1. Was kennzeichnet objektorientiertes Vorgehen?

Bei der traditionellen **strukturierten Systementwicklung** stehen die Funktionen und Abläufe eines Programms im Vordergrund. Es geht in erster Linie darum, den logischen Ablauf zur Lösung eines Problems zu entwickeln. Ein wichtiges graphisches Tool zur Veranschaulichung von strukturierten Problemlösungen ist das **Struktogramm**.

Bei der **datenorientierten Betrachtung** stehen die Daten und ihre Beziehungen (Relationen) zueinander im Mittelpunkt. Dazu werden **Datenbanken** entwickelt die mit **Entity-Relationship-Diagrammen** abgebildet werden können

Objektorientierte Systeme sind ebenfalls datenorientiert. Daten werden als Objekte bezeichnet, die zu einer Klasse gehören. Jede Klasse hat bestimmte Eigenschaften (**Attribute**).
Zwischen den Klassen bestehen Beziehungen. Im Unterschied zur Datenbanksystemen werden zwischen den Klassen Informationen ausgetauscht. Der Informationsaustausch wird in **Methoden** dargestellt, die einer Klasse zugehörig sind.
Attribute und Methoden gehören zu den Merkmalen einer Klasse.
Objkektorientierte Systeme werden mit mit verschiedenen Diagrammen der **UML** (unified modelling language) abgebildet.

Beispiel:

Eine Schule besteht aus Lehrern und Schülern. Daraus ließe sich ein Datenbanksystem konstruieren, welches aus einer Tabelle für Lehrer und einer Tabelle für Schüler besteht Zwischen diesen Tabellen können Relationen bestehen, wie z.B. ein Lehrer unterrichtet viele Schüler und ein Schüler hat viele Lehrer (1: n).
Wenn nun aber ein bestimmter Anwendungsfall realisiert werden soll, wie z.B. , Lehrer nimmt neuen Schüler auf, oder Schüler meldet sich beim Lehrer krank, so stoßen Datenbanksysteme an ihre Grenzen. Für solche Situationen sind objektorientierte Systeme geeignet.

2. Objektorientiertes Vorgehen

Bei der Softwareentwicklung gewinnt die objektorientierte Vorgehensweise immer mehr an Bedeutung. Deshalb stellt sich die Frage, welches Vorgehensmodell das richtige ist. Es gibt in der Literatur mehrere Modelle, die meist auf dem bekannten **Wasserfallmodell** oder dem **Spiralmodell** aufbauen und diese beiden kombinieren.

Hier noch mal in Kürze die Beschreibung der beiden Modelle mit ihren Vor- und Nachteilen:

Das **Wasserfallmodell** teilt den Prozess der Softwareentwicklung in 4 - 5 klar abgegrenzte Phasen ein (z.B. Analyse, Design, Realisierung, Test, Implementierung).
Die Phasen folgen streng aufeinander.

Vorteile: Der Gesamtprozess wird klar gegliedert und die einzelnen Phasen sind genau beschrieben. Es besteht Klarheit und Übersicht über den aktuellen Fortschritt des Projekts.

Nachteile: Bei großen Projekten ergeben sich lange Zeiträume zwischen den einzelnen Phasen, so dass Neuerungen, Unsicherheiten und Veränderungen in den Anforderungen schwer vorher zu sehen sind und kaum berücksichtigt werden können.
Oftmals ist es nicht möglich, das Endprodukt schon am Projektanfang vollständig und genau zu planen.
Das Wasserfallmodell bietet kaum die Möglichkeit, auf veränderte Bedingungen und neue Anforderungen zu reagieren.

Das **Spiralmodell** teil den Softwareentwicklungsprozess ebenfalls in Phasen ein. Diese Phasen werden aber im Zeitablauf mehrfach wiederholt, wobei man in jedem neuen Zyklus dem Endprodukt etwas näher kommt.

Vorteile: Änderungen in den Anforderungen können in einem neuen Zyklus berücksichtigt werden. Es gibt nach jedem Zyklus Zwischenergebnisse, die bereits dem Auftraggeber vorgestellt werden können.

Nachteile: Wenn den einzelnen Zyklen des Projekts keine klaren Zeitvorgaben zugrunde liegen, besteht die Gefahr, dass das Projekt zeitlich aus dem Ruder läuft, weil einzelne Phasen länger dauern als geplant.
Dem Wunsch des Kunden, nach schnellen Vorablösungen (Releases), die bereits funktionsfähig sind, kann oft nicht entsprochen werden, weil die Zwischenergebnisse der einzelnen Zyklen meist noch nicht die Anforderungen des Kunden erfüllen.

Unter Einbeziehung der Erkenntnisse der Vor- und Nachteile der beiden Modelle und unter Berücksichtigung der Erkenntnisse der XP-Programmierung (Extreme Programming) wollen wir mit einem Modell arbeiten, welches für die objektorientierte Vorgehensweise tauglich ist , ohne dass wir wesentlich neue Begrifflichkeiten einführen müssen. Dabei orientiere ich mich teilweise an der von Rau (Rau, Karl-Heinz, Objektorientierte Systementwicklung, Vieweg Verlag, 1. Aufl. 2007), ohne jedoch die Begriffe seiner Phaseneinteilung zu verwenden.

