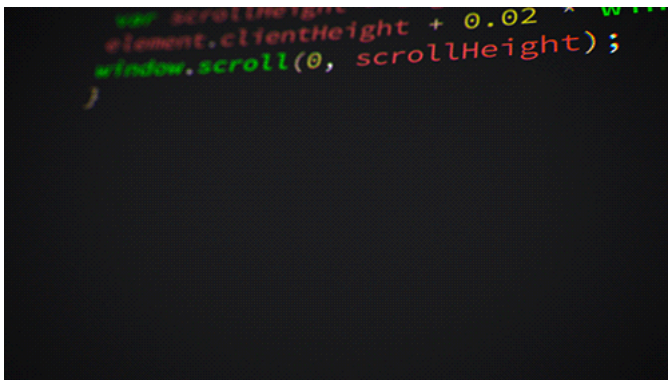


Modellierung von Software

| Parameter | Kursinformationen |
|----------------------|---|
| Veranstaltung: | Vorlesung Softwareentwicklung |
| Semester | Sommersemester 2022 |
| Hochschule: | Technische Universität Freiberg |
| Inhalte: | Ausgewählte UML Diagrammtypen |
| Link auf den GitHub: | https://github.com/TUBAF-lfi-LiaScript/VL_Softwareentwicklung/blob/master/14_UML_ModellierungII.md |
| Autoren | Sebastian Zug, Galina Rudolf & André Dietrich |



Hinweis auf die Prüfungen

- Softwareentwicklung
 - Miniprojekt als Prüfungsleistung - [Link](#) + Beispielprojekt - [Link](#)
- Einführung in die Softwareentwicklung
 - Voraussetzung für den positiven Abschluss ist die erfolgreiche Bearbeitung der letzten Übungsaufgabe. + Es ist keine Anmeldung zur Prüfung notwendig!

