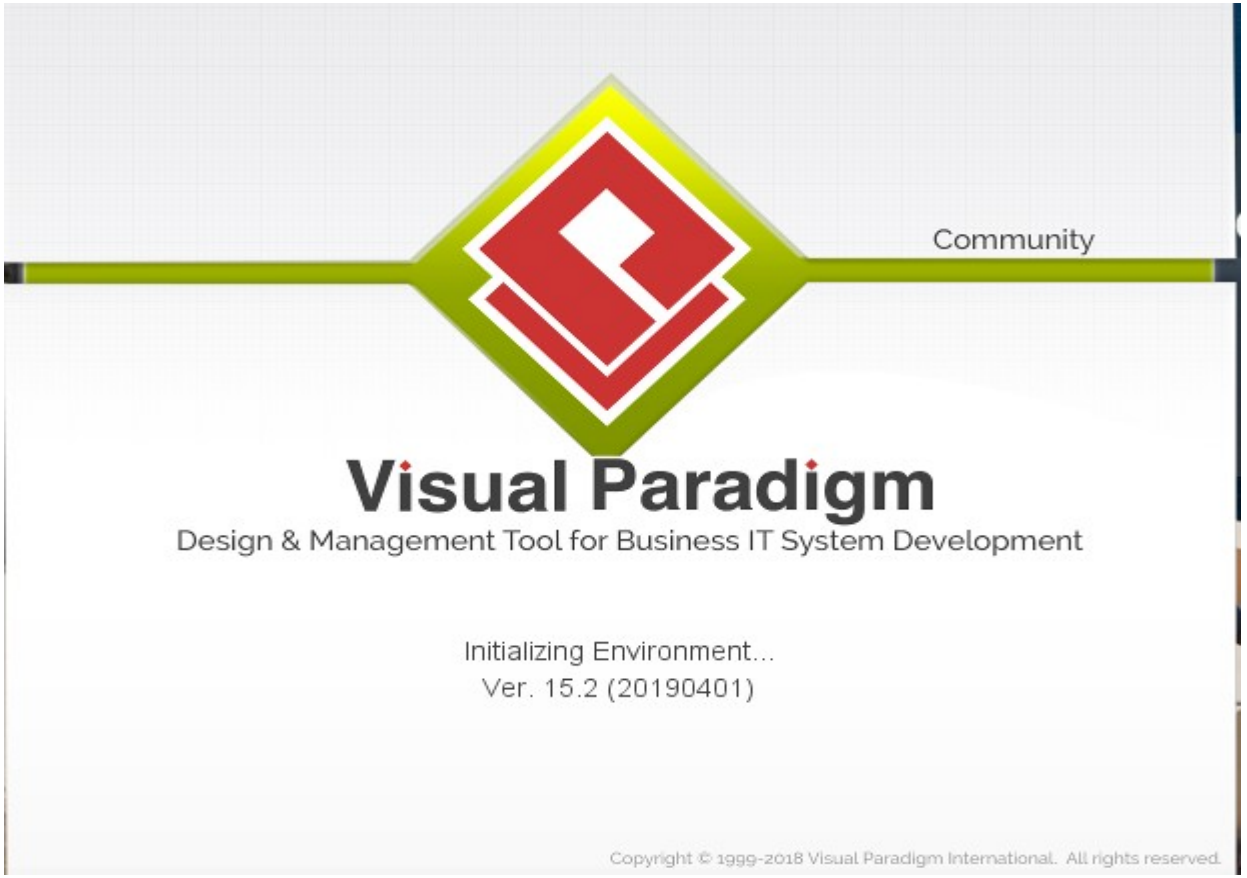
UML Einführung



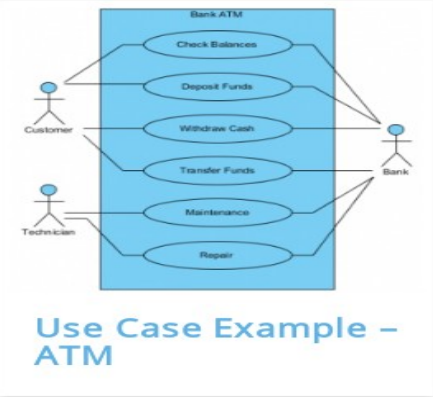
Inhalte

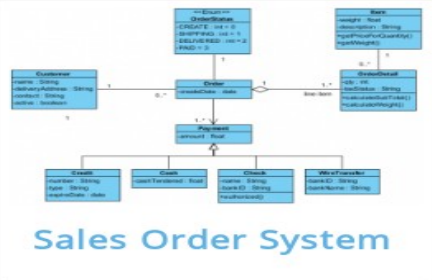
1. Motivation/Was ist UML?
   (siehe 1. Vorlesung)
2. Einleitung und Überblick über UML
3. UML im Detail: Klassendiagramme
4. UML-Werkzeuge: Visual Paradigm

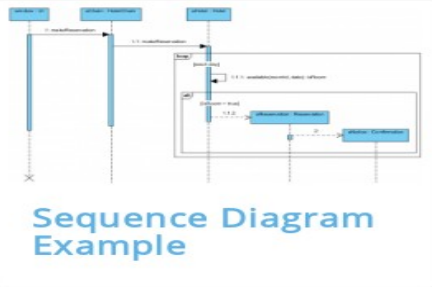
https://www.visual-paradigm.com/download/community.jsp

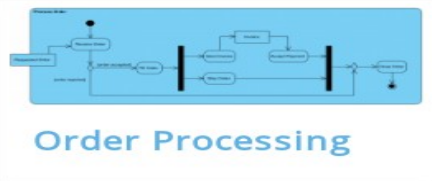
Visual Paradigm Community : Examples

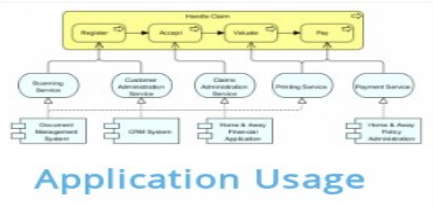
Featured Examples and Templates

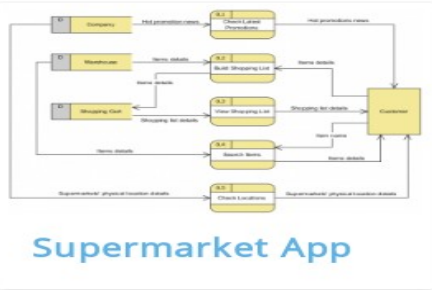












https://circle.visual-paradigm.com/

1.1 Motivation

Für Entwicklung (komplexer) Softwaresysteme

ist Modellierung unabdingbar

- ***Modellierung*** **= Abstraktion der Realität mit Konzentration auf die wesentlichen Aspekte**

Beispieldesign mit Hilfe von Visual Paradigm:

Softwareapplikation einer Theaterkasse

Motivation (Forts.)

Modellierung hilft

die gewünschte Struktur und das Verhalten eines Systems zu beschreiben und zu diskutieren

das System besser zu verstehen, indem man sich zu einem Zeitpunkt nur auf einen Ausschnitt bzw. einen einzelnen Aspekt konzentriert

frühzeitig Probleme des Modells zu erkennen und zu beheben

Motivation (Forts.)

- **Wünschenswert für die Modellierung von Softwaresystemen ist**

das Systems zu *visualisieren*, wie es ist oder wie es sein sollte

sowohl die Struktur als auch das Verhaltens des Systems *spezifizieren* zu können

eine Vorlage aufzubauen, aus der leicht ein System zu *konstruieren* ist

Möglichkeiten zu haben, Entscheidungen zu *dokumentieren*

Motivation (Forts.)