3. Das objektorientierte Vorgehensmodell

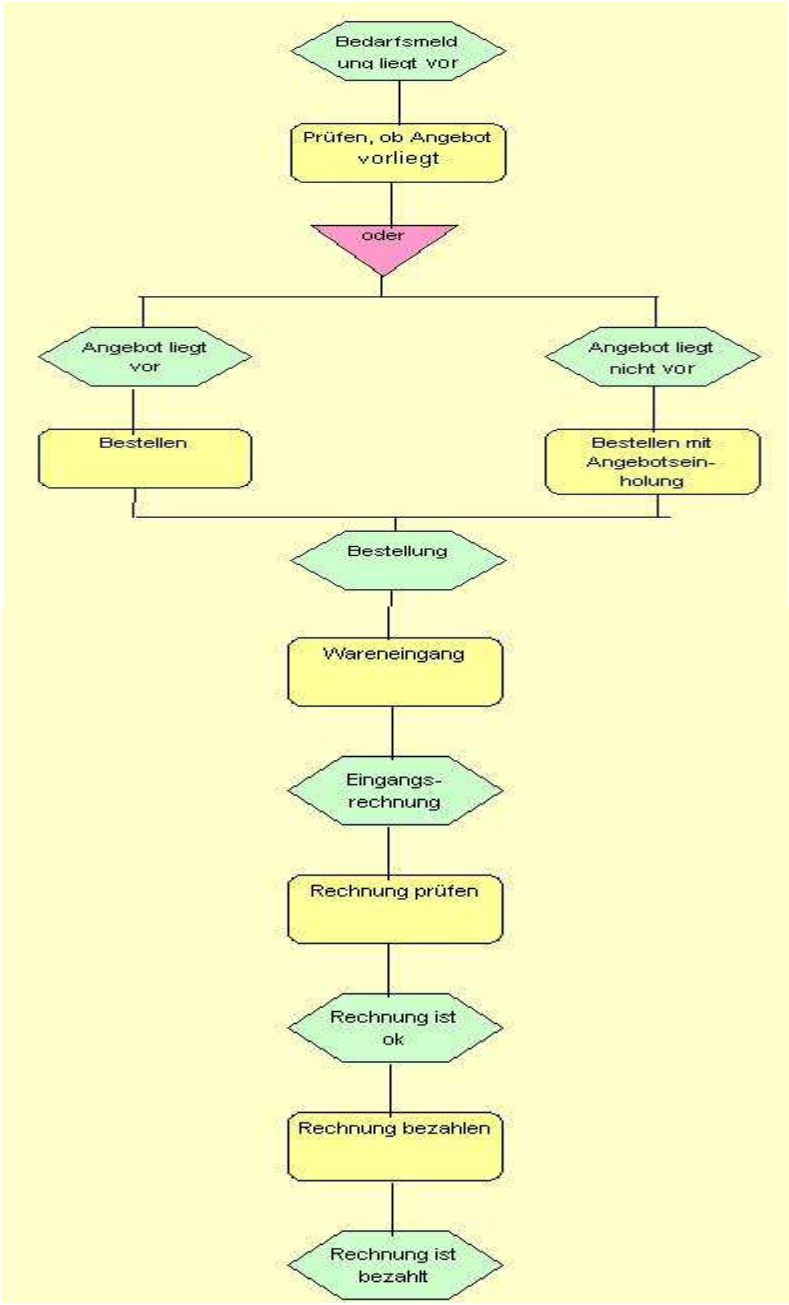
Phasen	Aktivitäten
Analyse	Erarbeiten der Vorstellung der entgültigen Softwarelösung durch Analyse der bestehenden **Geschäftsprozesse** und Darstellung der wichtigsten **Anwendungsfälle** (**use cases**). Bei Weiterentwicklung bestehender Systeme fällt diese Phase eher kurz aus. Die Geschäftsprozessanalyse stellt den bestehenden Ist-Zustand dar und soll Schwachstellen im System aufdecken **(Istanalyse**). Aufgrund dieser Erkenntniss wird der Geschäftsprozess neu modelliert (**Sollanalyse**). Dadurch sollen Auftragnehmer und Auftraggeber ein gemeinsames Verständnis für das künftige System entwickeln. _____ **Werkzeuge**: Ereignisgesteuerte Prozessketten (EPK), Use-Case-Diagramme **Fragestellungen und Beispiele:** Was ist ein Geschäftsprozess? Was ist ein Anwendungsfall?
Design	In der Designphase wird die Architektur des Softwaresystems entwickelt. Als erstes geht es darum, anhand der im Geschäftsprozess benötigten Daten ein **Klassenmodell** zu erstellen. Dabei sollen auch die Beziehungen (Assoziationen) zwischen den Klassen, sowie die Attribute der Klassen identifiziert werden. Im zweiten Schritt geht es darum, einzelne Arbeitsschritte in einem Anwendungsfall zu identifizieren und in eine zeitliche und logische Reihenfolge zu bringen. Außerdem muss dargestellt werden, welche Operationen (**Methoden**) die einzelnen Klassen durchführen müssen, um die gewünschten Ergebnisse zu erreichen. Ebenfalls müssen die **Schnittstellen** zwischen den Akteuren und dem System beschrieben werden. Dieses können z.B. Eingabemasken für die Dialoggestaltung sein oder Ausgabemasken bzw. Formulare. Besonders in der Designphase kommen sogenannte **CASE-Tools** (Computer Aided Software Engineering) zum Einsatz, die die Entwickler bei der Arbeit unterstützen. Case-Tools sind mehr als bloße Graphikwerkzeuge. Die einzelnen Tools sind so miteinander verknüpft, dass Veränderungen in einer Darstellung automatisch in anderen Darstellungen übernommen werden. außerdem können bei der graphischen Modellierung bereits Teile des Quellcodes erzeugt werden. _____ **Werkzeuge:** Klassendiagramme(mit Assoziationen und Kardinalitäten), Sequenzdiagramme, Aktivitätsdiagramme, Kommunikations-diagramme **Fragestellungen und Beispiele:** Wie identifiziert man eine Klasse? S. Beispiel
Codierung	Die Ergebnisse der Designphase werden nun in eine objektorientierte Programmiersprache umgesetzt. Bei Verwendung von Case-Tools werden die Grundstrukturen der **Klassen** schon vorliegen, so dass lediglich noch die Funktionsweise der Methoden programmiert werden muss. In den **Applikationen** werden die codierten Klassen verwendet, Objekte erzeugt und Methoden aufgerufen, die das gewüschte Systemergebnis erzeugen. _____ **Werkzeuge**: Eclipse, NetBeans, JavaEditor, Joe, BlueJ u. a.
Test	Ausführlicher Test des Systems mit Testdaten.
Implementierung	Übergabe der Software an den Auftraggeber, Schulung der Benutzer usw. Am Ende steht eine **Release**-Version der Software.