Neues aus GitHub

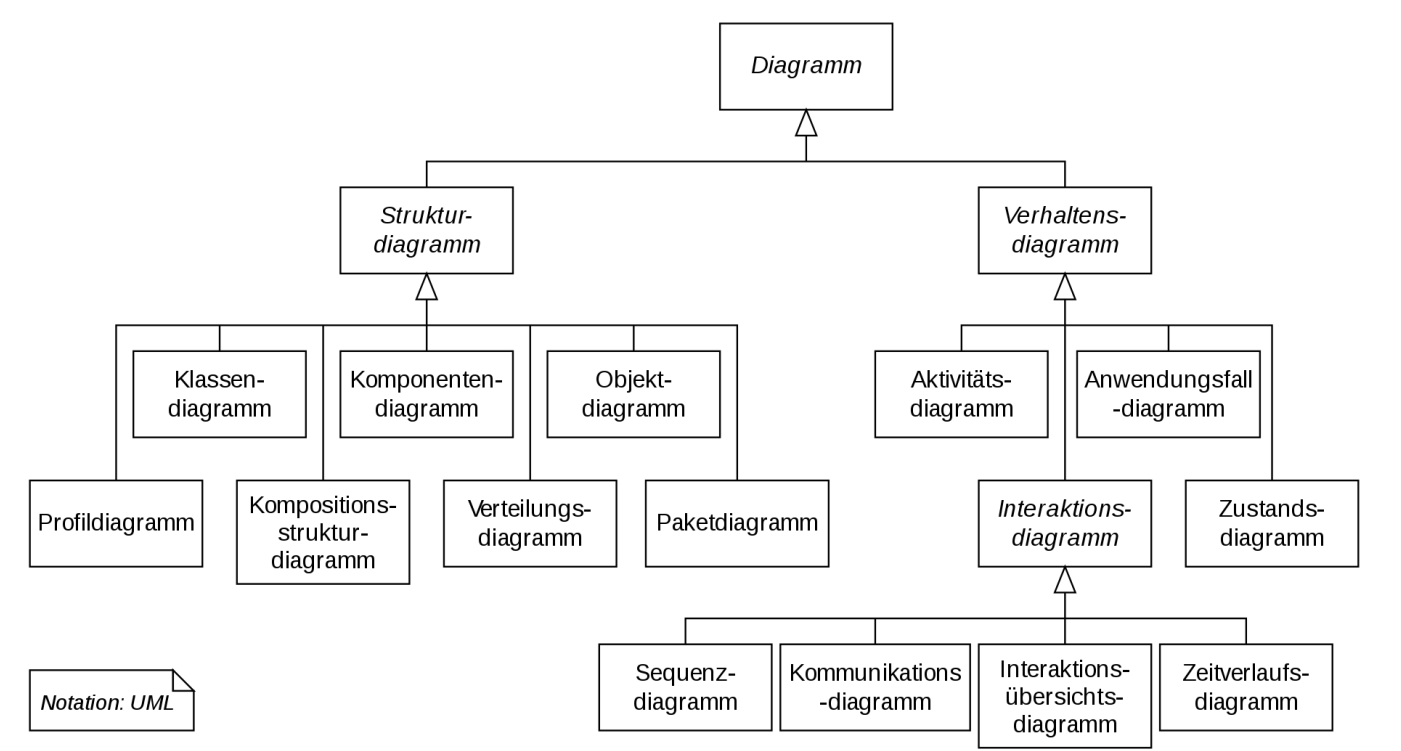
Statistiken der Git-Aktivitätsbalken

| TeamKey | branch_count | commit_count | release_count | stars_count |
|---------|--------------|--------------|---------------|-------------|
| 0 | 2 | 8 | 1 | 2 |
| 1 | 3 | 2 | 1 | 0 |
| 2 | 2 | 5 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 |
| 4 | 2 | 11 | 1 | 0 |
| 5 | 1 | 17 | 1 | 0 |
| 6 | 3 | 12 | 2 | 0 |
| 7 | 2 | 28 | 0 | 0 |
| 8 | 2 | 2 | 2 | 0 |
| 9 | 2 | 4 | 0 | 1 |
| 10 | 1 | 15 | 1 | 0 |
| 11 | 1 | 6 | 1 | 0 |
| 12 | 2 | 10 | 4 | 0 |
| 13 | 3 | 24 | 1 | 0 |
| 14 | 3 | 11 | 1 | 2 |
| 15 | 2 | 13 | 1 | 0 |
| 16 | 2 | 5 | 0 | 0 |
| 17 | 2 | 6 | 1 | 0 |

Commit Messages

| Messag | Häufigkeit |
|--|------------|
| Update CSharpBasics.md | 40 |
| Update team.config | 23 |
| Initial commit | 18 |
| Inital task selection - Variant_1.md | 11 |
| Update uebung.md | 10 |
| Create CSharpBasics.md | 9 |
| Update CSharpBasics.txt | 7 |
| Inital task selection - Variant_0.md | 7 |
| Add files via upload | 5 |
| Update Aufgabe3.md | 4 |
| Update csharpbasics.md | 3 |
| Test | 2 |
| tabelle | 2 |
| Merge pull request #2 from Ifi-Softwareentwick.. | 2 |
| Update converttxttomd.cs | 2 |

UML Diagrammtypen



[WikiUMLDiagrammTypes]

Im folgenden werden wir uns aus den beiden Hauptkategorien jeweils folgende Diagrammtypen genauer anschauen:

- Verhaltensdiagramme
 - Anwendungsfall Diagramm
 - Aktivitätsdiagramm
 - Sequenzdiagramm
- Strukturdiagramm
 - Klassendiagramm
 - Objektdiagramm

[WikiUMLDiagrammTypes] <https://upload.wikimedia.org/wikipedia/commons/thumb/d/da/UML-Diagrammhierarchie.svg/1200px-UML-Diagrammhierarchie.svg.png>,
Autor "Stkl"- derivative work: File: UML-Diagrammhierarchie.png: Sae1962, CC BY-SA 4.0

Anwendungsfall Diagramm

Das Anwendungsfalldiagramm (Use-Case Diagram) abstrahiert das erwartete Verhalten eines Systems und wird dafür eingesetzt, die Anforderungen an ein System zu spezifizieren.

Ein Anwendungsfalldiagramm stellt keine Ablaufbeschreibung dar! Diese kann stattdessen mit einem Aktivitäts-, einem Sequenz- oder einem Kollaborationsdiagramm (ab UML 2.x Kommunikationsdiagramm) dargestellt werden.

Basiskonzepte

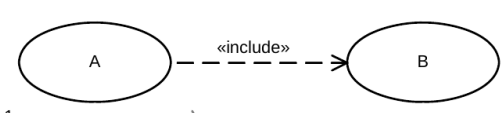
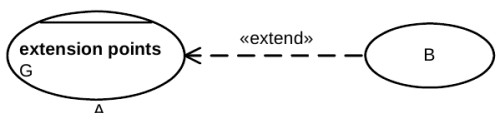
Elemente:

- Systemgrenzen werden durch Rechtecke gekennzeichnet.
- Akteure werden als „Strichmännchen“ dargestellt, dies können sowohl Personen (Kunden, Administratoren) als auch technische Systeme sein (manchmal auch ein Bandsymbol verwendet). Sie ordnen den Symbolen Rollen zu
- Anwendungsfälle werden in Ellipsen dargestellt. Üblich ist die Kombination aus Verb und ein Substantiv `Kundendaten Ändern`.
- Beziehungen zwischen Akteuren und Anwendungsfällen müssen durch Linien gekennzeichnet werden. Man unterscheidet "Association", "Include", "Extend" und "Generalization".

Einfaches Anwendungsfalldiagramm für einen Online-Versand

Verfeinerung

Use-Case Diagramme erlauben die Abstraktion von Elementen auf der Basis von Generalisierungen. So können Akteure von einander erben und redundante Beschreibungen von Verhalten über `<<extend>>` oder `<<include>>` (unter bestimmten Bedingungen) erweitert werden.

| | | |
|----------------|---|--|
| |  |  |
| | <code><<include>></code> Beziehung | <code><<extend>></code> Beziehung |
| Bedeutung | Ablauf von A schließt den Ablauf von B immer ein | Ablauf von A kann optional um B erweitert werden |
| Anwendung | Hierachische Zerlegung | Abbildung von Sonderfällen |
| Abhängigkeiten | A muss B bei der Modellierung berücksichtigen | Unabhängige Modellierung möglich |

Party als Anwendungsfalldiagramm - Beispiel motiviert aus dem Beispiel von ^[Jeckle] Seite 240

Anwendungsfälle

- Darstellung der wichtigsten Systemfunktionen
- Austausch mit dem Anwender und dem Management auf der Basis logischer, handhabbarer Teile
- Dokumentation des Systemüberblicks und der Außenschnittstellen
- Indentifikation von Anwendungsfällen

Vermeiden Sie ...