⇒ **Zur Realisierung o.a. Aspekte der Modellierung**

stellt UML eine *einheitliche* und

***umfassende* Notation zur Verfügung**

1.2 Was ist UML?

UML (Unified Modeling Language) ist eine graphische Sprache zur

– Visualisierung

– Spezifikation

– Konstruktion und

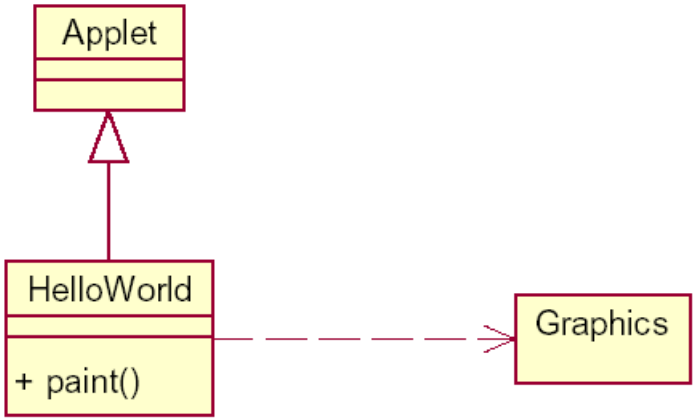
– Dokumentation

von Softwaresystemen

Visualisierung

Diagramme und graphische Notationen zur Darstellung von Softwaresystemen

Beispiel (Ein einfaches Klassendiagramm):



Spezifikation

„Sprachelemente“ sind mit eindeutiger Semantik versehen

Spezifikation in verschiedenen Detaillierungsgraden möglich

(sehr abstrakt bis sehr implementierungsnah)

Konstruktion

• **Abbildung von Modellen auf verschiedene Programmiersprachen möglich**

⇒ ***Forward Engineering*** **= Codegenerierung aus UML-Modellen (nur in Visual Paradigm Standard Edition)**

• **Umkehrung ebenfalls möglich**

⇒ ***Reverse Engineering*** **= Konstruktion von UML-Modellen aus Quell-Code (nur in Visual Paradigm Standard Edition)**

Konstruktion (Forts.)

- **Zusammen: *Round-trip Engineering***
- **Philosophie:**

– Manche Aspekte eines Softwaresystems lassen sich besser *graphisch* spezifizieren, andere Aspekte besser *textuell*

- **Benutzung von CASE-Werkzeugen für die Konstruktion**

(CASE = Computer Aided Software Engineering)

Dokumentation

Konstrukte zur Verwaltung und Erstellung von Dokumentation, die im Rahmen eines Softwareentwicklungsprojektes anfällt

- **Anforderungen an die Software**
- **Architektur und Design**
- **Quell-Code**
- **Projektpläne, Ablaufpläne**
- **Testfälle**
- **Prototypen und verschiedene Versionen**

Was ist UML? (Forts.)

UML ist eine standardisierte Modellierungssprache (bald ISO-Standard);

UML schreibt jedoch *keinen* bestimmten *Softwareentwicklungsprozess* vor!

2.1 Strukturierung der UML-Konstrukte

UML-Konstrukte lassen sich nach verschiedenen *Sichten* einteilen

Sicht **(View) = Betrachtung des Softwaresystems aus einer bestimmten Perspektive;**

Fokussierung auf einen speziellen Sachverhalt

Sichten

Verschiedene Sichten sinnvoll, da

– bei Erstellung eines Softwaresystems verschiedene Personengruppen involviert sind

– verschiedene Sichtweisen auf denselben Sachverhalt hilfreich für das Verständnis sein können

Personengruppen im Softwareentwicklungsprozess

Bearbeitung und Nutzung des Systems durch verschiedene Personengruppen

- **Endbenutzer, Auftraggeber**
- **Projektleiter**
- **Analytiker und Entwickler**
- **System-Administrator, -Integrator**
- **Tester**
- **Personen, die Dokumentation schreiben**

Sichten - grobe Unterteilung

UML bietet Vielzahl von Konzepten und Konstrukten zur Realisierung verschiedener Sichten an

Grobe Unterteilung der Sichten in

– Strukturelle Klassifikation

– Dynamisches Verhalten

– Modell Management

Sichten - grobe Unterteilung (Forts.)

Strukturelle Klassifikation **beschreibt die Objekte eines Systems und deren Beziehung zu anderen Objekten**

Bereich ***Dynamisches Verhalten*** **umfasst Sichten, die Änderungen des Systems in Abhängigkeit der Zeit beschreiben**

Modell Management **beschäftigt sich mit der Organisation des Modells selbst**

Sichten - grobe Unterteilung (Forts.)

UML besitzt *Erweiterungsmechanismen*, die auf alle Elemente angewendet werden können

2.2 Überblick über UML-Konstrukte

Überblick über die verschiedenen UMLKonstrukte und Sichten anhand eines Beispiels

Beispiel: Die Verwaltung eines Theaters soll auf EDV umgestellt werden (stark vereinfacht)

Statische Sicht (Static View)

Modellierung der Konzepte des Anwendungsbereichs sowie interner Konzepte

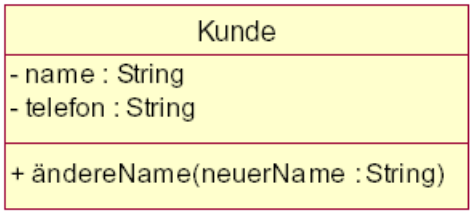
***kein* zeitabhängiges Verhalten**

- **Hauptbestandteile: *Klassen* (Classes) und ihre *Beziehungen* (Relationships) in Form von *Klassendiagrammen* (Class diagrams)**

Statische Sicht (Forts.)

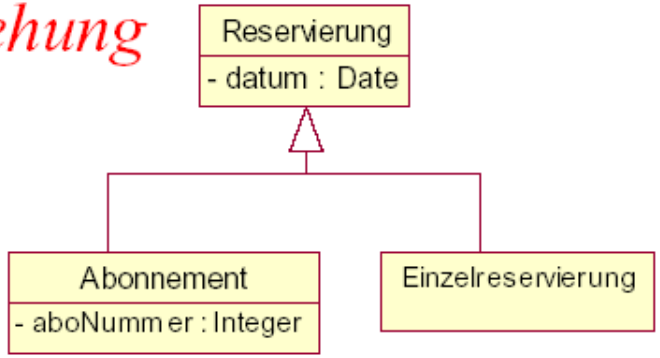
– ***Klassen*** = **Rechtecke** **mit getrennten Bereichen für *Attribute* (Attributes) und *Operationen* (Operations)**

– ***Beziehungen*** = **Linien** **zwischen betroffenen Klassen; Art der Beziehung unterscheidbar durch verschiedene Linienarten, -enden und -beschriftungen**

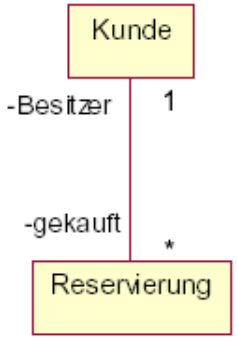


Graphische Darstellung (Forts.):