4. Fragestellungen und Beispiele

4.1 Was ist ein Geschäftsprozess

Ein Geschäftsprozess besteht aus einer Folge von **Geschäftsanwendungsfällen**, die zu einem bestimmten Ergebnis führt (z.B. einer Rechnung, einem Angebot). Ein Geschäftsprozess wird von einem **Ereignis** ausgelöst und endet mit einem Ereignis (Sechseckdarstellung). Zwischen zwei Ereignisse liegt immer eine **Aktivität** oder **Funktion** (abgerundetes Rechteck). Zur Darstellung von Geschäftsprozessen werden häufig **Ereignisgesteuerte Prozessketten** (**EPK**) verwendet (siehe unten). Innerhalb der UML-Darstellung ist auch das **Aktivitätsdiagramm** geeignet.

Beispiel: EPK-Darstellung des Geschäftsprozesses „Beschaffung durchführen".

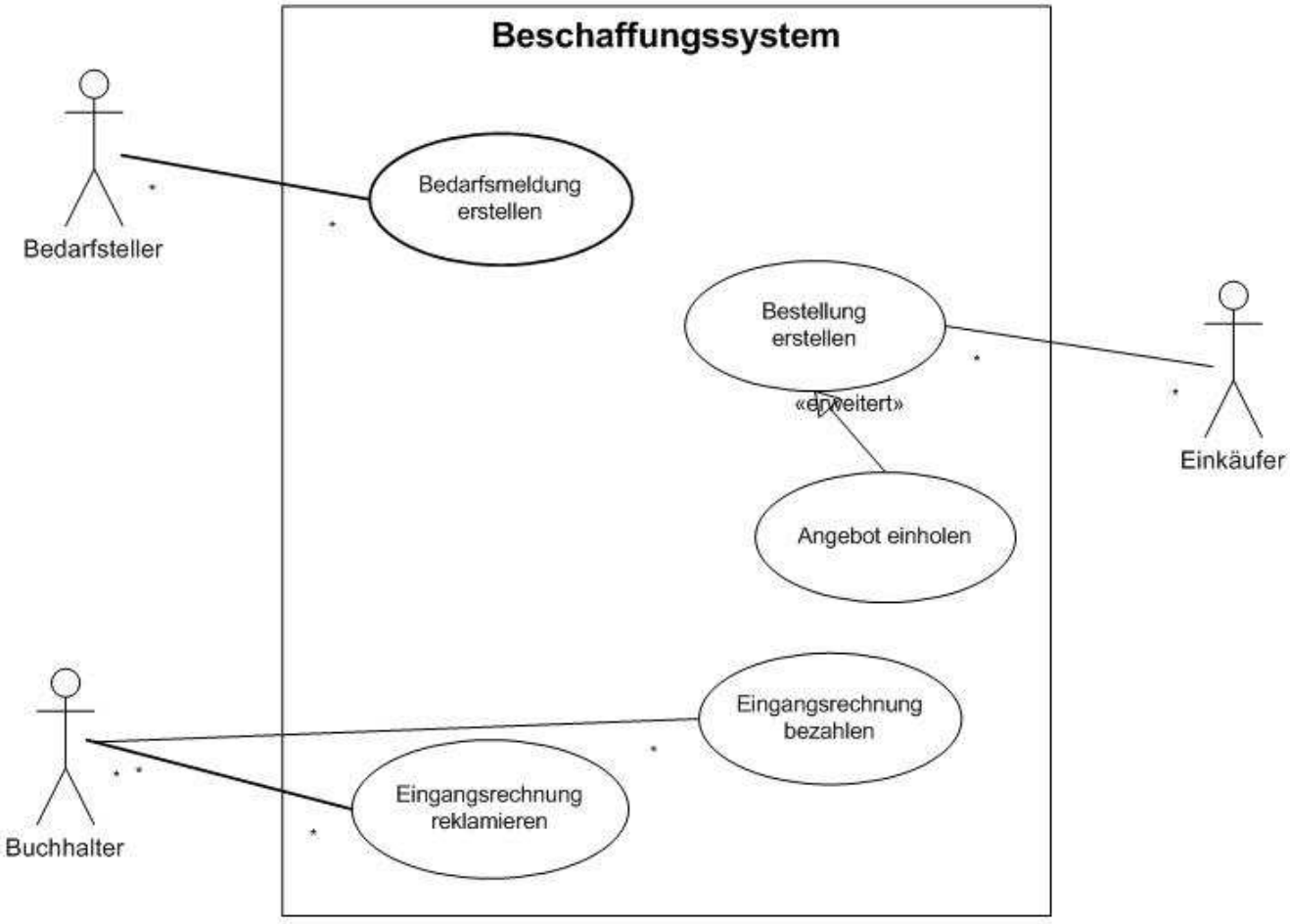


4.2 Was ist ein Anwendungsfall

Ein Anwendungsfall ist eine Folge von Aktivitäten innerhalb eines Geschäftsprozesses, die zu einem Teilergebnis führt. In obigem Geschäftsprozess „Beschaffung durchführen" sind folgende Anwendungsfälle vorhanden.

- Erstellen einer Bedarfsmeldung
- Angebot einholen (nicht immer erforderlich)
- Angebot erfassen
- Bestellung schreiben
- Wareneingang bearbeiten
- Eingangsrechnung prüfen
- Eingangsrechnung reklamieren (evtl.)
