist ein abstraktes Element unserer Vorstellung und kann damit alles oder nichts sein. Je nach Anwendung wird das Wort Objekt unterschiedlich interpretiert. So ist in der Datenkommunikation ein Objekt ein ablauffähiges Datenpaket, das Daten, Codierungen, Funktionen und Ablaufmechanismen enthält.

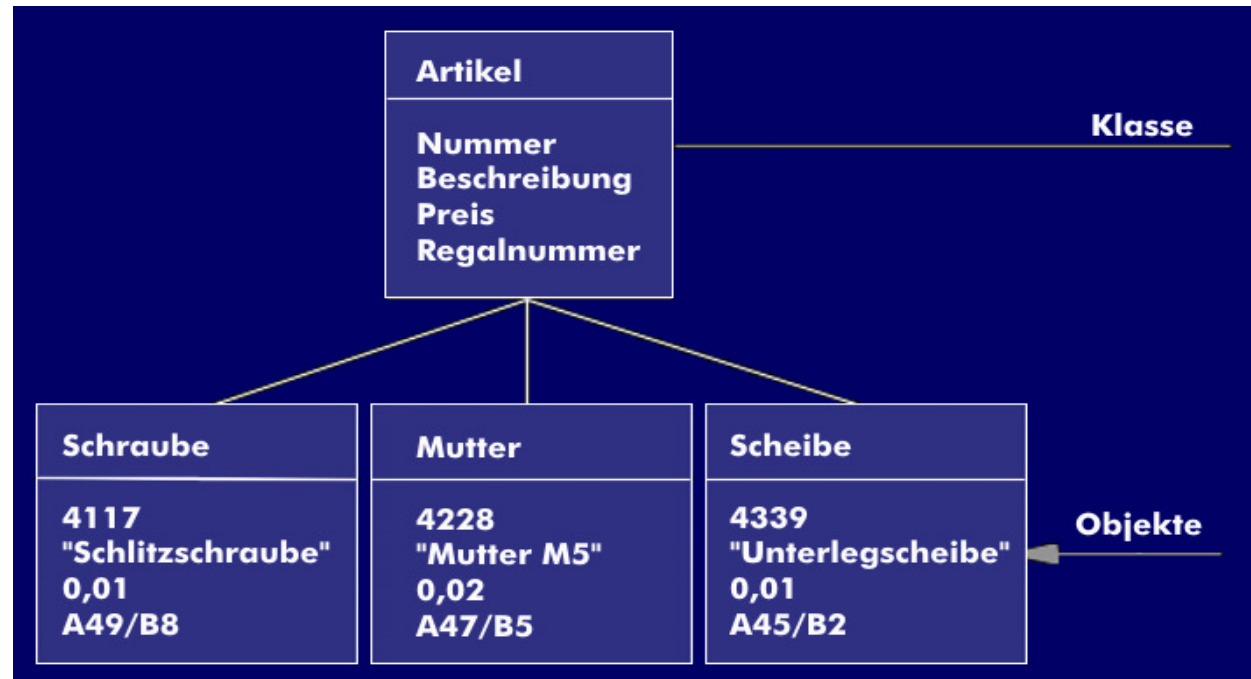
Das Objekt spiegelt eine Realität aus der Anwendungsumgebung mitsamt allen Zusammenhängen wider. Es ist die modellartige Nachbildung von Strukturen, die ein Softwareproblem beschreiben und die auf dem Weg zum Ergebnis zusammenwirken. Objekte können in Beziehung zueinander gestellt werden, wobei das Zusammenwirken zwischen den Objekten nach einem funktionalen Plan erfolgt.

In der Software versteht man unter einem Objekt eine *Instanz* einer Klasse, die wiederum durch die abstrakte Beschreibung der Objekte gebildet wird. Mit Objekten erstellte Systeme sind sehr sicher und haben eine hohe Qualität, weil auf die Daten nur ihr eigener Code zugreifen kann. Zur Durchsetzung dieser *Kapselung* wird der Code in einer begrenzten Anzahl von *Methoden* gruppiert. Unterschiedliche Objekte können unterschiedlich auf einen Befehl reagieren.

2. In der *objektorientierten Programmierung* (OOP) versteht man unter einem Objekt die Zusammenfassung einer Datenstruktur und der darauf anwendbaren Methoden zu einer Einheit. Ein Objekt besitzt eine Struktur - die Objektstruktur -, einen Zustand - den Objektzustand -, ein Verhalten - das Objektverhalten -, und eine Identität. In objektorientierten Sprachen ist jedes Objekt Instanz einer Klasse, durch deren Definition, die *Klassendefinition*, die *Merkmale* des Objekts festgelegt sind. Ein Objekt kann entweder *Instanzobjekt* oder *Klassenobjekt* sein. In der objektorientierten Programmierung bilden Objekte die Einheiten der Datenkapselung.

Unter Objektstruktur versteht man in der objektorientierten Programmierung die Datenstruktur

Objektorientierung



Klassen und Objekte der objektorientierten Programmierung (OOP)

eines Objekts, die durch die Menge der *Attribute* beschrieben wird.

Unter Objektzustand man speziell die aktuelle Wertbelegung der Attribute eines Objekts. Die zulässigen Zustände eines Objekts werden durch die Typen der Attribute und evtl.

zusätzlichen Restriktionen der Kombination der Werte verschiedener Attribute festgelegt.

Der Begriff Objektverhalten kennzeichnet das Verhalten eines Objekts, d.h. Aktionen und Reaktionen eines Objekts in Form von Änderungen des Objektzustands und dem Versenden von Nachrichten. Das Objektverhalten wird durch die Definition und Implementierung seiner Methoden festgelegt.

Objektbasierte Programmierung

*object based
programming*

Die Objektbasierte Programmierung ist ein Programmierparadigma, das *Objekte* und damit das Prinzip der *Datenkapselung* unterstützt. Im Gegensatz zur *objektorientierten Programmierung* unterstützt die objektbasierte Programmierung keine *Vererbung*.