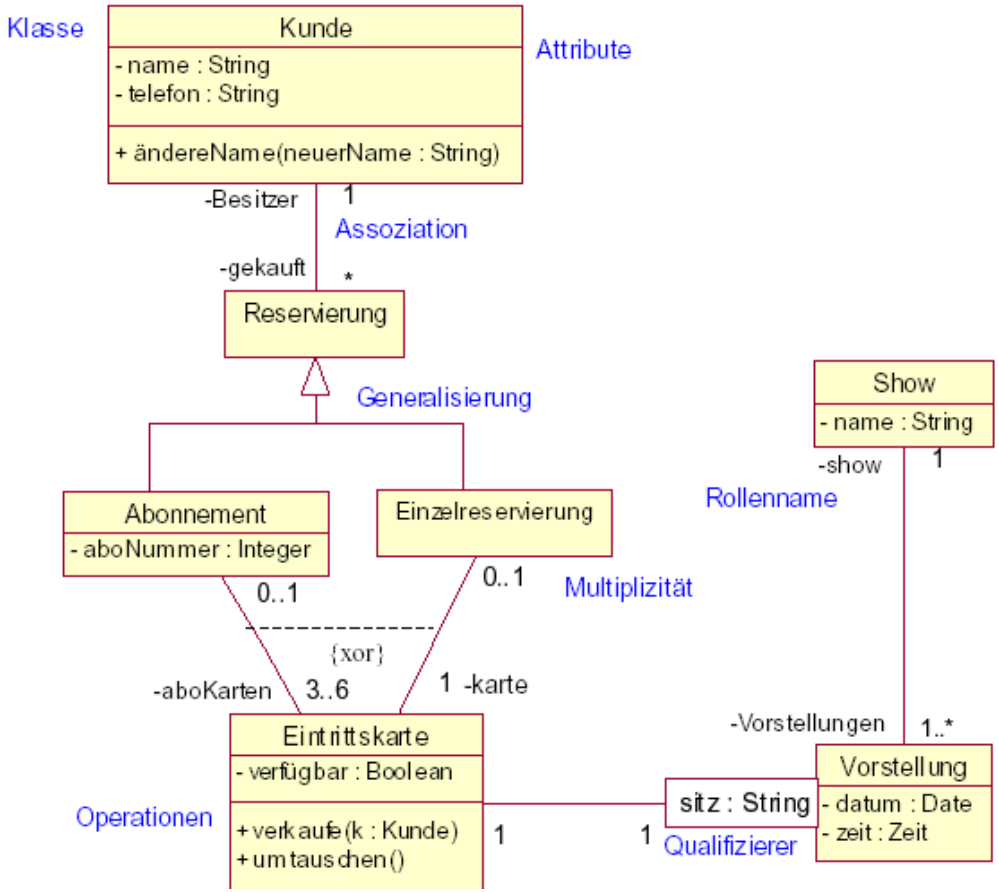
– *Generalisierungs-Beziehung*



– *Assoziations-Beziehung*



Klassendiagramm *Theaterkasse*



Anwendungsfall-Sicht (Use Case View)

Modellierung der Funktionalität des Systems, wie sie von aussenstehenden Benutzern, sog. *Akteuren* (Actors), wahrgenommen wird

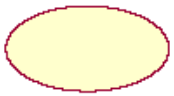
***Anwendungsfall* (Use Case) beschreibt typische Interaktion zwischen Benutzer und System (*WAS* macht das System, aber *nicht WIE*)**

Als Akteure können auch andere Systeme auftreten

Anwendungsfall-Sicht (Forts.)

Graphische Darstellung:

- *Anwendungsfälle* = Ellipsen



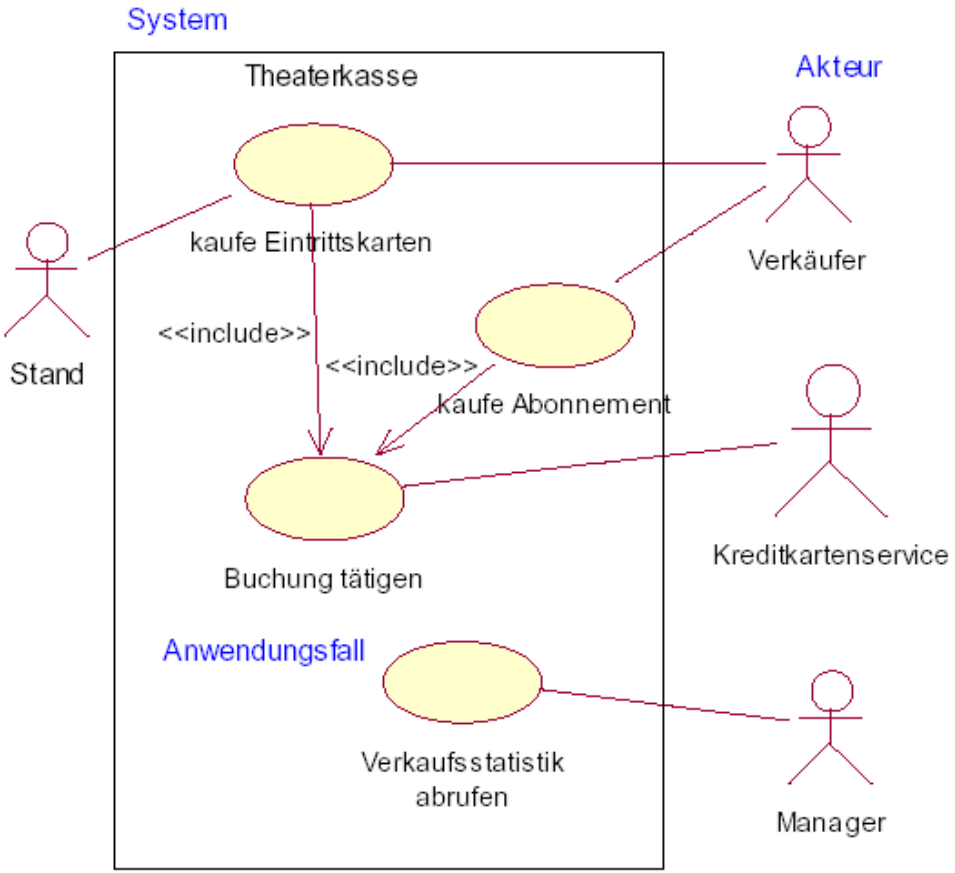
kaufe Eintrittskarten

- *Akteure* = Strichmännchen

Verkäufer

- *Systemgrenzen* werden durch Rechtecke visualisiert

Anwendungsfalldiagramm
Theaterkasse



Interaktions-Sicht (Interaction View)

Modellierung von Verhalten eines Systems
durch Darstellung des Nachrichtenaustauschs
zwischen verschiedenen Objekten
Zwei verschiedene Diagramme (Interaktionsdiagramme)
für die Interaktions-Sicht:

- ***Sequenzdiagramm*** **(Sequence diagram)**
- ***Kollaborationsdiagramm*** **(Collaborationdiagram)**

Sequenzdiagramm

Zeigt den Nachrichtenaustausch zwischen verschiedenen Objekten unter Betonung der zeitlichen Abfolge

Sequenzdiagramm (Forts.)