- Eingangsrechnung bezahlen

Die Darstellung von Anwendungsfällen geschieht in einem **Anwendungsfalldiagramm (use-case-Diagramm**)
Einem Beschaffungssystem könnte beispielsweise folgendes use-case-Diagramm zugrunde liegen:



Aufgabe: Vervollständigen Sie das Anwendungsfalldiagramm durch die fehlenden Anwendungsfälle

4.3 Wie identifiziert man eine Klasse

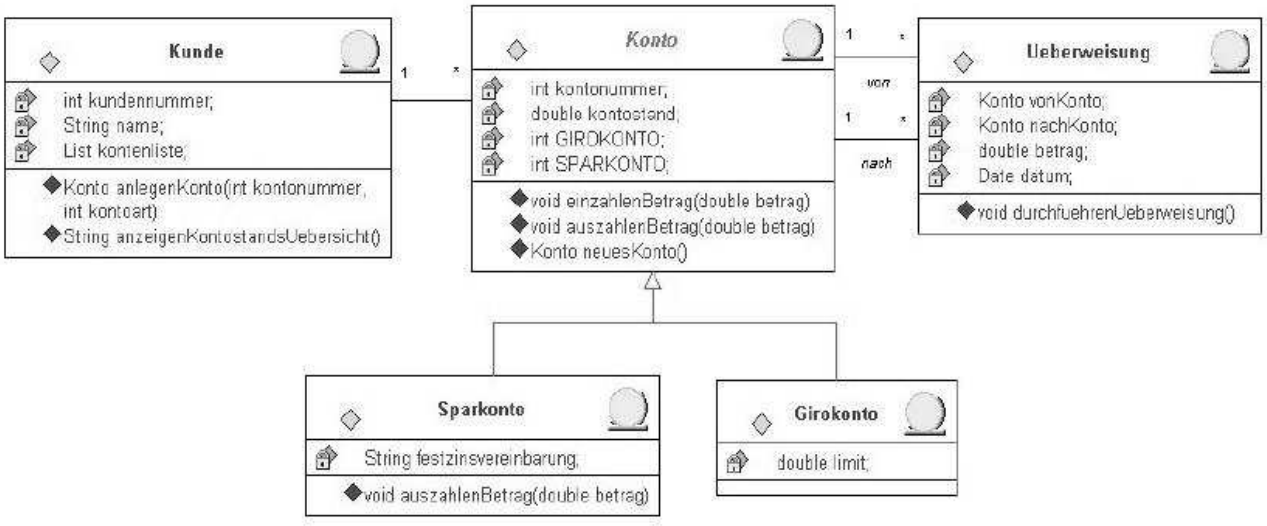
Grundlage des objektorientierten Systems ist die Identifikation der Klassen. Das Klassendiagramm stellt die statische Sicht auf das Systems dar.

Beispiel: Wir wollen eine einfaches Banksystem entwerfen. Die Kunden einer Bank können Sparkonto oder Girokonten haben. Sie sollen Einzahlen, Abheben und Überweisen können.
Erstellen Sie ein Klassendiagramm des Systems und nennen Sie die Anwendungsfälle.

Eine einfache Methode, die Klassen eines Systems zu finden, ist es nach den Substantiven in der Beschreibung des Systems zu suchen. In vielen Fällen finden wir dadurch schon die benötigten Klassen. Diese Methode wird **Nominalextraktionsmethode** genannt. Zusätzlich ist es sinnvoll zu untersuchen, welche **Dokumente** und **Formulare** im System benötigt werden, (z.B. Rechnung, Überweisung ...) Dadurch findet man weitere Klassen.