Objektbasierte Programmierung unterstützt *Abstraktionen* (nur) dadurch, dass Objekte erzeugt werden, die durch Nachrichten miteinander kommunizieren und *Methoden* ausführen. Dadurch werden Manipulationen des Zustands der Objekte vorgenommen. In *Klassendefinitionen* werden genau die Konstruktionsbeschreibungen aller Objekte einer Klasse festgelegt, d.h. alle ihre *Attribute*, deren Wertebelegung ihren Zustand festlegen, sowie das Verhalten der Objekte der Klasse.

Objekthierarchie

Der Begriff Objekthierarchie wird in der *objektorientierten Programmierung* verwendet. Darunter versteht man die partielle Ordnung auf der Menge der *Objekte* entsprechend der zwischen ihnen bestehenden Kompositionsbeziehung („part-of“-Beziehung). Komponenten werden dabei als Nachfolger des Objekts, dem sie angehören, angeordnet. Objekthierarchien werden aufbauend auf komplexen Objekten gebildet und sind dynamisch, da sie zur Laufzeit durch Zuweisen, Erzeugen und Löschen von Objekten verändert werden können. Sie werden durch gerichtete Graphen dargestellt.

Objektmodell

object model

1. Objektmodelle beschreiben spezifische *Objekte*, die durch ihre Eigenschaften gekennzeichnet sind und auf die über ihre Schnittstellen zugegriffen werden kann. Häufig handelt es sich dabei um Software-Komponenten, die in Bibliotheken gesammelt und verwaltet werden und die jede für sich identifizierbar ist. Die Objektmodelle beschreiben die Eigenschaften für die *objektorientierte Programmierung* und schaffen die Basis damit die einzelnen Software-Komponenten miteinander kommunizieren und verknüpft werden können.

Objektorientierung

Das Component Object Model (COM) von Microsoft stellt so ein Objektmodell dar, weitere sind das System Object Model (SOM) von IBM, das Document Object Model (DOM), das Java Document Object Model (JDOM) sowie das Objektmodell für Netzanwendungen DCOM (Distributed) und CORBA.

2. In der objektorientierten Programmierung versteht man unter einem Objektmodell die Menge der objektorientierten Mechanismen, die einer objektorientierten Sprache oder einen objektorientierten Entwurf zugrunde liegen. Mögliche Kriterien zur Charakterisierung eines Objektmodells sind *Abstrakte Datentypen, Datenkapselung, Klassen, Klassenobjekte, Metaklassen, Vererbung, Komposition, Typisierung, Polymorphie, Persistenz* und Parallelität.

Objektorientierte Programmierung

OOP, object oriented programming

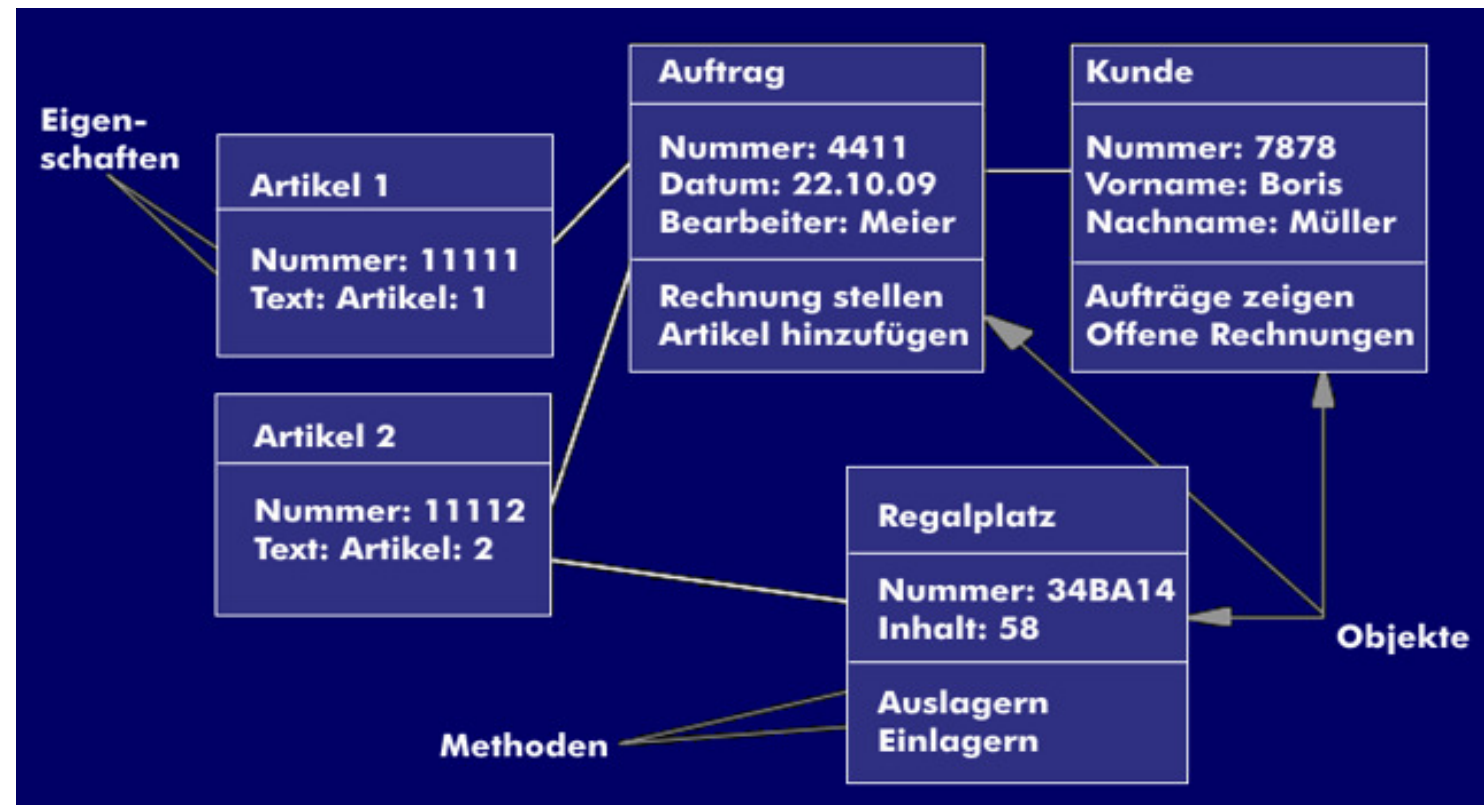
Die objektorientierte Programmierung (OOP) ist eine *Methode* zur Modularisierung von Programmen, die sich stark von der klassischen *prozeduralen Programmierung* unterscheidet. Objektorientierte Software ist, wenn sie gut entworfen wurde, leichter zu warten und zu erweitern als prozedurale. Zudem vereinfacht sie durch die strenge Modularisierung Unit-Tests und Wiederverwendung von Softwareteilen. Sie folgt dem Programmierparadigma der *imperativen Programmierung*.