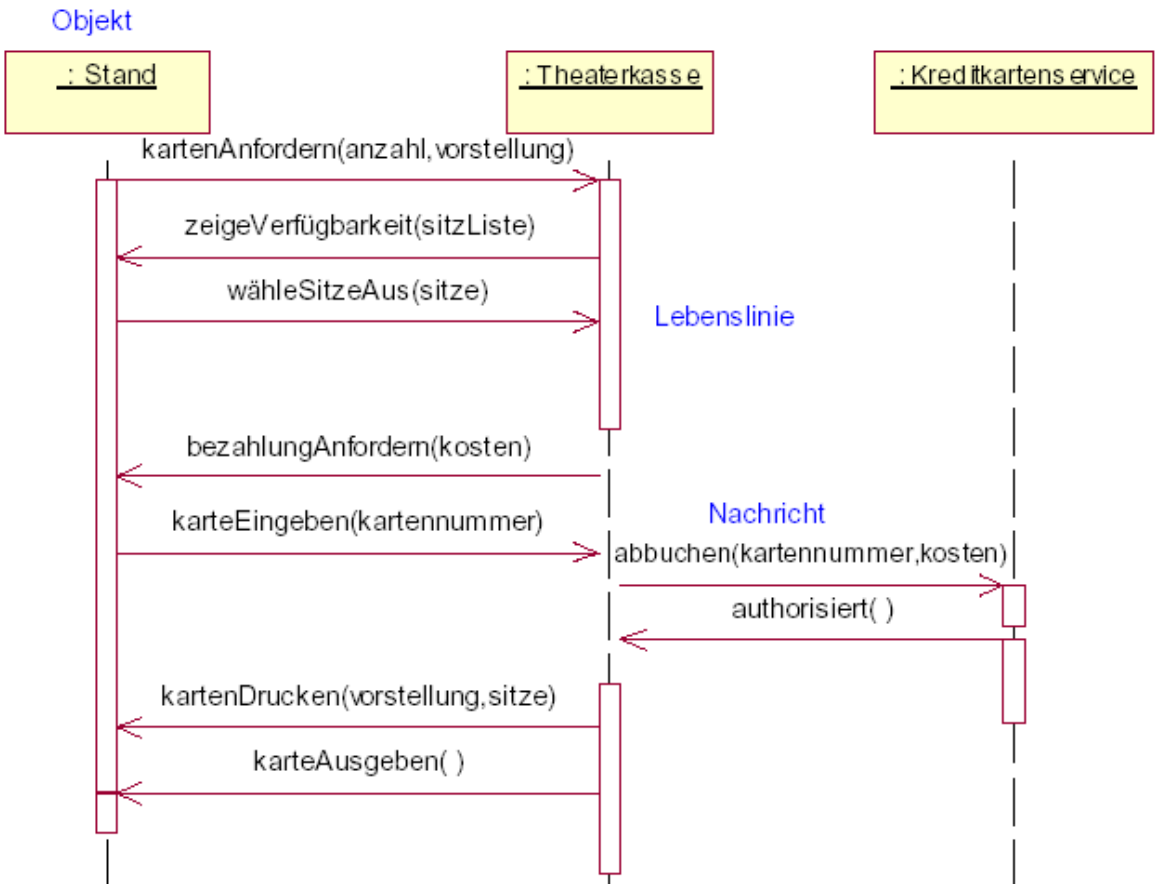
Graphische Darstellung:

– beteiligte Objekte werden horizontal angeordnet

– jedes Objekt hat eine *Lebenslinie* (Lifeline), die vertikal aufgetragen wird

– *Nachrichten* (Messages) = Pfeile zwischen den Lebenslinien der beteiligten Objekte

Sequenzdiagramm
kaufe Eintrittskarten



Kollaborationsdiagramm

Zeigt den Nachrichtenaustausch zwischen verschiedenen Objekten unter Betonung der Beziehung zwischen den Objekten

Kollaborationsdiagramm (Forts.)

Graphische Darstellung:

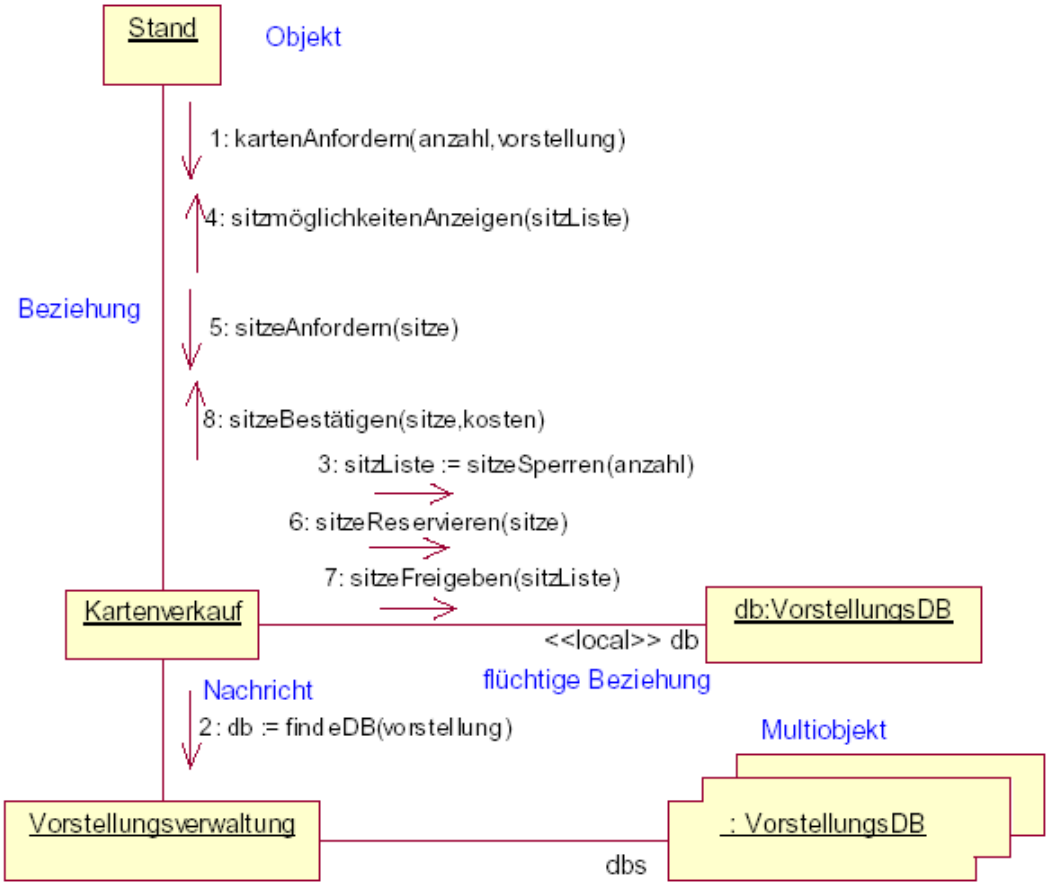
– Anordnung der beteiligten Objekte gemäß ihrer Beziehung

– Darstellung der Beziehungen durch Verbindungslinien

– *Nachrichten* = Pfeile entlang der Beziehungslinien

– zeitliche Abfolge der Nachrichten wird durch Numerierung festgelegt

Kollaborationsdiagramm
reserviere Eintrittskarten

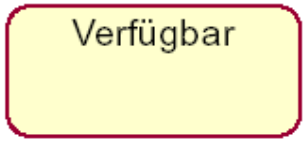


Zustandsmaschinen-Sicht (State Machine View)

- **Modellierung des Verhaltens eines einzelnen Objektes oder Systems:**
  - ***Zustand*** **(State) = Objekt erfüllt gewisse Bedingungen**
  - ***Zustandsübergang*** **(State Transition) = Übergang von einem Zustand in einen anderen Zustand, welcher durch ein Ereignis ausgelöst wird**
  - ***Ereignis*** **(Event) = Geschehen, das zu einem Zustandsübergang führt**