In unserem Beispiel finden wir die Substantive Kunden, Bank, Sparkonto und Girokonto.
Bei den Substantiven Einzahlen, Abheben und Überweisen handelt es sich wohl um Anwendungsfälle. Da für die Überweisung aber ein Formular benötigt wird, haben wir hier eine weitere Klasse.

Nach grober Anordnung der Klassen auf dem Papier geht es noch darum, die Beziehungen zwischen den einzelnen Klassen zu finden, zu benennen und ihnen evtl. eine Zahl zuzuordnen. Zum Beispiel kann ein Kunde mehrere Konten haben oder von einem Konto können mehrere Überweisungen getätigt werden (1 ..*). Diese Beziehungen werden in **Assoziationen** und **Kardinalitäten** ausgedrückt.

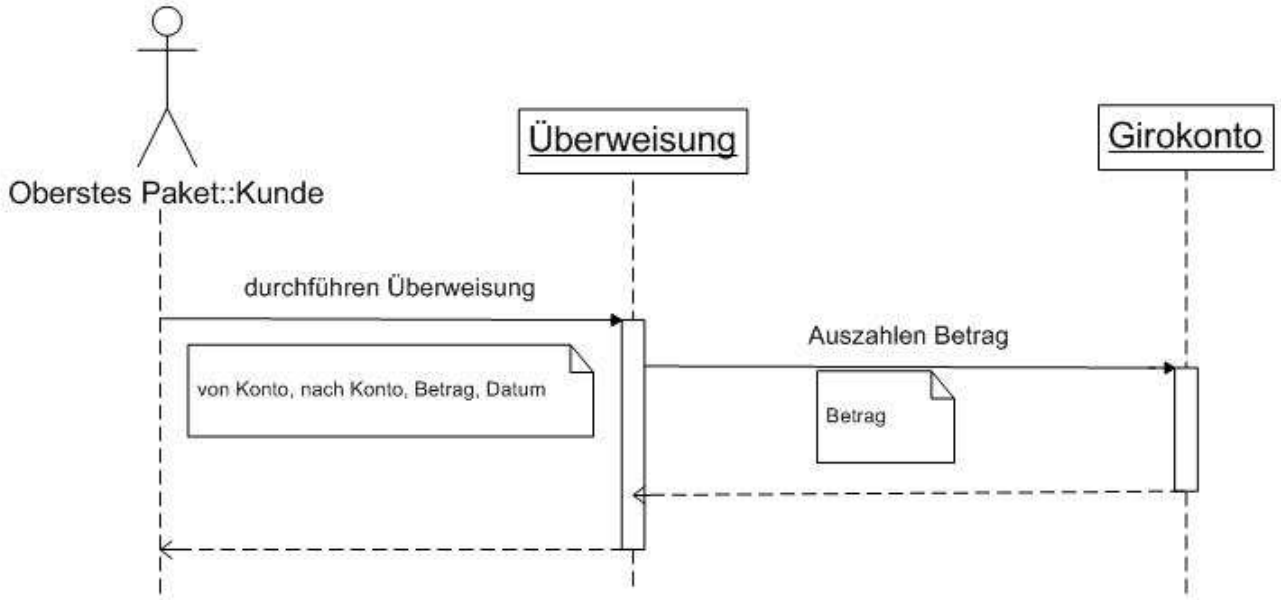


Aufgabe: Erstellen Sie für das obige System ein use-case-Diagramm.

4.4 Dynamische Sicht auf das System

Nachdem die Architektur des Systems feststeht, werden die einzelnen Anwendungsfälle geplant. Es geht darum, in welcher logischer und zeitlicher Reihenfolge Aktionen zwischen den Klassen stattfinden. Diese Phase erinnert am ehesten an die strukturierte Programmierung, bei der mit Hilfe von Struktogrammen der logische Ablauf dargestellt wurde.

Das **Sequenzdiagramm** zeigt den Anwendungsfall **Überweisung durchführen.**
Auslöser ist der Kunde, es werden die beiden Objekte *Überweisung* und *Girokonto* berührt.
Das Objekt Überweisung erhält als Nachricht den Aufruf der Methode *‚durchführen Überweisung'* mit den Argumenten *von Konto*, *nach Konto*, *Betrag* und *Datum*. Diese Methode wiederrum ruft die Methode *‚Auszahlen Betrag'* des Girokonto-Objektes auf.