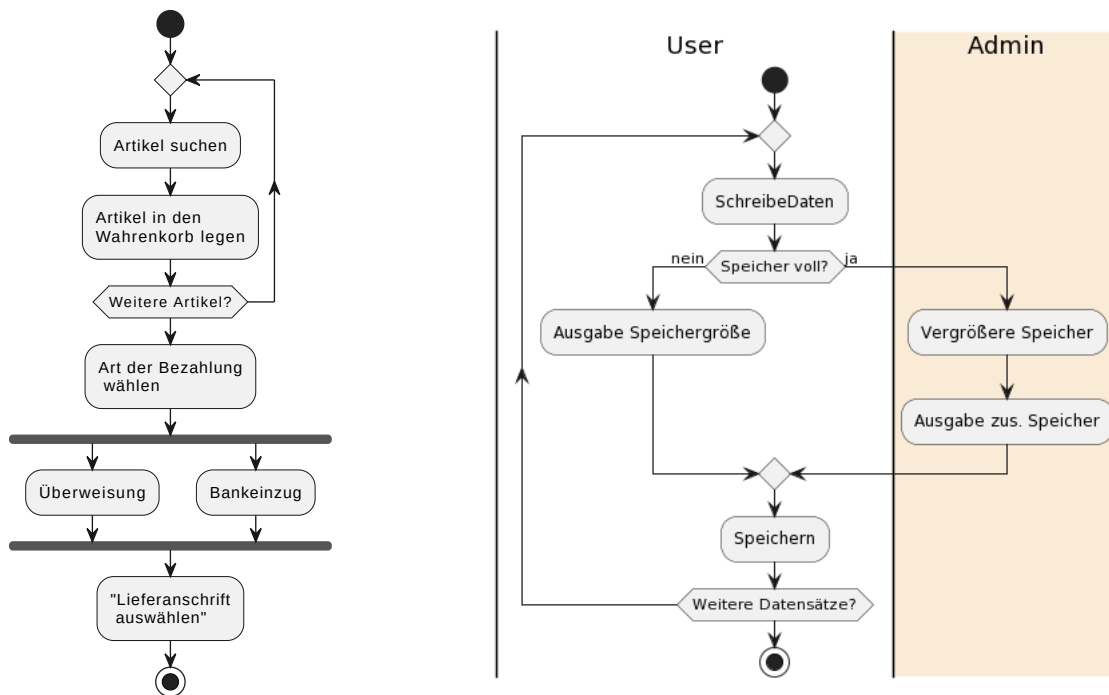
- ... eine zu detaillierte Beschreibung von Operationen und Funktionen
- ... nicht funktionale Anforderungen mit einem Use-Case abbilden zu wollen
- ... Use-Case Analysen aus Entwicklersicht durchzuführen
- ... zu viele Use-Cases in einem Diagramm abzubilden (max. 10)

[Jeckle] Mario Jeckle, Christine Rupp, Jürgen Hahn, Barbara Zengler, Stefan Queins, UML 2 glasklar, Hanser Verlag, 2004

Aktivitätsdiagramm

Aktivitätsdiagramme stellen die Vernetzung von elementaren Aktionen und deren Verbindungen mit Kontroll- und Datenflüssen grafisch dar.

Aktivitätsmodellierung in UML1



Aktivitätsdiagramme.plantUML

ActivityUser.plantUML

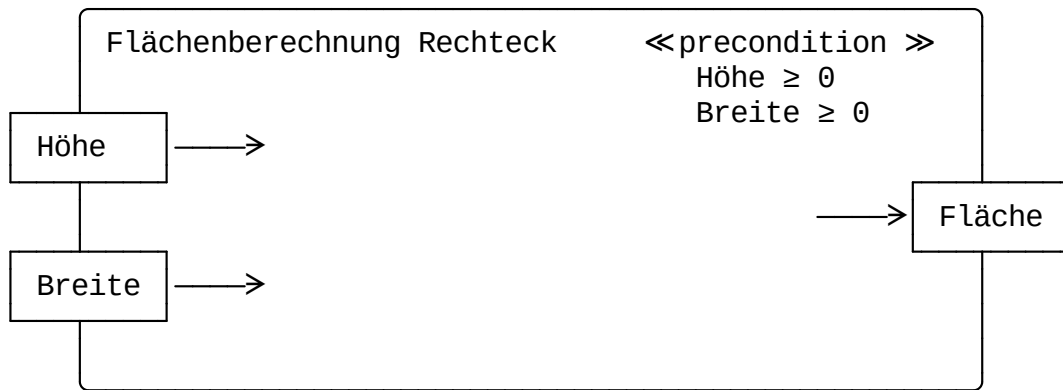
Bis UML 1.x waren Aktivitätsdiagramme eine Mischung aus Zustandsdiagramm, Petrinetz und Ereignisdiagramm, was zu theoretischen und praktischen Problemen führte.