Bei der objektorientierten Programmierung werden Programme in Einheiten unterteilt, die *Objekte* genannt werden. Jedes Objekt besitzt einen Zustand, der durch dessen Eigenschaften (Objektattribute) beschrieben wird. Nur die im Objekt selbst vorhandenen Funktionen (Methoden genannt), können dessen Daten manipulieren und so den Zustand verändern. Objekte können anderen Objekten Botschaften senden (indem sie deren Methoden aufrufen) und sie damit auffordern, ihren Zustand zu ändern. Letztendlich bleibt es aber dem Objekt selbst überlassen, ob es der Aufforderung nachkommt. Somit befindet sich das Objekt immer

Objektorientierung

in einem wohldefinierten, selbstkontrollierten Zustand.

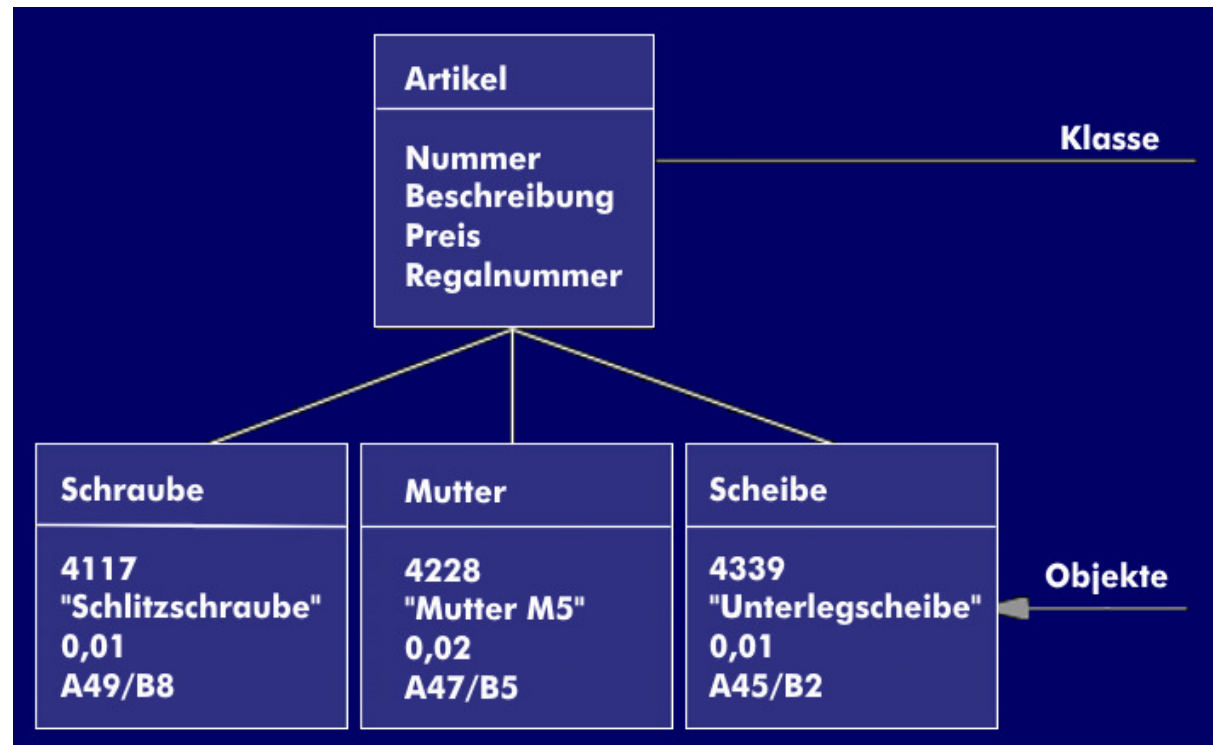
Man fasst in der OOP-Programmierung also Daten und Funktionen zu Objekten zusammen. Diese Objekte können auf vielfältige Weise miteinander in Verbindung stehen, indem sie gegenseitig ihre Methoden aufrufen oder ein Objekt andere Objekte enthält. So bilden die Objekte einer Software ein sehr flexibles Gesamtsystem.



Zusammenfassung von Daten und Funktionen in der objektorientierten Programmierung (OOP)

Objektorientierung

Jedes Objekt gehört zu einer Klasse. Klassen werden auch oft als Baupläne für Objekte beschrieben, weil sie definieren welche *Attribute* und Methoden die dazugehörigen Objekte besitzen. Jedes Objekt ist eine *Instanz* seiner Klasse, man sagt auch: Das Objekt „o“ instanziiert die Klasse „k“. Von einer Klasse kann es beliebig viele Objekte geben, deren Eigenschaften sich unterscheiden können aber nicht müssen. Ein Warenwirtschaftssystem beispielsweise kennt nur eine Klasse Artikel, aber viele Artikelobjekte.



Klassen und Objekte der objektorientierten Programmierung (OOP)

Objektorientierung

Objektorientierte Programme erfüllen verschiedene wichtige Kriterien, die die Entwicklung vereinfachen, beschleunigen und gleichzeitig die Qualität verbessern können:

- Datenkapselung: Die Daten eines Objekts können nicht unkontrolliert von außen verändert werden, letztendlich entscheidet immer das Objekt selbst über Änderungen seines Zustands.
- Austauschbarkeit: Objekte sind *Abstraktionen* realer Entitäten und Akteure, die ihre tatsächliche Implementierung vor der Außenwelt verbergen und dadurch austauschbar werden.
- Vererbung: Eigenschaften und Funktionen eines Objekts können an andere Objekte weitergegeben und von diesen übernommen, verändert oder überschrieben werden.
- Polymorphie unterstützt die Austauschbarkeit von Objekten, denn die gleiche Nachricht kann an unterschiedliche Objekte gesendet werden und dementsprechend unterschiedliche Aktionen bei diesen bewirken.