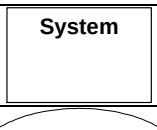
5. Die wichtigsten UML-Diagramme

5.1 Das Anwendungsfalldiagramm (use case Diagramm)

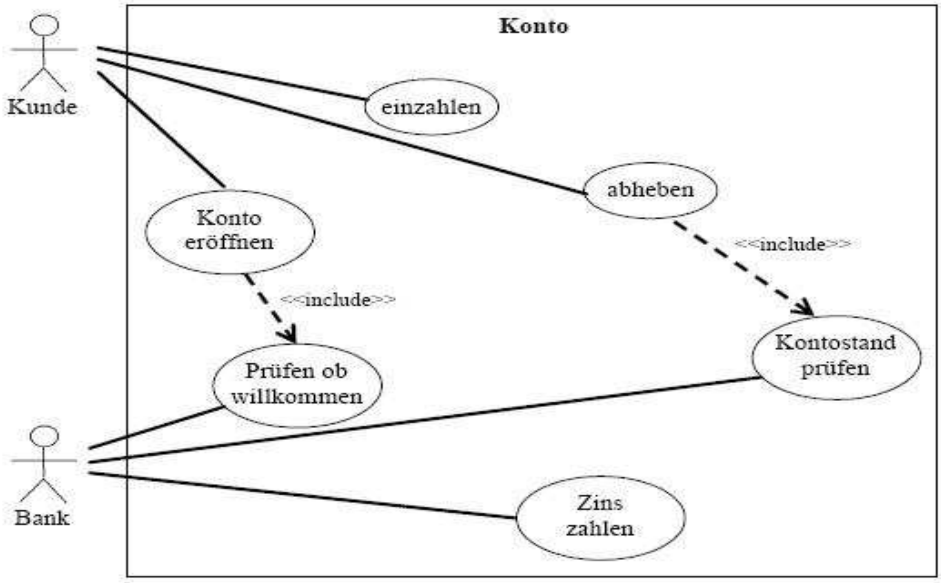
Das Anwendungsfalldiagramm ist ein *Verhaltensdiagramm*. Es zeigt eine bestimmte Sicht auf das *erwartete* Verhalten eines Systems und wird deshalb für die Spezifikation der Anforderungen an ein System eingesetzt. Anwendungsfalldiagramme beschreiben die Beziehungen zwischen einer Gruppe von Anwendungsfällen und den teilnehmenden Akteuren.
Dabei ist zu beachten, dass ein Anwendungsfalldiagramm nicht das Systemdesign widerspiegelt und damit keine Aussage über die Systeminterna trifft. Anwendungsfalldiagramme werden zur Vereinfachung der Kommunikation zwischen Entwickler und zukünftigen Nutzer bzw. Kunde erstellt. Sie sind vor allem bei der Festlegung der benötigten Kriterien des zukünftigen Systems hilfreich. Somit treffen Anwendungsfalldiagramme eine Aussage, *was* zu tun ist, aber nicht *wie* das erreicht wird.

Anwendungsfälle werden durch **Ellipsen** die den Namen des Anwendungsfalles tragen und einer Menge von beteiligten Objekten (**Akteuren**) dargestellt. Zu jedem Anwendungsfall gibt es eine Beschreibung in Textform. Die entsprechenden Anwendungsfälle und Akteure sind durch **Linie**n miteinander verbunden. Akteure können durch Strichmännchen dargestellt werden. Die **Systemgrenze** wird durch einen Rahmen um die Anwendungsfälle symbolisiert. **Include-Beziehungen** bestehen zwischen Anwendungsfällen, die einen anderen Anwendungsfall beinhalten. Diese werden durch gestrichelte Linien dargestellt.

Das use case Diagramm beinhaltet folgende Elemente:

	Der Rahmen stellt das Anwendungssystem dar, innerhalb dessen der Anwendungsfall realisiert wird.
	Ein Anwendungsfall ist ein Teilproblem, welches innerhalb eines Anwendungssystems gelöst werden soll. Der Anwendungsfall wird immer von einem oder mehreren Akteuren oder anderen Anwendungsfällen ausgelöst
	Der Akteur kann eine Person oder ein anderes Anwendungssystem sein, welches einen Anwendungsfall auslöst
	Die Assoziation besteht verbindet Akteure und Anwendungsfälle
	include – oder extend-Beziehung. Eine include-Beziehung besteht zwischen einem Anwendungsfall, der einen anderen Anwendungsfall beinhaltet. Eine extends-Beziehung besteht, wenn eine Anwendungsfall durch einen anderen Anwendungsfall erweitert wird, wenn eine bestimmte Bedingung eintritt.

Allgemeines Beispiel



5.2 Klassendiagramme

Das Klassendiagramm ist die wichtigste Diagrammart in der UML. Es beschreibt den Aufbau einer Klasse mit Attributen und Methoden. Es stellt sozusagen den Bauplan dar für die Objekte, die dann von dieser Klasse erzeugt werden. Folgende Regeln sollten für die Erstellung eines Klassendiagramms beachtet werden:

Das Klassendiagramm besteht aus einem Rechteck, welches in drei Bereiche unterteilt ist.
In der **oberen Zeile** steht der **Klassenname** zentriert und fett gedruckt.
In der **zweiten Zeile** werden die **Attribute** aufgelistet. Das Minuszeichen vor der Attributsbezeichnung bedeutet, dass das Attribut gekapselt ist und mit dem Schlüsselwort **private** versehen.
Außerdem ist es möglich, den Datentyp des Attributes hinzuzufügen. Z.B*. –kilometerstand ; int*
In der **dritten Zeile** stehen die **Methoden**. Diese sind mit dem Schlüsselwort **public** versehen, was an dem Pluszeichen zu erkennen ist. Methoden können mit Übergabe- oder Rückgabewert noch näher beschrieben werden.
Zur Darstellung von Klassendiagrammen mit mehreren Klassen und deren Beziehungen zueinander, können Attribute und Methoden zunächst weggelassen werden.