Erweiterung des Konzeptes in UML2

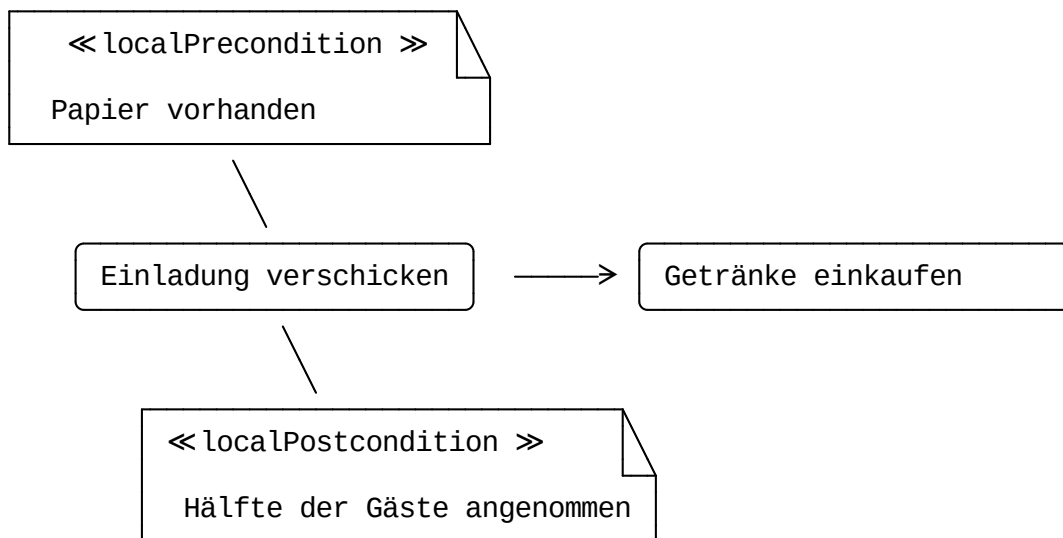
"Was früher Aktivitäten waren sind heute Aktionen."

UML2 strukturiert das Konzept der Aktivitätsmodellierung neu und führt als übergeordnete Gliederungsebene Aktivitäten ein, die Aktionen, Objektknoten sowie Kontrollelemente der Ablaufsteuerung und verbindende Kanten umfasst. Die Grundidee ist dabei, dass neben dem Kontrollfluss auch der Objektfluss modelliert wird.

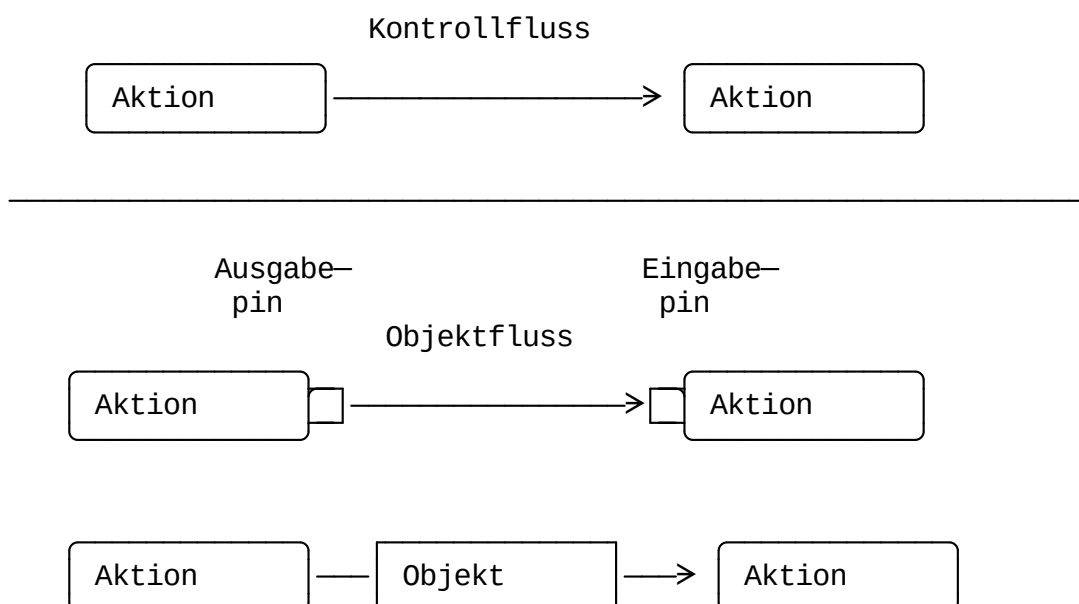
- Aktivitäten definieren Strukturierungselemente für Aktionen, die durch Ein- und Ausgangsparameter, Bedingungen, zugehörige Aktionen und Objekte sowie einen Bezeichner gekennzeichnet sind.



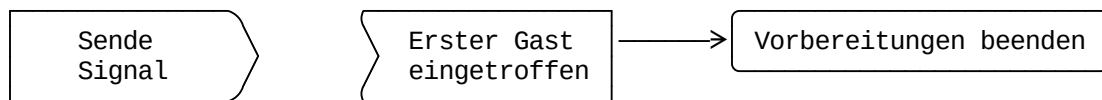
- Aktionen stehen für den Aufruf eines Verhaltens oder die Bearbeitung von Daten, die innerhalb einer Aktivität nicht weiter zerlegt wird.



- Objekte repräsentieren Daten und Werte, die innerhalb der Aktivität manipuliert werden. Damit ergibt sich ein nebeneinander von Kontroll- und Objektfluss.



- Signale und Ereignisse sind die Schnittstellen für das Auslösen einer Aktion



Beispiel

Beispiel eines Anwendungsfall Diagramms ^[WikiActivityDiagram]

Anwendungsfälle

- Verfeinerung von Anwendungsfällen (aus den Use Case Diagrammen)
- Darstellung von Abläufen mit fachlichen Ausführungsbedingungen
- Darstellung für Aktionen im Fehlerfall oder Ausnahmesituationen

[WikiActivityDiagram] Wikimedia, Autor Gubaer, UML2 Aktivitätsdiagramm, <https://commons.wikimedia.org/wiki/File:Uml-Activity-Beispiel2.svg>

Sequenzdiagramm

Sequenzdiagramme beschreiben den Austausch von Nachrichten zwischen Objekten mittels Lebenslinien.