Nicht jede Programmiersprache unterstützt die objektorientierte Programmierung. C und Pascal sind beispielsweise rein prozedural. Andere Sprachen wie C++ oder Object Pascal erlauben OOP, erzwingen sie aber nicht. Die daraus resultierenden Programme können eine Mischung aus verschiedenen Paradigmen sein. Das gleiche gilt für Sprachen wie PHP oder Perl, die durch *Erweiterungen* OOP-fähig gemacht werden können. Weitere wie Eiffel, Smalltalk, Java oder C# sind rein objektorientiert.

Neben der eigentlichen objektorientierten Programmierung haben sich auch objektorientierte Verfahren zur objektorientierten Analyse (OOA) und zum objektorientierten Design (OOD) von Softwaresystemen etabliert. Dabei hat sich die Unified Modelling Language (UML) als allgemein anerkannte Notation für diese Verfahren durchgesetzt. Entsprechende Softwaretools, die teilweise in Entwicklungsumgebungen (IDEs) integriert werden, erlauben

die Generierung von Software aus grafisch entworfenen Systemen oder dokumentieren umgekehrt fertige Systeme in UML-Notation.

Ein in der Praxis immer wiederkehrendes Problem mit vielen Lösungsansätzen ist die Verbindung von relationalen Datenbanken (RDBMS) und objektorientierter Software. Das Object Relational Mapping (ORM) führt eine Umsetzung relationaler Strukturen in Objekte und umgekehrt durch. Alternativ können auch objektrelationale oder objektorientierte Datenbanken benutzt werden, deren Verbreitung allerdings nicht sehr groß ist.

Objektorientierte Software-Metrik

Zur Messung von Softwareprodukten werden sogenannte Software-Metriken verwendet, die unterschiedliche Eigenschaften von Softwareprodukten und -prozessen quantifizieren. Die in den 70er und 80er vorgestellten klassischen Software-Metriken nach Halstead, McCabe oder Rechenberg sowie die LOC-Metrik werden in die Gruppe der konventionellen Software-Metriken eingeordnet. Dazu gehören die Umfangsmetriken, welche unter verschiedenen Aspekten die „Größe“ einer Software ermitteln. Das können sowohl die Anzahl der codierten Zeilen wie bei Lines-of-Code (LOC) sein, die Verknüpfung der Anzahl von Operatoren und Operanden (Halstead) als auch der prozessorientierte Ansatz der Function-Point-Metrik.

Einen anderen Weg weist die McCabe-Metrik, die die logische Struktur u.a. die Schachtelungstiefe von Schleifen in die Messung einbezieht. Gemäß der Metrik nach Rechenberg wird der Aspekt von Messung der Software-Komplexität hervorgehoben, was auch allgemein als Datenstruktur-Metrik bezeichnet wird. Dabei werden die Anzahl der Variablen, deren Gültigkeit und *Lebensdauer* sowie deren Referenzierung in die Messung integriert. Eine weitere konventionelle *Methode* ist die sogenannte Stilmetrik, die sich darauf beschränkt, den Anteil der Kommentare im Source Code zu ermitteln.

Ein Ergebnis von Forschungen jüngerer Datums sind die sogenannten objektorientierten Software-Metriken. Diese berücksichtigen bei der Messung von Software die Zusammenfassung von Datenstrukturen und der darauf anwendbaren Methoden zu einem *Objekt*, dessen Beziehungen zu anderen Objekten sowie die generellen Strukturmerkmale *objektorientierter Programmierung* - wie da beispielsweise sind *Datenkapselung*, *Vererbung*, *Klassen*, *Methoden* und *Abstraktion*. Zur besseren Einordnung dieser Metriken lassen sich die folgenden Ebenen differenzieren:

Messung auf Methodenebene. Hier werden die Eigenschaften von Methoden gemessen. Dafür können noch konventionelle Methoden wie Lines-of-Code (LOC) oder McCabe eingesetzt werden.

Messung auf Klassenebene. Hier werden die Strukturmerkmale von Klassen gemessen. Als Maß dafür bieten sich die folgenden Metriken an: Response for a Class (RFC), Weighted Methods for Class (WMC) und Lack of Cohesion in Methods (LCOM).

Messung von Vererbungshierarchien. Hier werden die durch Vererbung und Abstraktion entstandenen Strukturen betrachtet. Als Maß hierfür bietet sich die Methode Number of Children (NOC) an.

Messung von Aggregationshierarchien. Hier wird ein Maß für die Verknüpfung von Klassen untereinander ermittelt. Dafür bietet sich das Verfahren Coupling between Objects (CBO) an. Im Allgemeinen werden hinsichtlich der genannten Metriken deren partiell fehlende theoretische Fundierung und die ausschließliche Messung trivialer Sachverhalte kritisiert.

Objektorientierung
OO, object oriented

In diesem Artikel wird die Objektorientierung aus verschiedenen Sichten definiert. Dies aus der Tatsache heraus, dass letztendlich über die Gewichtung der einzelnen Aspekte der konkrete Anwendungsfall entscheidet. Ein allgemein üblicher Nenner definiert

Objektorientierung

Objektorientierung mit den Punkten Objektbasierung, Klassenbasierung und dem Vererbungsmechanismus.

Objektorientierung beinhaltet:

1. Objektbasierung, d.h. *Kapselung* von Attributen,
2. Klassenbasierung durch Mengenabstraktion, d.h. die gemeinsame Modellierung gleichartiger *Objekte* in Form einer Klasse, und
3. Vererbung, also die Weitergabe von Attributen zwischen Klassen.