Beziehungen zwischen Klassen

In praktischen Anwendungsfällen der OOP hat man es normalerweise nicht nur mit einer einzigen Klasse zu tun, sondern mit mehreren Klassen zwischen denen Beziehungen bestehen.

Fallsituation:
Eine Bank hat Kunden und Konten. Jeder Kunde hat mindestens ein Konto

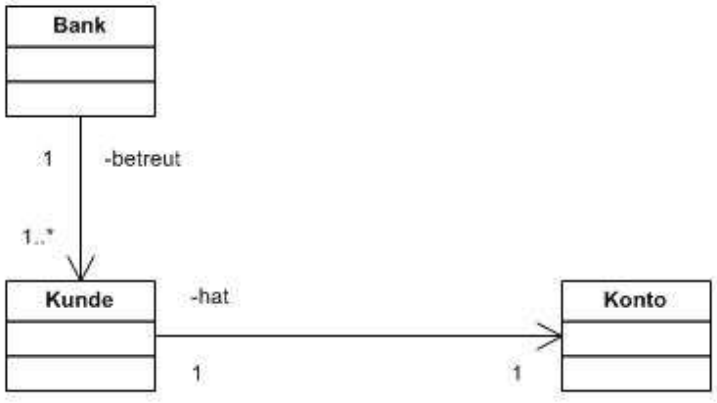
Zunächst wollen wir diesen Fall in einem vollständigen Klassendiagramm mit Assoziationen und Kardinalitäten darstellen. Aus dem Text heraus identifizieren wir folgende mögliche Klassen:
Bank, **Kunde** und **Konto**.

Zwischen diesen Klassen bestehen logische und mengenmäßige Beziehungen:

Assoziationen: Bezeichnung der Beziehung und der Beziehungsrichtung: z.B. Bank betreut Kunden oder Kunde hat Konto

Kardinalität: Mengenmäßige Beziehung zwischen Klassen: z.B. Eine Bank betreut eine unbekannte Anzahl (aber mindestens 1) Kunden.

Daraus ergibt sich folgendes Klassendiagramm:



Zur besseren Übersichtlichkeit wurden hier keine Attribute und Methoden in der Klasse dargestellt.

Die Pfeilspitze der Assoziationslinien gibt die **Leserichtung** der Assoziation an. Es ist auch möglich, dass es zwischen Klassen mehrere Assoziationen mit unterschiedlicher Leserichtung gibt. Die Art der Assoziation wird durch den **Assoziationsnamen** ausgedrückt (betreut, hat).
(In der korrekten UML-Syntax sollte die Leserichtung durch eine ausgefüllte Pfeilspitze hinter dem Assoziationsnamen dargestellt werden.)

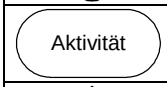
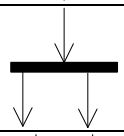
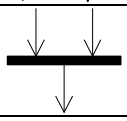
5.3 Objektdiagramme

Wenn wir nun nach dem Plan der Klasse Auto ein konkretes Auto bauen, so entsteht ein **Objekt**. Ein Objekt eine eigene Bezeichnung, z.B. meinAuto. Objekte werden in der UML mit einem Objektdiagramm dargestellt und mit konkreten Eigenschaften beschrieben.

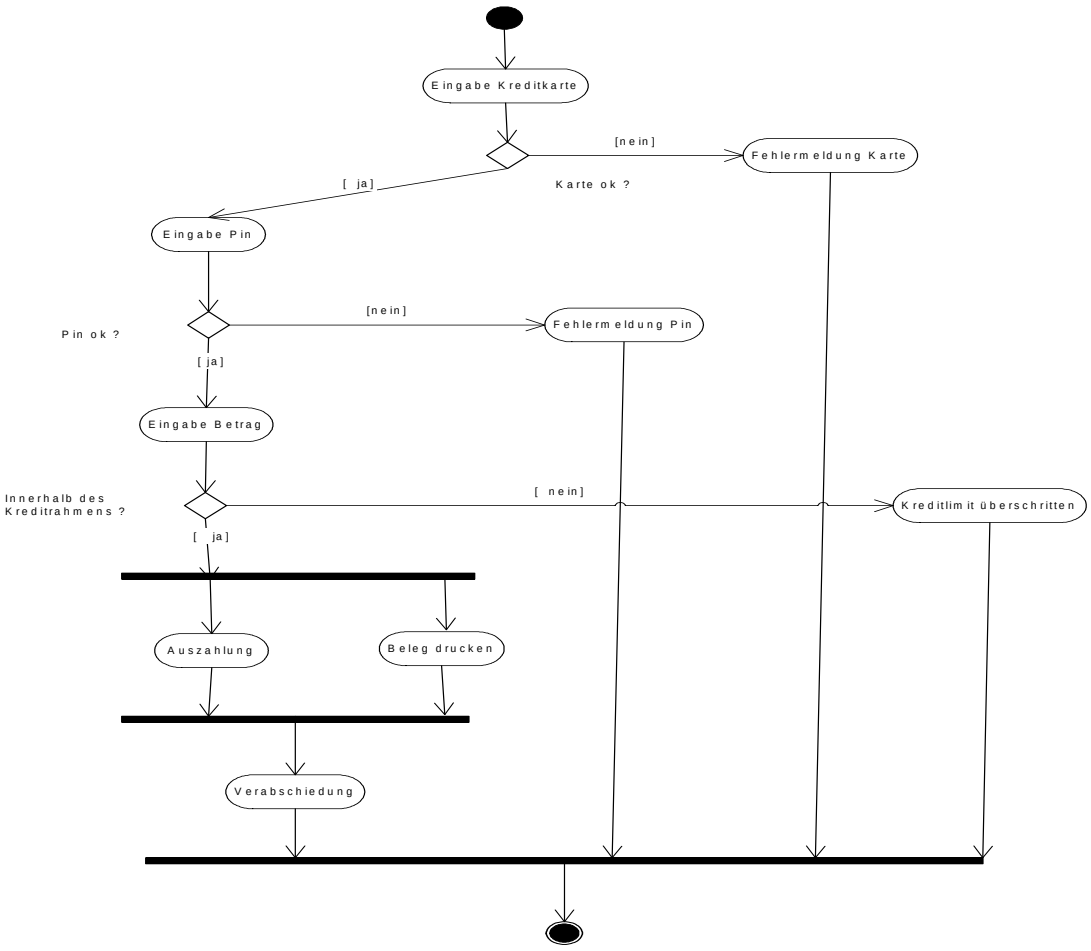
meinAuto : Auto	
kw	70
farbe	rot
km	25600
tankinhalt	60