Ein Sequenzdiagramm besteht aus einem Kopf- und einem Inhaltsbereich. Von jedem Kommunikationspartner geht eine Lebenslinie (gestrichelt) aus. Es sind zwei synchrone Operationsaufrufe, erkennbar an den Pfeilen mit ausgefüllter Pfeilspitze, dargestellt. Notationsvarianten für synchrone und asynchrone Nachrichten

Eine Nachricht wird in einem Sequenzdiagramm durch einen Pfeil dargestellt, wobei der Name der Nachricht über den Pfeil geschrieben wird. Nachrichten können:

- Operationsaufrufe einer Klasse sein
- Ergebnisse einer Operation
- Signale
- Interaktionen mit dem Nutzern
- das Setzen einer Variablen

Synchrone Nachrichten werden mit einer gefüllten Pfeilspitze, asynchrone Nachrichten mit einer offenen Pfeilspitze gezeichnet.

Die schmalen Rechtecke, die auf den Lebenslinien liegen, sind Aktivierungsbalken, die den Focus of Control anzeigen, also jenen Bereich, in dem ein Objekt über den Kontrollfluss verfügt, und aktiv an Interaktionen beteiligt ist.

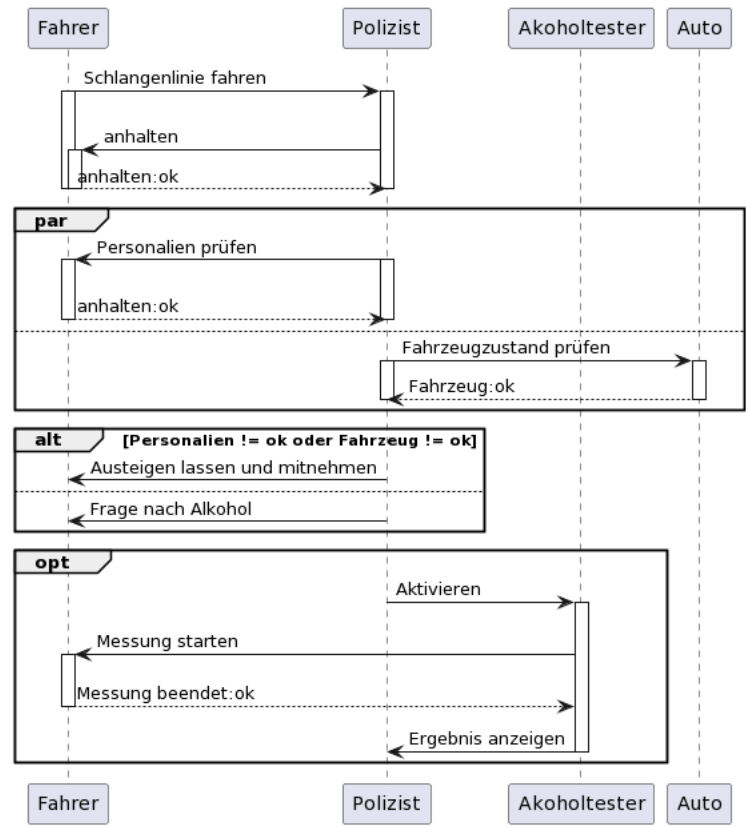
Beispiel

Sequenzdiagramm der Interaktion zwischen Emailclient und Server [\[WikiSequenceDiagram\]](#)

Bestandteile

| Name | Beschreibung |
|------------------------|---|
| Objekt | Dient zur Darstellung einer Klasse oder eines Objekts im Kopfbereich. |
| Nachrichtensequenzen | Modelliert den Informationsfluss zwischen den Objekten |
| Aktivitätsbalken | Repräsentiert die Zeit, die ein Objekt zum Abschließen einer Aufgabe benötigt. |
| Paket | Strukturiert das Sequenzdiagramm |
| Lebenslinien-Symbol | Stellt durch die Ausdehnung nach unten den Zeitverlauf dar. |
| Fragmente | Kapseln Sequenzen in „Wenn-dann“-Szenarien, optionalen Interaktionen, Schleifen, etc. |
| Alternativen-Symbol | Stellt eine Auswahl zwischen zwei oder mehr Nachrichtensequenzen (die sich in der Regel gegenseitig ausschließen) dar |
| Interaktionsreferenzen | Binden Submodelle und deren Ergebnisse ein deren |