Objektorientierter Entwurf ist diejenige *Methode*, die zu Software-Architekturen führt, die auf den von jedem System oder Teilsystem bearbeitenden Objekten beruhen, und nicht auf der Funktion, die das System realisiert. Dabei wird mit einem objektorientierten Entwurf auch die Entwicklung von Software-Systemen definiert als strukturierte Sammlung von Implementierungen *abstrakter Datentypen*. Es lässt sich somit festhalten, dass objektorientierte Strukturierung sicher auf einer anderen Sichtweise beim Entwurf von Programmen beruht und durch eine Sammlung abstrakter Datentypen realisierbar ist. Die Struktur von Programmen basiert auf deren Daten.

Ein objektorientiertes Programm ist die Sammlung autonom agierender Agenten, genannt Objekte. Die Berechnung schreitet durch die Interaktion der Objekte fort. Dabei werden als Hauptkomponenten Objekte und deren Kommunikation untereinander identifiziert und die *Vererbung* spielt eine untergeordnete Rolle.

Objektorientierte Systeme sind objektbasiert, klassenbasiert, unterstützen Vererbung zwischen Klassen und Elternklassen und erlauben Objekten das Versenden von Botschaften untereinander. Hier liegt der Aspekt auf der Vererbung und der *Abstraktion* von Objekten durch Klassen.

Objektorientierung

Zusammenfassend unterstützt eine objektorientierte Programmiersprache:

- eine objektbasierte, modulare Struktur,
- Datenabstraktion,
- automatische Speicherplatzverwaltung,
- Klassen (als Typen),
- Vererbung,
- Polymorphie und dynamisches Binden sowie
- Mehrfaches und wiederholtes Erben.

Die Definition im konkreten Fall hängt sicher vom Gewicht ab, welches den einzelnen Konzepten gegeben wird.

Objektreferenz *object reference*

Die Objektreferenz ist ein Begriff aus der *objektorientierten Programmierung*. Darunter versteht man speziell einen Zeiger auf ein *Objekt*. In *getypten objektorientierten Sprachen* ist jede Objektreferenz auf Objekte von bestimmten Klassen eingeschränkt. Aufgrund von *Polymorphismus* kann dabei eine Objektreferenz auf *Instanzen* der ihr zugeordneten Klasse oder einer deren *Unterklassen* zeigen.

Objekttyp *object type*

Der Begriff Objekttyp wird in der *objektorientierten Programmierung* verwendet. Darunter versteht man einen Typ, der durch eine *Klassendefinition* festgelegt wird. In *getypten objektorientierten Sprachen* beschreibt eine Klassendefinition eine benutzerdefinierten Typen, der als Objekttyp bezeichnet wird und über den Namen der Klasse angesprochen werden kann. Die Interpretation dieses Namens ist dabei kontextabhängig, d.h. er kann abhängig von der Verwendung als Bezeichner für einen Typ oder eine Klasse stehen.

Persistenz *persistence*

Persistenz ist die Eigenschaft eines *Objekts*, durch die seine *Lebensdauer* permanent und damit unabhängig von der Ausführung des erzeugenden Programmes wird. Persistente Objekte besitzen eine potentiell unendliche Lebensdauer und können nur durch expliziten Aufruf einer speziellen *Methode*, z.B. des *Destruktors*, wieder gelöscht werden. Sie werden deshalb in permanenten Speichersystemen z.B. objektorientierten Datenbanksystemen verwaltet.

Klassen sind statische Beschreibungen einer Menge von Objekten. Die Sammlung von Klassen beschreibt ein objektorientiertes Programm. Das Programm selbst besteht aus den Objekten, die durch ihren Austausch von Botschaften untereinander das Programm zum Ablauf bringen. Aus verschiedenen Gründen ist es notwendig und wünschenswert, ein Programm an einer bestimmten Stelle zu unterbrechen und es an dieser Stelle wieder aufsetzen zu lassen. Dazu müssen die während des Programmlaufs berechneten Daten geeignet gespeichert werden. Dies betrifft bei der *objektorientierten Programmierung* hauptsächlich die Objekte mit all den zwischen ihnen bestehenden Verknüpfungen.

Persistenz ist nun die Eigenschaft von Objekten, über die Laufzeit des Programms hinaus in ihrer Identität, ihrem Zustand und ihrer Beschreibung zu existieren. Am einfachsten wäre es natürlich, wenn jedes Objekt weiß, wie es sich selbst abspeichert und wieder identifiziert. Tatsächlich wird dies dadurch erreicht, dass es in den meisten Programmiersprachen allgemein verfügbare Elternklassen gibt, die eben diese Funktionalitäten zur Verfügung stellen.

Zum Problem wird Persistenz von Objekten dann, wenn sich Klassenbeschreibungen ändern, die gleichen gespeicherten Daten aber weiter verwendet werden sollen. Mit diesem Problembereich beschäftigen sich vor allem objektorientierte Datenbanken.

Ein interessantes Problem ist in diesem Zusammenhang die häufig gestellte Forderung nach dem Einsatz relationaler Datenbankmanagementsysteme in Verbindung mit der objektorientierten Struktur eines Systems. Dies führt in der Regel zu einer weiteren Schicht in

der Architektur eines Systems, welche Objekte in Relationen umsetzt und umgekehrt.

Polymorphie *polymorphism*

Wörtlich bedeutet Polymorphie „viele Erscheinungsformen“. Der Begriff wird für verschiedene Erscheinungsformen objektorientierter Programmiersprachen verwendet.

1. Die Eigenschaft von Variablen, verschiedene Typen annehmen zu können.
2. Die Eigenschaft von Funktionen, polymorphe Argumente zuzulassen.
3. Die Eigenschaft einer Funktion, für verschiedene Parametersätze mit gleichem Namen definiert zu sein.
4. Die Eigenschaft eines Operators, sich auf *Instanzen* verschiedener Klassen während der Laufzeit zu beziehen. Daher werden polymorphe Botschaften von verschiedenen Objekten verschieden interpretiert.

Programmiersprachen werden entsprechend der Rolle, die die Typisierung dort spielt, in zwei Klassen unterschieden: statisch und dynamisch getypte Sprachen. Statisch bedeutet hier, dass bereits am Programmtext - also durch den Compiler - bestimmte Typinformationen ausgewertet werden können. Im Gegensatz dazu bedeutet dynamisch hier, dass erst zur Laufzeit des Programms das Laufzeitsystem diese Typinformationen auswertet.