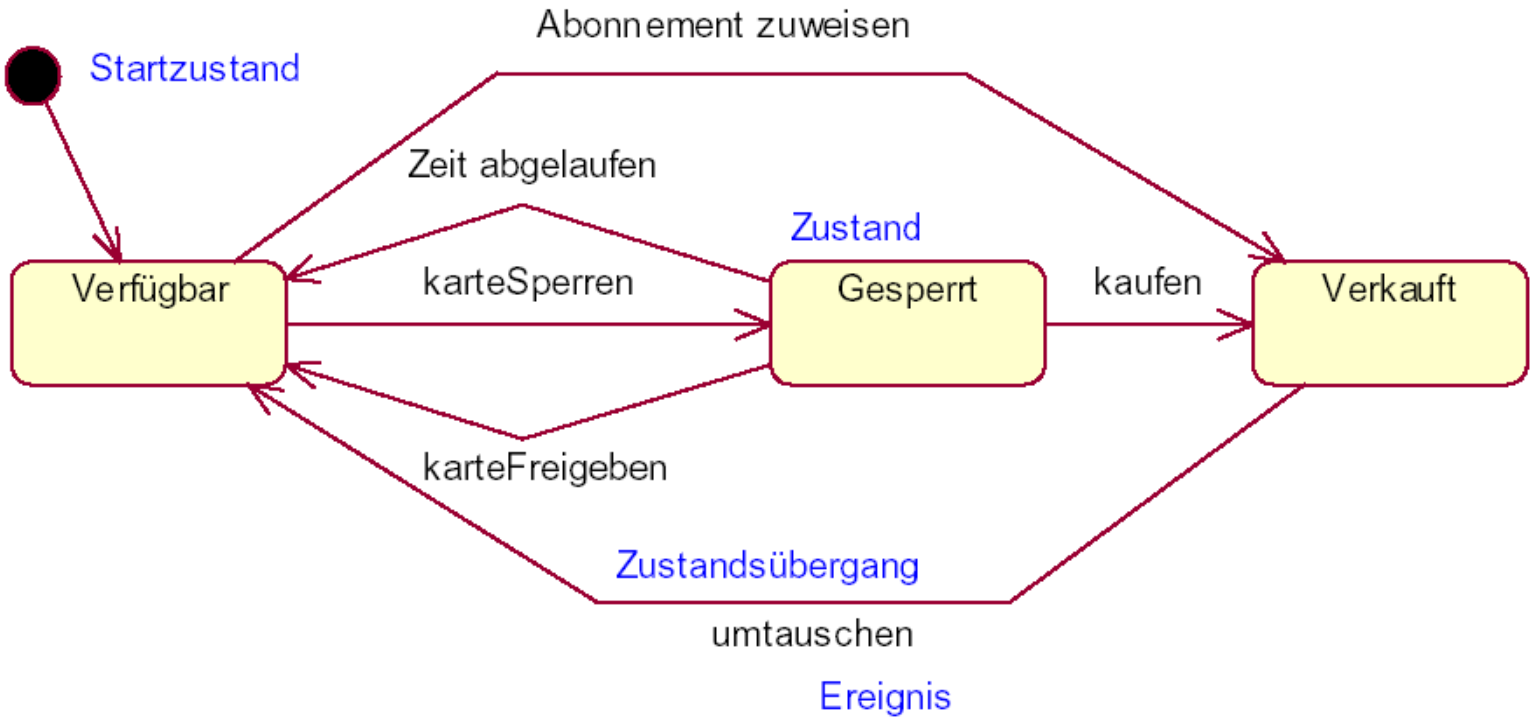
Graphische Darstellung:

- *Zustände* = Rechtecke mit abgerundeten Ecken



- *Zustandsübergänge* = Pfeile zwischen betroffenen Zuständen
- *Startzustand* = schwarzer ausgefüllter Kreis
- *Endzustand* = in Kreis eingeschlossener schwarzer ausgefüllter Kreis

Zustandsübergangsdiagramm
Eintrittskarte



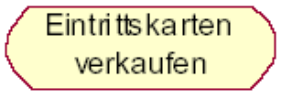
Aktivitäts-Sicht (Activity View)

- **Modellierung der Ablaufmöglichkeiten eines Systems durch Angabe der einzelnen Aktivitäten**
- **Aktivitätsdiagramm ist spezielle Form eines Zustandsübergangsdiagrammes**
- ***Aktivität* (Activity) ist Zustand mit interner Aktion und einem oder mehreren ausgehenden Zustandsübergängen, die automatisch der Beendigung der internen Aktion folgen**

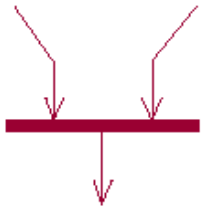
Aktivitäts-Sicht (Forts.)

Graphische Darstellung:

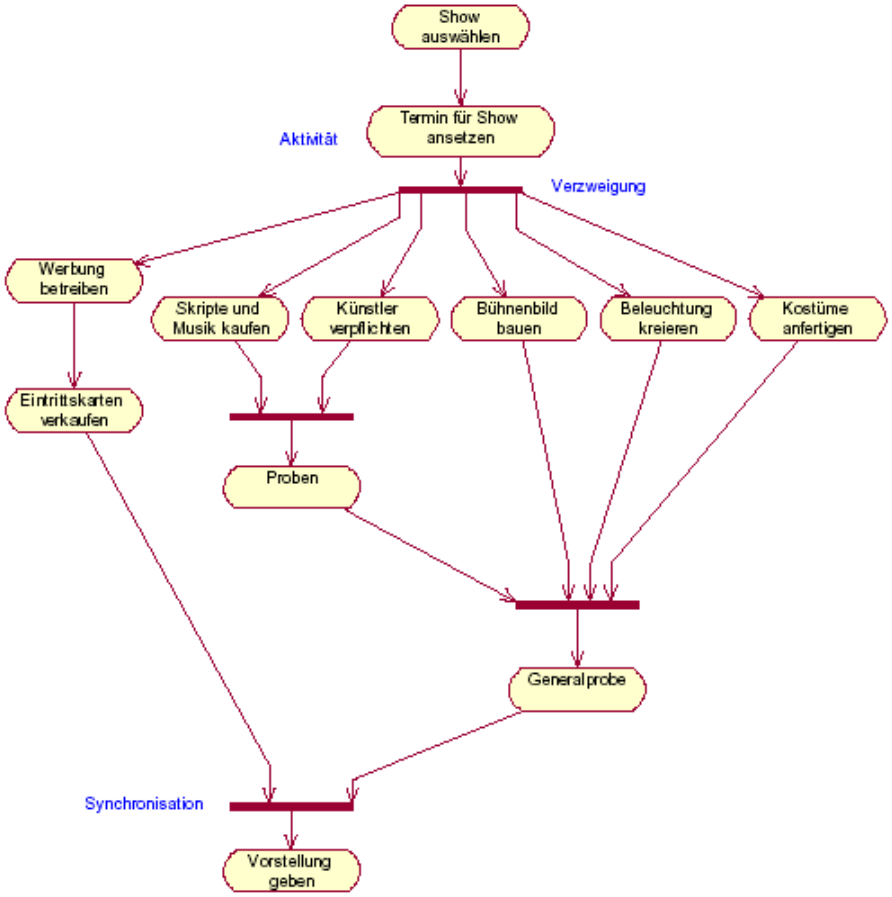
- *Aktivitäten* = Rechtecke, deren linke und rechte Seite Kreisbögen sind



- *Übergänge* = Pfeile zwischen den betroffenen Aktivitäten
- *Verzweigung* (Fork), *Synchronisation* (Join) = dickere Balken, bei denen sich Pfeile verzweigen bzw. von denen Pfeile abgehen



Aktivitätsdiagramm
Planung/Ausführung einer Show



Physische Sichten (Physical Views)

Bisherige Sichten modellierten *logische Struktur* der Applikation

Physische Sichten modellieren

– *Implementationsstruktur* der Applikation (Implementations-Sicht)

– *Verteilung der Komponenten* auf Rechnerknoten (Verteilungs-Sicht)

Implementations-Sicht (Implementation View)

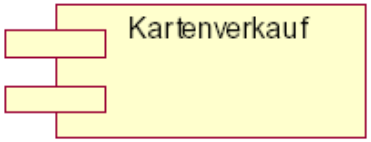
Zeigt die Komponenten (sowie deren Schnittstellen) eines Systems und deren Abhängigkeiten untereinander

Komponente **(Component) = Software Einheit, aus der die Applikation zusammengebaut wird (z.B. Quellcode, Executable, Library, ...)**

Schnittstelle **(Interface) = extern sichtbares Verhalten einer Komponente**

Graphische Darstellung:

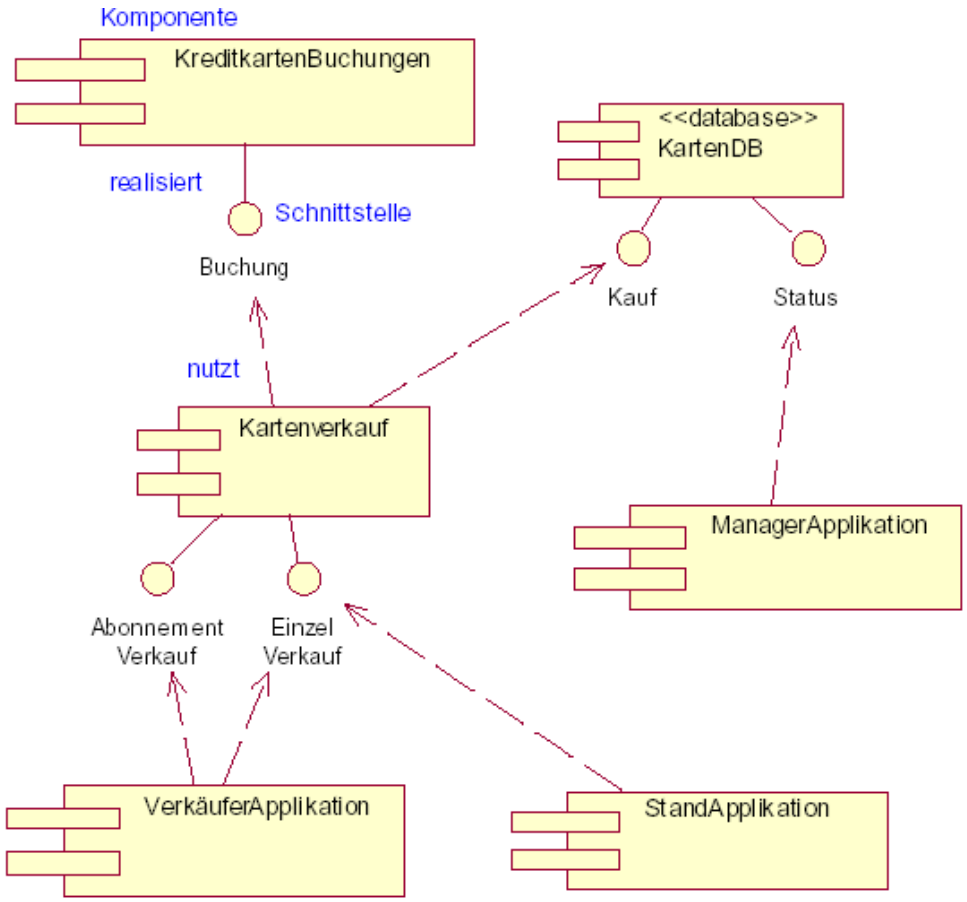
- *Komponente* = Rechteck, das am linken Rand zwei kleine Rechtecke enthält



- *Schnittstelle* = Kreis

- Komponente *bietet* Schnittstelle *an* ⇒ Komponente und Schnittstelle durch *Assoziation* verbunden
- Komponente *nutzt* Schnittstelle ⇒ Komponente und Schnittstelle durch *Abhängigkeit* verbunden

Komponentendiagramm
Theaterkasse



Verteilungs-Sicht (Deployment View)

Zeigt Anordnung der Komponenten auf den zur Verfügung stehenden Knoten zur Laufzeit

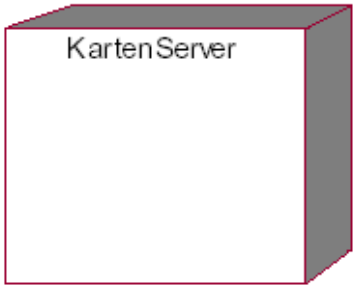
Knoten **(Node) = zur Laufzeit physisch vorhandenes Objekt, das über Rechenleistung bzw. Speicher verfügt (z.B. Computer, Eingabeterminal, RAID, ...)**

Verteilungs-Sicht wird durch *Verteilungsdiagramme* (Deployment diagrams) dargestellt

Verteilungs-Sicht (Forts.)

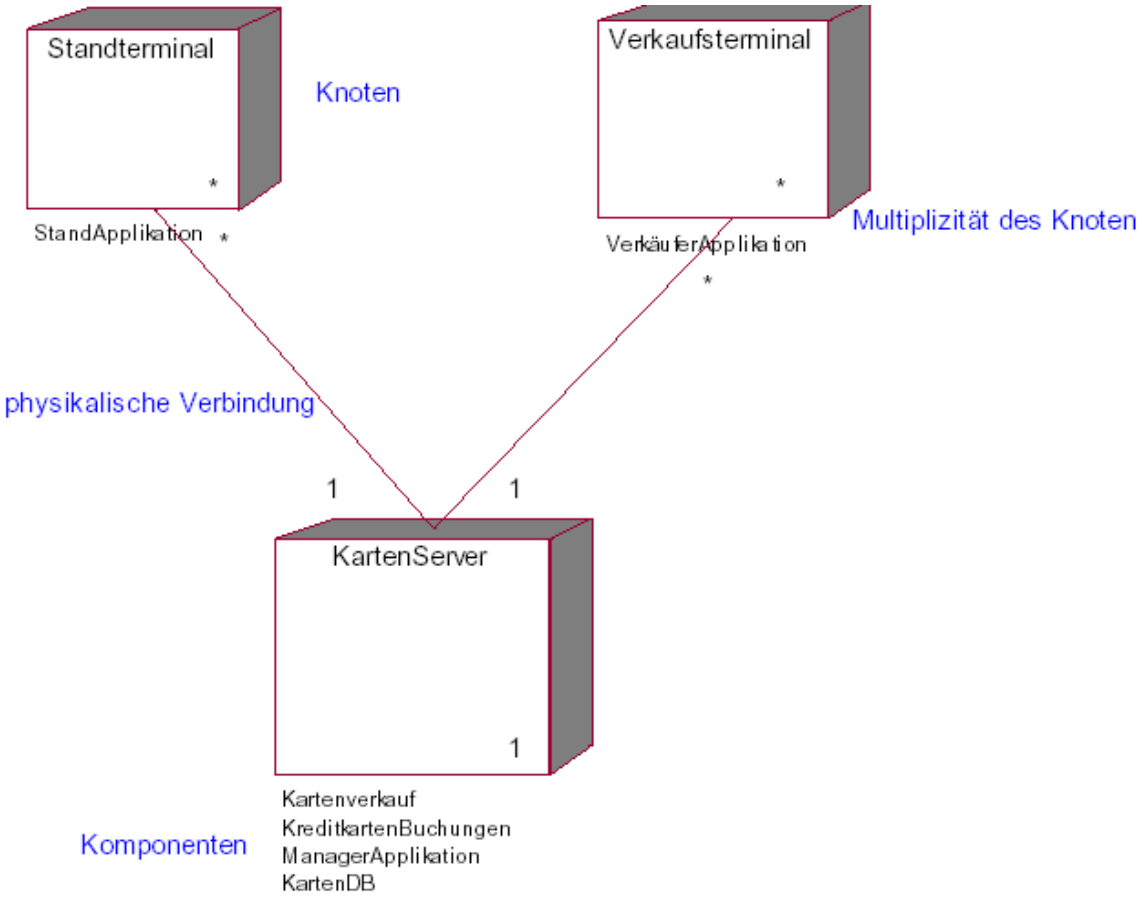
Graphische Darstellung:

- *Knoten* = Quader



- Physikalisch Verbindungen zwischen Knoten ⇒ Assoziation
- Plazierung von Komponenten und deren Abhängigkeiten innerhalb von Knoten möglich