Beispiel



Alkoholkontrolle.plantUML

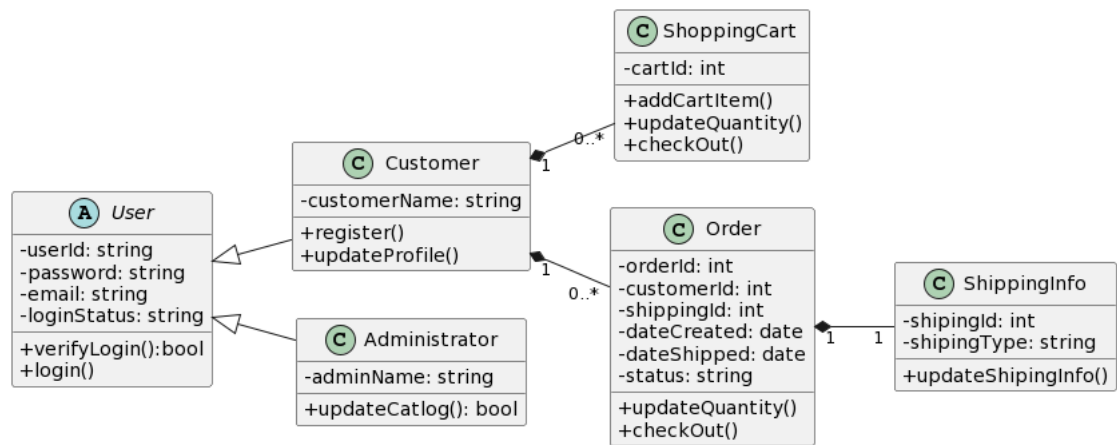
[\[WikiSequenceDiagram\]](#) Wikimedia, Autor Coupling/ossgraph.svg, UML of message sequence, <https://commons.wikimedia.org/wiki/File:CheckEmail.svg>

Klassendiagramme

Ein Klassendiagramm ist eine grafischen Darstellung (Modellierung) von Klassen, Schnittstellen sowie deren Beziehungen.

Beispiel

Nehmen wir an, sie planen die Software für ein Online-Handel System. Es soll sowohl verschieden Nutzertypen (*Customer* und *Administrator*) als auch die Objekt *ShoppingCart* und *Order* umfassen.



Klassen

Klassen werden durch Rechtecke dargestellt, die entweder nur den Namen der Klasse (fett gedruckt) tragen oder zusätzlich auch Attribute, Operationen und Eigenschaften spezifiziert haben. Oberhalb des Klassennamens können Schlüsselwörter in Guillemets und unterhalb des Klassennamens in geschweiften Klammern zusätzliche Eigenschaften (wie {abstrakt}) stehen.

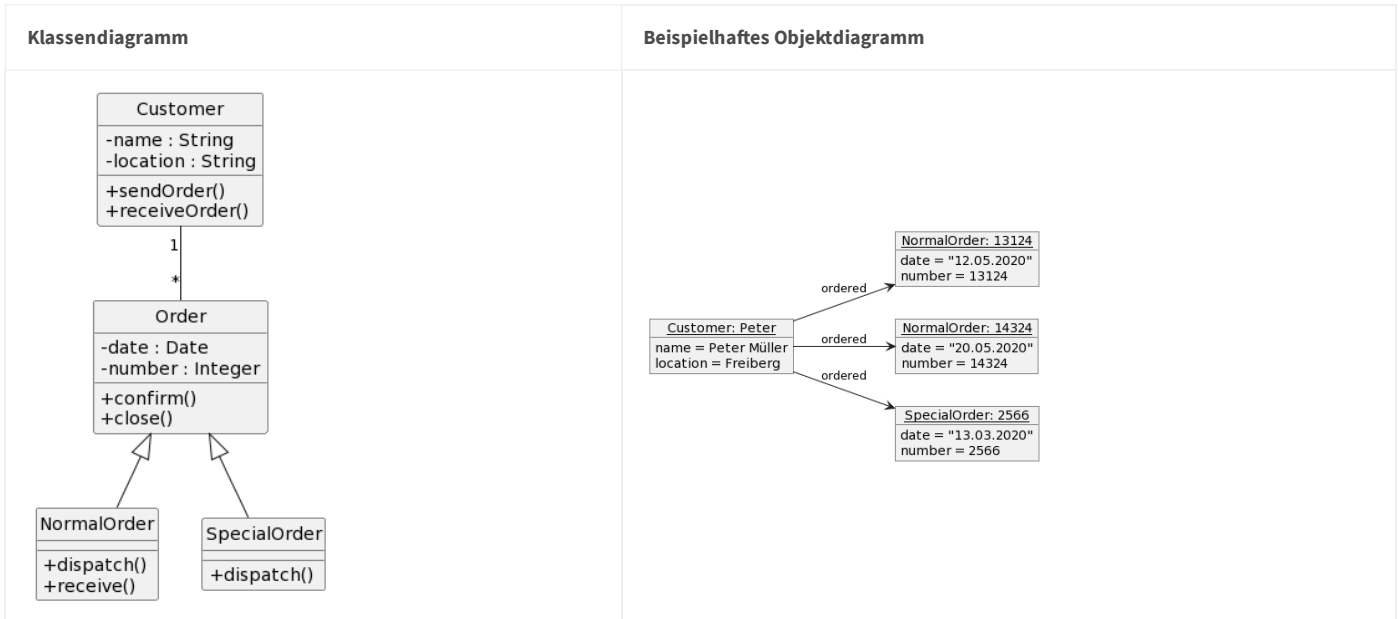
Elemente der Darstellung :

| Eigenschaften | Bedeutung |
|---------------|--|
| Attribute | beschreiben die Struktur der Objekte: Bestandteile und darin enthalten Daten |
| Operationen | Beschreiben das Verhalten der Objekte (Methoden) |
| Zusicherungen | Bedingungen, Voraussetzungen und Regeln, die die Objekte erfüllen müssen |
| Beziehungen | Beziehungen einer Klasse zu anderen Klassen |

Wenn die Klasse keine Eigenschaften oder Operationen besitzt, kann die unterste horizontale Linie entfallen.

UML Klassendiagramme und deren Attribute - adaptiert aus [\[WikiUMLClass\]](#)

Objekte vs. Klassen



Darstellung motiviert nach *What is Object Diagram?*, <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/>, Autor unbekannt

Merke: Vermeiden Sie bei der Benennung von Klassen, Attributen, Operationen usw. sprachspezifische Zeichen

Modellierung in UML

Welcome to PlantUML!