In statisch getypten Programmiersprachen wie z.B. Pascal, Modula, der Programmiersprachen C, C++, wird ein Typ gerade den Variablen in einer expliziten Deklaration zugewiesen und legt fest, wie der Wert einer Variablen interpretiert und manipuliert werden kann. In dynamisch getypten Programmiersprachen wie z.B. Lisp und Smalltalk, ist dies gerade nicht der Fall. Dort werden die Typinformationen nicht mit der Variablendeklaration abgelegt, sondern der Wert einer Variablen enthält seine Typinformation. Eine Konsequenz ist, dass Variablen oder Funktionsargumente während der Laufzeit Werte verschiedener Typen aufnehmen können und dann entsprechend der Wertebelegung anders manipuliert werden.

Objektorientierung

Funktionen können nun so programmiert werden, dass sie unabhängig von der Typisierung ihrer Argumente unterschiedliche Aktionen durchführen können, so ist z.B. „+“ auf int-Argumenten anders definiert als auf real-Argumenten und liefert unterschiedliche Werte von Typen als Funktionswerte ab. Solche Variablen, Argumente und Funktionen nennt man polymorph (vielgestaltig).

Prototyp

Im Zusammenhang mit der *objektorientierten Programmierung* ist ein Prototyp ein *Objekt*, welches als Vorlage für die Generierung von weiteren Objekten dient. Die Vorlage kann wiederum eine Kopie eines Objektes oder auch eine entsprechende Referenz auf dieses sein. Grundsätzlich können alle bereits existierenden Objekte als Prototypen für neue Objekte gelten. Die geerbten Eigenschaften - das sind *Attribute* und *Methoden* - verhalten sich wie die ursprünglichen Eigenschaften des Prototyps. JavaScript ist ein Beispiel für eine Programmiersprache, die dieses auf der Basis von Objekten aufbauende Verfahren der *Vererbung* unterstützt. Ein Vorteil dieses Mechanismus ist es, dass Prototypen und davon abgeleitete Objekte zur Laufzeit des Programms modifiziert werden können. Man spricht hier auch von Prototyp-basierter Programmierung oder gleichfalls von der klassenlosen *Objektorientierung*.

Bei der Erzeugung von Objekten auf der Basis von Klassen wird deren Struktur durch die Struktur der Klasse bestimmt. Damit sind die Eigenschaften eines Objektes - d.h. seine Attribute und Methoden - bereits bei deren Erzeugung festgelegt, wobei diese dann nicht mehr modifizierbar sind. Die Klasse ist das zugrundeliegende „Baumuster“ des Objektes. Einen alternativen Weg geht die Prototyp-basierte Programmierung. Dabei wird ein neues Objekt auf der Grundlage eines Prototypens erstellt, und die konkrete Struktur des Objektes wird erst im Anschluss daran festgelegt. So können beispielsweise Attribute und/oder Methoden ergänzt

Objektorientierung

```
function Mitarbeiter (name) {  
    this.name = name;  
}  
  
Mitarbeiter.prototype.getName = function () {  
    return this.name;  
}
```

Beispiel für ein Prototypen-Konstrukt

Anpassung von Prototypen zur Laufzeit eines Programms hat aber den offensichtlichen Nachteil von möglichen Seiteneffekten oder den der schlechteren Wartbarkeit von Programmen. Die Flexibilität von Prototyp-basierten Programmiersprachen bei dieser Form der Modellierung bedingt auch, dass Optimierungen, wie das sie von Compilern klassenbasierter Strukturen während der Übersetzung durchgeführt werden, so nicht möglich sind. Dies kann auch negativ auf die Performance Prototyp-basierter Anwendungen einwirken.

Ein populäres Beispiel für eine Prototyp-basierte Programmiersprache ist JavaScript, die das Konzept der Klassen - und damit auch das der Klassen-basierten Vererbung - nicht unterstützt. Allerdings kann JavaScript aufgrund der Flexibilität der Sprache so verwendet werden, dass es wie bei einer Programmiersprache mit klassischen Vererbungsmechanismen aussieht. Dazu werden Klassen durch Funktionen deklariert und *Instanzen* mit dem auch in JavaScript bekannten new Operator erzeugt. Die so entstehenden Objekte verhalten sich dann ähnlich wie Objekte in beispielsweise in Java, C++ u.a.

In JavaScript deklariert man ein Objekt zunächst gänzlich ohne bestimmte Eigenschaften. Erst

werden, um so das Verhalten des neuen Objektes spezifisch beeinflussen zu können. Auch kann der Prototyp modifiziert werden, um so ein bestimmtes Verhalten eines Objektes zu generieren. Dabei sind in Folge natürlich alle auf Basis dieser geänderten Prototypen erzeugten Objekte von diesen Modifizierungen betroffen. Dieser Mechanismus der

wenn das neue Objekt bereits vorliegt, wird mit dem prototype Konstrukt ein Objekt zugeordnet, welches die Vorlage (das Prototyp-Objekt) darstellt. Dadurch werden alle Eigenschaften - Attribute sowie Methoden - komplett auf das neue Objekt abgebildet. So erhält das neue Objekt praktisch eine Referenz auf das Prototyp-Objekt. Auch hier gilt, dass alle Modifizierungen an der Vorlage sichtbar werden in den Objekten, die auf deren Basis erstellt wurden. Weil es immer nur ein prototype Konstrukt für eine Funktion gibt, ist jedoch eine Mehrfachvererbung nicht möglich.

Zunächst wird der *Konstruktor* für die Klasse erstellt - das ist - wie auch in Java u.a. Programmiersprachen üblich - der Name der Klasse. Die Instanzattribute werden mit `this` im Konstruktor erstellt. Für die Erstellung von Methoden werden diese in das prototype-Objekt der Klasse eingefügt.

Um ein Objekt zu erzeugen, muss die o.g. Klasse instanziiert werden. Dazu muss deren Konstruktor mit `new` aufgerufen werden.