5.4 Aktivitätsdiagramme

Use-Case, Klassen- und Objektdiagramme beschreiben den Zustand des Systems und damit die **statische Sicht** auf das System. Für die Darstellung von Abläufen benötigen wir eine **dynamische Sichtweise**.
Das Aktivitätsdiagramm stellt den Ablauf des zu entwickelnden Programms dar. Es entspricht etwa der Programmdarstellung durch ein Struktogramm oder einen Programmablaufplan bei der strukturierten Programmierung. Ein Aktivitätsdiagramm enthält sechs unterschiedliche Elemente:

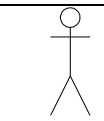
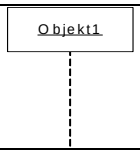
	Startpunkt einer Aktivität
	Endpunkt einer Aktivität
	Beschreibt die Aktion, die vom Programm auszuführen ist
	Entscheidung. In Abhängigkeit von einer Bedingung können verschiedene Wege beschritten werden
	Gabelung. Eine Aktivität gabelt sich in zwei weitere Aktivitäten auf.
	Zusammenführung. Zwei Aktivitäten werden zu einer Aktivität zusammengeführt.

Beispiel: Ein Bankkunde möchte an einem Geldautomat Geld abheben. Zunächst wird die Gültigkeit seiner Kreditkarte geprüft. Danach gibt er seine Pin ein. Ist die Pin gültig, kann er einen Betrag eingeben. Falls sein Kontostand innerhalb des Kreditrahmens liegt, wird der Betrag ausgezahlt und gleichzeitig ein Beleg gedruckt, andernfalls wird die Auszahlung verweigert



5.5 Sequenzdiagramme

Im Sequenzdiagramm wird der Nachrichtenaustausch (**Botschaften**) zwischen den Objekten in zeitlicher Reihenfolge dargestellt. In den Spalten werden die angesprochenen Objekte dargestellt, in den Zeilen die Reihenfolge der Aktivitäten, die sich hier als Nachricht darstellen. Rechtecke bezeichnen Rechenzeiten des Computers und können näher erläutert werden. Im Gegensatz zum Aktivitätsdiagramm bezieht das Sequenzdiagramm die beteiligten Objekte mit in das Diagramm ein. Ein Sequenzdiagramm besteht aus folgenden Grundelementen:

	Das auslösende Objekt, der Akteur, der den Anwendungsfall auslöst
	Ein beteiligtes Objekt, welches eine Botschaft über mittelt oder ein Botschaft erhält. Die gestrichelte Linie ist die Lebenslinie des Objekts.
Botschaft 	Eine Botschaft, die mittels eines Methodenaufrufes übermittelt wird. Es können die Methodenbezeichnung sowie die Übergabeparameter genannt werden.
Rückgabebotschaft	Der Antwort auf eine Botschaft. Die Antwort kann auch als Text auf der Linie stehen.
	Stellt den Zeitraum dar, der zwischen der Botschaft und der Rückmeldung liegt. Symbolisiert die Rechnerzeit in der die Daten verarbeitet werden. Innerhalb des Kästchens kann auch eine kurze Beschreibung des Vorgangs stehen.

Beispiel: Der Filialleiter eines Supermarktes ruft abends von seinem PC aus die Tagesumsätze der beiden Kassen ab und ermittelt den täglichen Filialumsatz. Das Objekt FilialPC gehört zur Klasse Filiale und besitzt die Methode ***BerechneTagesumatzFiliale***. Diese Methode ruft die Methoden ***getEinlage*** und ***getKassenbestand*** bei den einzelnen Kassenobjekten auf, die jeweils die Einlage bzw. den Kassenbestand als Rückgabeparameter liefern. Die Methode ***BerechneTagesumatzFiliale*** berechnet die Kassenumsätze, addiert diese auf und liefert dann das Ergebnis zurück.