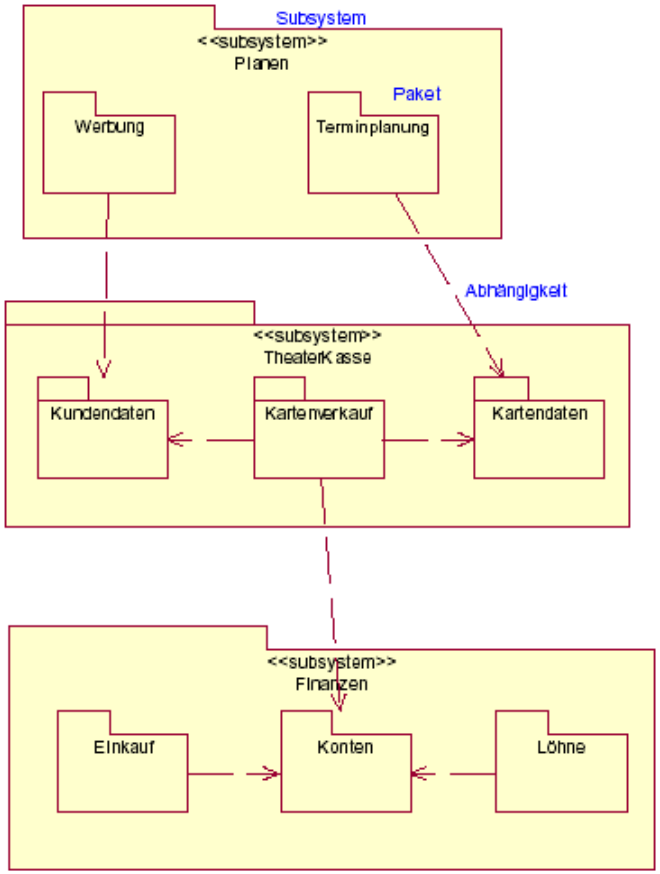
Verteilungsdiagramm
Theaterkasse



Modell Management Sicht

- **Organisation und Strukturierung des Modells selbst in Pakete und Subsysteme**
- ***Modell* (Model) = komplette Beschreibung des Systems**
- ***Paket* (Package) enthält Modellierungselemente und u. U. weitere Pakete**
- ***Subsystem* = Teil des Systems, der isoliert betrachtet ein eigenständiges System darstellt**

Organisation des Modells *Theater*



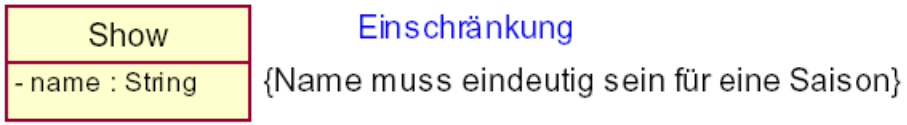
Erweiterbarkeitskonstrukte (Extensibility Constructs)

UML enthält drei Klassen von Erweiterbarkeitskonstrukten

- ***Einschränkungen*** **(Constraints)**
- ***Eigenschaftswerte*** **(Tagged values)**
- ***Stereotypen*** **(Stereotype)**

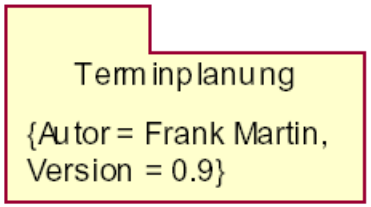
Einschränkung (Constraint)

- **Ausdruck, der die möglichen Inhalte, Zustände oder die Semantik eines Modellelements einschränkt und stets erfüllt sein muss**



Eigenschaftswert (Tagged value)

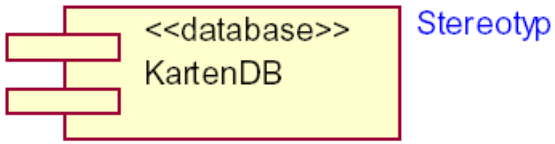
- **Eine Information - bestehend aus Attribut und zugehörigem Wert - die an ein beliebiges Modellierungselement angehängt werden kann**



Eigenschaftswerte

Stereotyp

- **Ein neuartiges Modellierungselement, welches auf einem bereits vorhandenen basiert**
  - **zusätzliche Eigenschaften (Eigenschaftswerte)**
  - **speziellere Semantik (Einschränkungen)**
  - **neue Darstellung**



Notizen (Notes)

Konstrukte, die zur Darstellung eines Kommentars oder anderer textueller Information dienen, z.B.

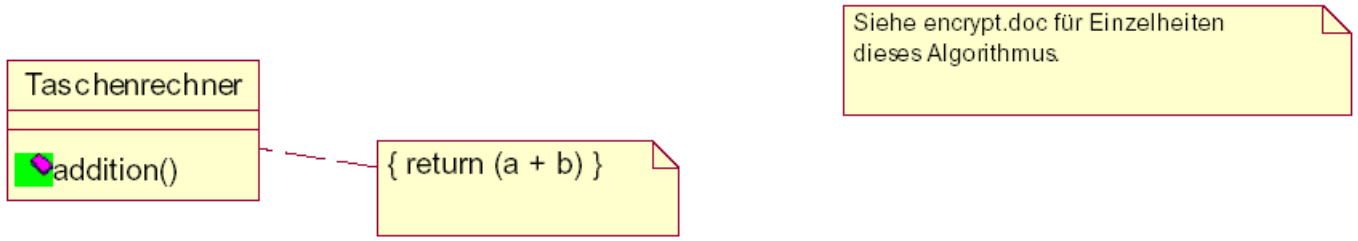
- **Quell-Code Fragmente**
- **Einschränkungen**

Notizen (Forts.)

Graphische Darstellung:

– *Notiz* = Rechteck mit Eselsohr, welches Text oder Verweis auf Dokument enthält

– Verbindung zum zugehörigen Element = gestrichelte Linie



Zusammenfassung - Sichten, Diagramme und UML-Konstrukte

Bereich	*Sicht*	*Diagramme*	*Hauptkonzepte*
Struktur	Statische Sicht	Klassendiagramm	Klasse, Assoziation, Generaliserung, Abhängigkeit, Realisierung, Schnittstelle
	Anwendungsfall-Sicht	Anwendungsfalldiagramm	Anwendungsfall, Akteur, Assoziation, extend, include, Anwendungsfall, Generalisierung
	Implementations-Sicht	Komponentendiagramm	Komponente, Schnittstelle, Abhängigkeit, Realisierung
	Verteilungs-Sicht	Verteilungsdiagramm	Knoten, Komponente, Abhängigkeit, Ort
Dynamisch	Zustandsmaschinen-Sicht	Zustandsübergangs-diagramm	Zustand, Ereignis, Übergang, Aktion
	Aktivitäts-Sicht	Aktivitätsdiagramm	Zustand, Aktivität, Beendigungsübergang, Verzweigung, Synchronisation
	Interaktions-Sicht	Sequenzdiagramm	Interaktion, Objekt, Nachricht, Aktivierung
		Kollaborationsdiagramm	Kollaboration, Interaktion, Kollaborationsrolle, Nachricht
Modell Mgmt.	Modell Mgmt. Sicht	Klassendiagramm	Paket, Subsystem, Modell
Erweiter-barkeit	Alle	Alle	Einschränkung, Stereotyp, Eigenschaftswert

2. Einleitung und Überblick über UML

2.1 Strukturierung der UML-Konstrukte

2.2 Überblick über UML-Konstrukte

3. UML im Detail: Klassendiagramme

Klassen stellen das „Vokabular“ des Systems dar

Sie bilden die Basis, auf der alle anderen Konstrukte aufbauen

Klassen - Findungsprozess

- **Identifikation von „Dingen“, die Benutzer und Entwickler zur Beschreibung des Problems und der Lösung benutzen**
- **Hilfsmittel:**
  - **Anwendungsfall-basierte Analyse**
  - **CRC-Karten**

  (CRC = Class, Responsibility, Collaborations)
  - **...**

Klassen - Verantwortlichkeiten (Responsibilities)

- **In der Analyse-Phase werden Klassen *Verantwortlichkeiten* zugeteilt; Ausgewogenheit ist wichtig**
- **Erst in der Entwurfs- und Implementationsphase werden Attribute und Operationen festgelegt, um die speziellen Verantwortlichkeiten zu bewerkstelligen**

Klassen - Attribute und Operationen - Sichtbarkeit (Visibility)