Prozedurale Programmierung *procedural programming*

Bei der prozeduralen Programmierung wird die Gesamtaufgabe, die eine Software lösen soll, in kleinere Teilaufgaben aufgelöst. Jede Teilaufgabe für sich ist einfacher zu beschreiben, programmieren und testen. Außerdem kann der entstehende Programmcode in anderen Programmen wieder verwendet werden, ein sehr wichtiger Aspekt in der Softwaretechnik. Die bei der Aufgabenlösung entstehenden Programm-Module werden Prozeduren bzw. Funktionen genannt, wobei zu beachten ist das prozedurale Programmierung keineswegs mit funktionaler Programmierung gleichzusetzen ist - beide folgen unterschiedlichen Programmierparadigmen.

Prozeduren und Funktionen werden üblicherweise nach Aufgabengebieten gruppiert zu Bibliotheken zusammengefasst, die dann verteilt und in beliebig viele andere Programme

eingebunden werden können.

Typische prozedurale Programmiersprachen sind, wie beispielsweise Pascal, C und BASIC, bereits recht betagt. Modernere Programmiersprachen wie Java und C# betrachten den prozeduralen Ansatz als veraltet und setzen stattdessen auf die Weiterentwicklung zur *objektorientierten Programmierung*.

Spezialisierung

Der Begriff Spezialisierung wird auch in der *objektorientierten Programmierung* verwendet. Unter Spezialisierung versteht man die Definitionen einer neuen Klasse als *Unterklasse* einer oder mehrerer anderer Klassen (Oberklassen). Aufgrund der *Vererbung* besitzt die neue Klasse alle *Merkmale* ihrer Oberklassen. In ihrer *Klassendefinition* können jedoch zusätzliche Merkmale (*Erweiterung*) definiert oder geerbte Merkmale überschrieben (Redefinition) werden. Die neue Klasse stellt deshalb eine Spezialisierung ihrer Oberklassen dar. Eine Unterklasse muss jedoch nicht unbedingt zusätzliche Merkmale oder geerbte Merkmale redefinieren, sie kann auch ausschließlich aus der Vereinigung der Merkmale ihrer Oberklasse gebildet sein - hier spricht man auch von aggregierten Klassen.

Subobjekt

Der Begriff Subobjekt wird in der *objektorientierten Programmierung* verwendet. Unter Subobjekt versteht man ein Teilobjekt eines komplexen *Objekts*, d.h. der Wert eines *Attributs* mit *Objekttyp*. Beim objektorientierten Entwurf wird zwischen exklusiven und gemeinsamen Subobjekten unterschieden.

Unter einem Exklusives Subobjekt versteht man speziell ein Subobjekt, das ausschließlich in einem komplexen Objekt verwendet wird. Aufgrund dieser exklusiven Verwendung ist die Existenz exklusiver Subobjekte i.a. an die Existenz komplexer Objekte gebunden. Ein exklusives Subobjekt wird deshalb gemeinsam mit seinem übergeordneten komplexen Objekt

erzeugt und gelöscht.

Ein gemeinsames Subobjekt ist ein Subobjekt, das gleichzeitig in mehreren komplexen Objekten verwendet werden kann. Aufgrund dieser mehrfachen Verwendung ist die Existenz eines gemeinsamen Subobjekts unabhängig von der Existenz komplexer Objekte. Ein gemeinsames Subobjekt wird deshalb unabhängig von seinen übergeordneten komplexen Objekten erzeugt und gelöscht.

Subtyp

In der *objektorientierten Programmierung* (OOP) versteht man unter einem Subtyp speziell einen *Objekttyp*, der durch *Vererbung* definiert wird. In objektorientierten Programmiersprachen, in denen die Namen der Klassen auch die entsprechenden Objekttypen bezeichnen, wird durch die Anordnung der Klassen in einer Klassenhierarchie auch eine entsprechende *Typhierarchie* definiert, in der analog zum Begriff der Subklasse, was der *Unterklasse* entspricht, der Begriff des Subtyps verwendet wird.

Typhierarchie

Typhierarchie ist ein Begriff, der in der *objektorientierten Programmierung* verwendet wird. Die Typhierarchie ordnet Typen in Obertyp-Untertyp-Beziehungen an. Sie basiert darauf, dass sich Operationen auf Werten eines Untertyps genauso verhalten wie auf Werten des Obertyps. Handelt es sich um *Objekttypen*, bedeutet das, dass die den Untertyp realisierende Klasse aus der Klasse des Obertyps durch *Vererbung* mit Übernahme des Protokolls entstanden ist. In vielen objektorientierten Sprachen wird dies stets erzwungen.