You can start with a simple UML Diagram like:

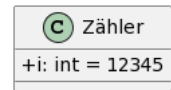
Bob->Alice: Hello

Or

```
class Example
```

You will find more information about PlantUML syntax on <https://plantuml.com>

(Details by typing `license` keyword)



Ausführbarer Code in Python 2

```
KlasseMitUmlaut.py

1 class Zähler:
2     """A simple example class"""
3     i = 12345
4
5 A = Zähler()
6 print(A.i)
```

timeout: failed to run command 'python': No such file or directory

Ausführbarer Code in C++ 20

KlasseMitUmlaut.cpp

```
1 #include <iostream>
2
3 class Zähler{
4     public:
5     int i = 12345;
6 };
7
8 int main()
9 {
10     Zähler A = Zähler();
11     std::cout << A.i;
12     return 0;
13 }
```

12345

Sichtbarkeitsattribute

| Zugriffsmodifizierer | Innerhalb eines Assemblys Vererbung | Innerhalb eines Assemblys Instanziierung | Außerhalb eines Assemblys Vererbung | Außerhalb eines Assemblys Instanziierung |
|---------------------------------|-------------------------------------|--|-------------------------------------|--|
| <code>public</code> | ja | ja | ja | ja |
| <code>private</code> | nein | nein | nein | nein |
| <code>protected</code> | ja | nein | ja | nein |
| <code>internal</code> | ja | ja | nein | nein |
| <code>internal protected</code> | ja | ja | ja | nein |

public, private

Die Sichtbarkeitsattribute `public` und `private` sind unabhängig vom Vererbungs-, Instanzierungs- oder Paketstatus einer Klasse. Im Beispiel kann der `TrafficOperator` nicht auf die Geschwindigkeiten der Instanzen von `Car` zurückgreifen.

Private member [PublicPrivate.plantUML](#)

protected

Die abgeleitete Klassen `Bus` und `PassagerCar` erben von `Car` und übernehmen damit deren Methoden. Die Zahl der Sitze wird beispielsweise mit ihrem Initialisierungswert von 5 auf 40 gesetzt. Zudem muss die Methode `StopAtStation` auch auf die Geschwindigkeit zurückgreifen können.

Protected Member [protected.plantUML](#)

internal

Ein Member vom Typ `protected internal` einer Basisklasse kann von jedem Typ innerhalb seiner enthaltenden Assembly aus zugegriffen werden.

Internal Member [internal.plantUML](#)

Merke: Der UML Standard kennt nur `+ public`, `- private`, `# protected` und `~ internal`. Das C# spezifische `internal protected` ist als weitere Differenzierungsmöglichkeit nicht vorgesehen.

Attribute

Merke: In der C# Welt sprechen wir bei Attributen von Membervariablen und Feldern.

Im einfachsten Fall wird ein Attribut durch einen Namen repräsentiert, der in der Klasse eindeutig sein muss - Die Klasse bildet damit den Namensraum der enthaltenen Attribute.

Entsprechend der Spezifikation sind folgende Elemente eines Attributes definierbar:

```
[Sichtbarkeit] [/] Name [: Typ] [ Multiplizität ] [= Vorgabewert]
[{Eigenschaftswert}]
```

- *Sichtbarkeit* ... vgl. vorheriger Absatz Das "/" bedeutet, dass es sich um ein abgeleitetes Attribut handelt, dessen Daten von anderen Attributen abhängt
- *Name* ... des Attributes, Leer und Sonderzeichen sollten weggelassen werden, um zu vermeiden, dass Sie bei der Implementierung Probleme generieren.
- *Typ* ... UML verwendet zwar einige vordefinierte Typen (Integer, String, Boolean) beinhaltet aber keine Einschränkungen zu deren Wertebereich!
- *Multiplizität* ... die Zahlenwerte in der rechteckigen Klammer legen eine Ober- und Untergrenze der Anzahl (Kardinalitäten) von Instanzen eines Datentyps fest.

| Beispiel | Bedeutung |
|-------------------|--|
| <code>0..1</code> | optionales Attribut, das aber höchstens in einer Instanz zugeordnet wird |
| <code>1..1</code> | zwingendes Attribut |
| <code>0..n</code> | optionales Attribute mit beliebiger Anzahl |
| <code>1..*</code> | zwingend mit beliebiger Anzahl größer Null |
| <code>n..m</code> | allgemein beschränkte Anzahl größer 0 |

- *Vorgabewerte* ... definieren die automatische Festlegung des Attributes auf einen bestimmten Wert
- *Eigenschaftswerte* ... bestimmen die besondere Charakteristik des Attributes

| Eigenschaft | Bedeutung |
|------------------------|---|
| <code>readOnly</code> | unveränderlicher Wert |
| <code>subsets</code> | definiert die zugelassen Belegung als Untermenge eines anderen Attributs |
| <code>redefines</code> | überschreiben eines ererbten Attributes |
| <code>ordered</code> | Inhaltes eines Attributes treten in geordneter Reihenfolge ohne Dublikate auf |
| <code>bag</code> | Attribute dürfen ungeordnet und mit Dublikaten versehen enthalten sein |
| <code>sequence</code> | legt fest, dass der Inhalt sortiert, aber ohne Dublikate ist |
| <code>composite</code> | |