Unterklasse

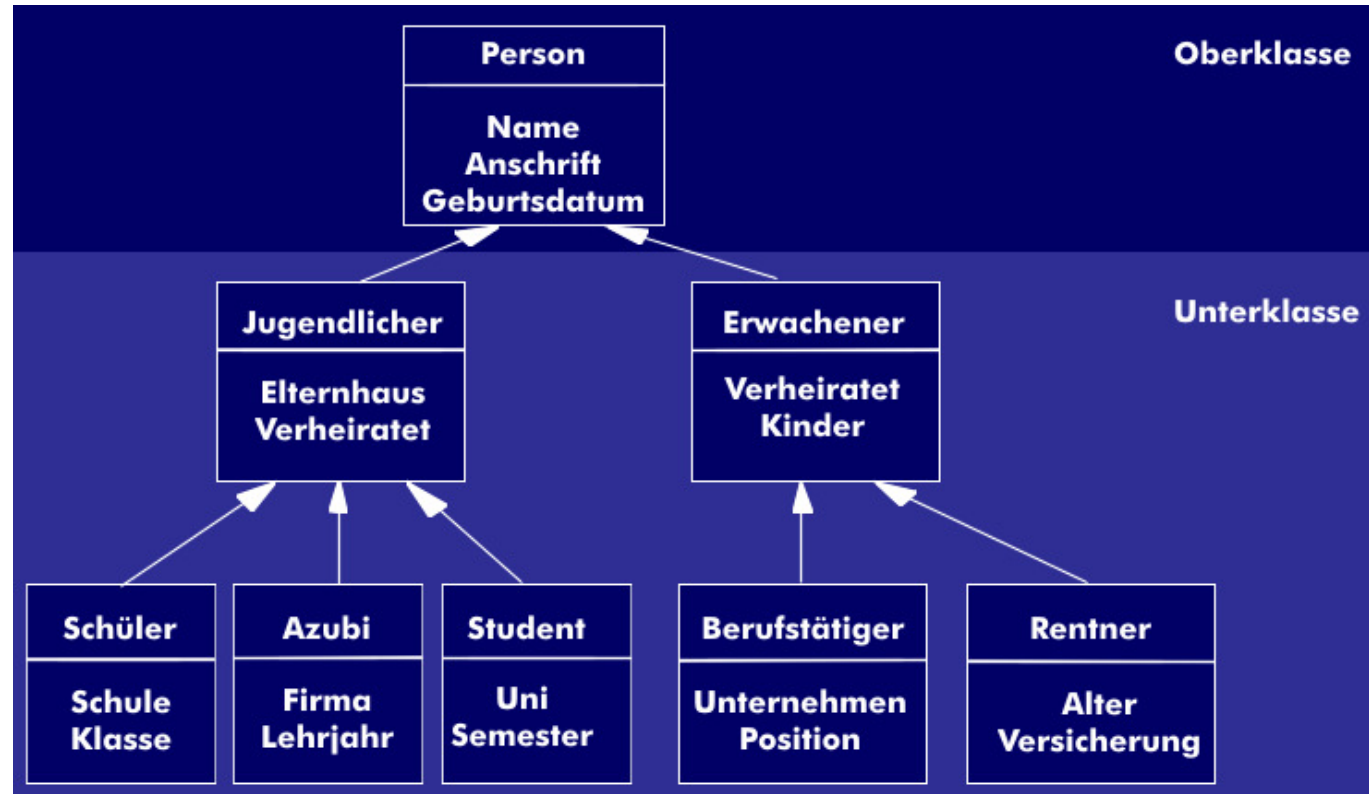
In der *objektorientierten Programmierung* definiert eine Unterklasse (auch als Subklasse bezeichnet) eine Klasse, deren *Merkmale* durch *Vererbung* aus ein oder mehreren anderen Klassen (Oberklassen) übernommen werden. In Abhängigkeit von der Anzahl der

Objektorientierung

Vererbungsstufen (Anzahl der Klassen zwischen einer Unter- und einer Oberklasse) spricht man auch von den direkten oder indirekten Unterklassen einer Klasse.

Vererbung *inheritance*

Vererbung ist eines der grundlegenden Prinzipien der *objektorientierten Programmierung*. Dort können von bestehenden Klassen ausgehend neue Klassen erstellt werden, die zunächst die



Vererbung mit Unter- und Oberklassen

gleichen Eigenschaften und *Methoden* besitzen wie die Ausgangsklasse. Die neue Klasse wird als abgeleitete Klasse oder *Unterklasse* bezeichnet, die Ausgangsklasse als Super- bzw. Oberklasse. Die abgeleitete Klasse kann die von ihrer Superklasse geerbten Eigenschaften und Methoden überschreiben oder durch zusätzliche ergänzen.

Vererbung wird üblicherweise in der Form eingesetzt, dass Unterklassen speziellere Fähigkeiten haben als Ihre Ahnen. Der Prozess des Ableitens wird deshalb auch als *Spezialisierung* bezeichnet. Die Superklasse implementiert also allgemeine Fähigkeiten, die von allen Unterklassen genutzt werden können und sollen, während sich die Unterklassen darauf aufbauend auf andere Fähigkeiten spezialisieren. Dabei kann fast jede Unterklasse wiederum Superklasse für eine noch weitergehende Spezialisierung sein. Ausnahmen gibt es in manchen Programmiersprachen, in denen eine Klasse als „Ende der Klassenhierarchie“ deklariert werden kann, um weitere Ableitung zu verhindern.

Es wird zwischen Einfach- und Mehrfachvererbung unterschieden. Bei der Einfachvererbung hat jede Klasse maximal eine Superklasse, bei der Mehrfachvererbung können es mehrere sein. Mehrfachvererbung wird beispielsweise von C++ unterstützt, während modernere Sprachen wie C-Sharp (C#) und Java sie auf Grund des im Vergleich zu den dadurch entstehenden Problemen geringen Nutzens nicht zulassen. Viele Praxisprobleme, für die Mehrfachvererbung wünschenswert ist, lassen sich durch das Entwurfsmuster (Design Pattern) „Schnittstelle“ sogar noch eleganter und allgemeiner lösen.

Unter wiederholter Vererbung versteht man speziell ein mehrfaches Erben der gleichen *Merkmale* aufgrund einer mehrfachen Oberklasse. Die Merkmale einer mehrfachen Oberklasse können abhängig von der Semantik der Vererbung in abgeleiteten Klassen, den Unterklassen, jeweils mehrfach vorhanden sein. Durch wiederholte Vererbung können Konflikte entstehen, die auf der mehrfachen Verwendung der Namen von Merkmalen beruhen.

Herausgeber

Klaus Lipinski
Datacom-Buchverlag GmbH
84378 Dietersburg

ISBN: 978-3-89238-201-0

Objektorientierung

E-Book, Copyright 2010

Trotz sorgfältiger Recherche wird für die angegebenen Informationen keine Haftung übernommen.



Dieses Werk ist unter einem Creative Commons Namensnennung-Keine kommerzielle Nutzung-Keine Bearbeitung 3.0 Deutschland Lizenzvertrag lizenziert.

Erlaubt ist die nichtkommerzielle Verbreitung und Vervielfältigung ohne das Werk zu verändern und unter Nennung des Herausgebers. Sie dürfen dieses E-Book auf Ihrer Website einbinden, wenn ein Backlink auf www.itwissen.info gesetzt ist.

Layout & Gestaltung: Sebastian Schreiber
Titel: © kentoh #24089040 Fotolia.com

Produktion: www.media-schmid.de
Weitere Informationen unter www.itwissen.info