Daraus ergeben sich UML-korrekte Darstellungen

| Attributdeklaration | Korrekt | Bemerkung |
|---|---------|--|
| <code>public zähler:int</code> | ja | Umlaute sind nicht verboten |
| <code>/ alter</code> | ja | Datentypen müssen nicht zwingend angegeben werden |
| <code>privat adressen: String [1..*]</code> | ja | Menge der Zeichenketten |
| <code>protected bruder Person</code> | ja | Datentyp kann neben den Basistypen jede andere Klasse oder eine Schnittstelle sein |
| String | nein | Name des Attributes fehlt |
| privat, public name: String | nein | Fehler wegen mehrfachen Zugriffsattributen |

Korrespondierendes UML-Klassendiagramm [AttributeExample.plantUML](#)

AttributeExample

```
using System;

namespace Rextester
{
    class Example
    {
        int attribute1;
        public int attribute2;
        public static double pi = 3.14;
        private bool attribute3;
        protected short attribute4;
        internal const string attribute5 = "Test";
        B attribute6;
        System.Collections.Specialized.StringCollection attribute7;
        private int wert;
        Object attribute8{
            get{return wert * 10;}
        }
    }
}
```

Operationen

Merke: In der C# Welt sprechen wir bei Operationen von Methoden.

Operationen werden durch mindestens ihren Namen sowie wahlfrei weitere Angaben definiert. Folgende Aspekte können entsprechend der UML Spezifikation beschrieben werden:

| | |
|---|--|
| [[Sichtbarkeit]] Name (Parameterliste) [: Rückgabotyp] [{Eigenschaftswert}] | |
|---|--|

Dabei ist die Parameterliste durch folgende Elemente definiert:

| | |
|--|--|
| [[Übergeberichtung]] Name [: Typ] [Multiplizität] [= Vorgabewert] [{Eigenschaftswert}] | |
|--|--|

- *Sichtbarkeit* ... analog Attribute
- *Name* ... analog Attribute
- *Parameterliste* ... Aufzählung der durch die aufrufende Methode übergebenden Parameter, die im folgenden nicht benannten Elemente folgend den Vorgaben, die bereits für die Attribute erfasst wurden:
 - *Übergeberichtung* ... Spezifiziert die Form des Zugriffs (**in** = nur lesender Zugriff, **out** = nur schreibend (reiner Rückgabewert), **inout** = lesender und schreibender Zugriff)
 - *Vorgabewert* ... default-Wert einer Übergabevariablen
- *Rückgabotyp* ... Datentyp oder Klasse, der nach der Operationsausführung zurückgegeben wird.
- *Eigenschaftswert* ... Angaben zu besonderen Charakteristika der Operation

Korrespondierendes UML-Klassendiagramm [OperationsExample.plantUML](#)

OperationsExample

```
using System;

class Example
{
    public static void operation1(){
        // Implementierung
    }

    private int operation2 (int param1 = 5)
    {
        // Implementierung
        return value;
    }

    protected void operation3 (ref C param3)
    {
        // Implementierung
        param3 = ...
    }

    internal B operation4 (out StringCollection param3)
    {
        // Implementierung
        return value;
    }
}
```

[WikiUMLClass] https://de.wikipedia.org/wiki/Klassendiagramm#/media/Datei:UmlCd_Klasse-3.svg, Autor Gubaer, CC BY-SA 3.0

Schnittstellen

Eine Schnittstelle wird ähnlich wie eine Klasse mit einem Rechteck dargestellt, zur Unterscheidung aber mit dem Schlüsselwort `interface` gekennzeichnet.

Darstellung von Schnittstellen in UML-Klassendiagrammen [PublicPrivate.plantUML](#)

Eine alternative Darstellung erfolgt in der LolliPop Notation, die die grafische Darstellung etwas entkoppelt.

Steigerung der Lesbarkeit durch die Verwendung von Lollipop Symbolen [lollipop.plantUML](#)

OperationsExample

```
using System;

interface Sortierliste{
    void einfuegen (Eintrag e);
    void loeschen (Eintrag e);
}

class Datenbank : SortierteListe
{
    void einfuegen (Eintrag e) {//Implementierung};
    void loeschen (Eintrag e) {//Implementierung};
}
```

Beziehungen

Die Möglichkeiten der Verknüpfung zwischen Klassen und Interfaces lassen sich wie folgt gliedern:

| Beziehung | Darstellung | Bedeutung |
|---------------------------------------|-------------|--|
| Generalisierung | | gerichtete Beziehung zwischen einer generelleren und einer spezielleren Klasse (Vererbung) |
| Assoziationen (ohne Anpassung) | | beschreiben die Verknüpfung allgemein |
| Assoziation (Komposition/Aggregation) | | Bildet Beziehungen von einem Ganzen und seinen Teilen ab |

Verwendung von UML Tools

Verwendung von Klassendiagrammen

- ... unter Umbrello (UML Diagramm Generierung / Code Generierung)
- ... unter Microsoft Studio ([Link](#))

- ... unter Visual Studio Code mit PlantUML oder UMLet (siehe Codebeispiele zu dieser Vorlesung)
- ... unter Visual Studio Code mit [PlantUmlClassDiagramGenerator](#)

Aufgaben

- ☐ Experimentieren Sie mit [Umbrello](#)
- ☐ Experimentieren Sie mit der automatischen Extraktion von UML Diagrammen für Ihre Computer-Simulation aus den Übungen
- ☐ Evaluieren Sie das Add-On "Class Designer" für die Visual Studio Umgebung