- ***Sichtbarkeit*** **(Visibility) = Festlegung der Zugriffsrechte anderer Klassen auf die Attribute bzw. Operationen einer Klasse**

– *public* (+)

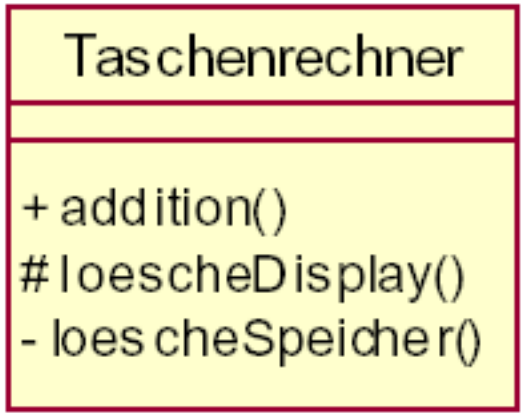
Jede beliebige Klasse darf zugreifen

– *protected* (#)

Nur die definierende Klasse selbst und von dieser abgeleitete Klassen dürfen zugreifen

– *private* (-)

Graphische Darstellung:



Klassen - Attribute

- ***Attribut*** **(Attribute) = mit Namen versehene *Eigenschaft* einer Klasse**
- **Klasse kann beliebig viele (auch keine) Attribute besitzen**

Klassen - Attribute (Forts.)

Verschiedene Detaillierungsgrade möglich

Generelle Syntax:

```
[Sichtbarkeit] Name [[Multiplizität]][:Typ]
[= initialer Wert][{Eigenschaft}]
```

Beispiele:

ursprung	**Name**
+ ursprung	**Sichtbarkeit und Name**
ursprung :Punkt	**Name und Typ**
zahlenListe [0 .. 10] :Integer	**Name, Multiplizität,Typ**
zaehler : Integer = 0	**Name, Typ,Initialisierung**
id : Integer = 4711 {frozen}	**Name, Typ,Eigenschaft**

Klassen - Attribute (Forts.)

Eigenschaften für Attribute

– `changeable`

keine Restriktion (Standard)

– `frozen`

Wert darf nicht mehr geändert werden, nachdem das Objekt initialisiert ist

Klassen - Operationen

Operation **= mit Namen versehenes** ***Verhalten*** **einer Klasse, das für jedes Objekt dieser Klasse angefordert werden kann**

Operationen können den Zustand eines Objektes verändern

Verschiedene Detaillierungsgrade möglich

Generelle Syntax:

```
[Sichtbarkeit] Name [(Parameter-Liste)]
[:Rückgabe-Typ][{Eigenschaft}]
```

Beispiele:

display	**Name**
+ display	**Sichtbarkeit und Name**
setze(n:Name, s:String)	**Name und Parameter**
getId():Integer	**Name, Parameter, Rückgabe-Typ**

Klassen - Operationen (Forts.)

Signatur = `Name [(Parameter-Liste)]`

Einzelnen Parameter der Signatur haben die folgende Syntax:

```
[Richtung] Name : Typ [= Standard-Wert]
```

Klassen - Operationen (Forts.)

Richtungen

`in`

Eingabe-Parameter; darf nicht geändert werden

`out`

Ausgabe-Parameter; Übermittlung von Information zum Aufrufer

`inout`

Eingabe-Parameter, der modifiziert werden kann

Klassen - Operationen (Forts.)

Eigenschaften für Operationen:

- `leaf`

Operation darf nicht überschrieben werden

(Java: `final`, C++: nicht-virtuelle Funktion)

- `isQuery`

Ausführung der Operation ändert den Systemzustand nicht (keine Seiteneffekte)

(C++: `const` Funktion)

Klassen - Abstrakte Klassen und Operationen

In komplexeren Klassenhierarchien werden üblicherweise gewisse Klassen als *abstrakt* (abstract) spezifiziert

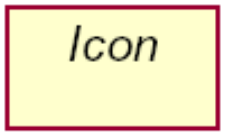
Abstrakte Klasse **= Klasse, von der es keine direkten Objekte geben darf**

(Java: `abstract`, C++: mind. eine rein virtuelle Funktion)

Zur Nutzung der Funktionalität von abstrakten Klassen leitet man eigene Klassen von diesen ab

Graphische Darstellung:

- Klassenname wird kursiv geschrieben



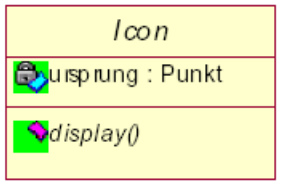
Klassen - Abstrakte Klassen und Operationen (Forts.)

Ebenso lassen sich Operationen als abstrakt spezifizieren (Java: `abstract`, C++: rein virtuelle Funktion)

Abstrakte Operationen *müssen* in abgeleiteten Klassen implementiert werden

Graphische Darstellung:

– Operationsname wird kursiv geschrieben



Generalisierung (Generalization)

Generalisierung **(Generalization) = Beziehung zwischen einem allgemeinen Element (hier *Oberklasse* (Superclass)) und einem spezielleren Element (hier *Unterklasse* (Subclass))**

Durch Generalisierung *erbt* (inherits) die Unterklasse die Struktur und das Verhalten der Oberklasse

Generalisierung (Forts.)

Unterklasse erweitert oder modifiziert i.a. Funktionalität der Oberklasse

Vorteile der Generalisierung:

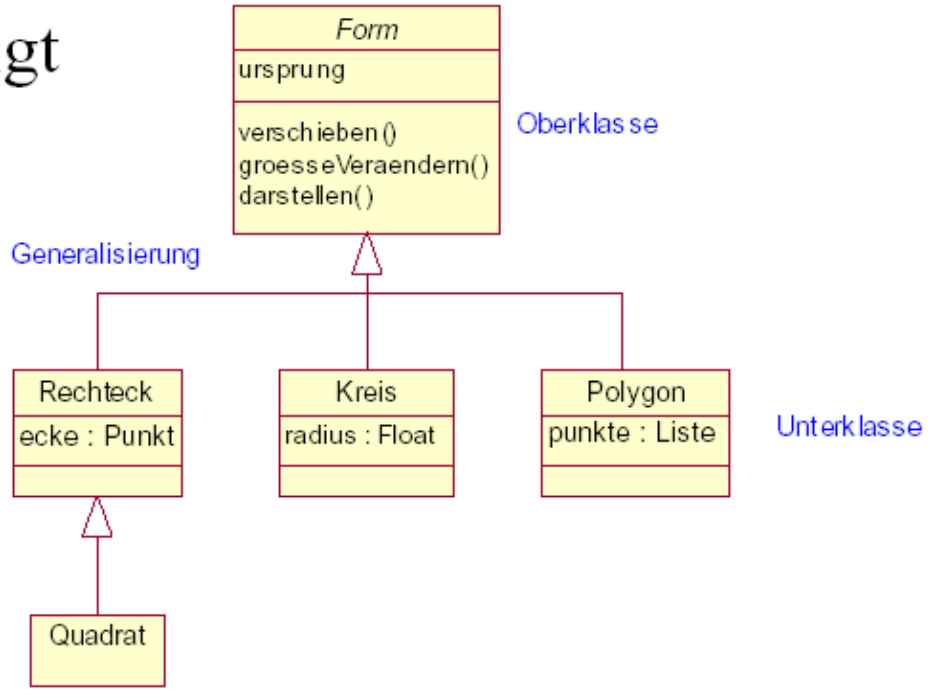
– Reduzierung des Implementationsaufwandes

– Oberklasse ist durch Unterklasse ersetzbar

– Nutzung von Polymorphie

Graphische Darstellung:

– Linie mit Dreieck an dem Ende, welches auf die Oberklasse zeigt



Assoziation (Association)

Assoziation **(Association) = strukturelle Beziehung zwischen Elementen**

Mittels Assoziation modelliert man die direkte Zugriffsmöglichkeit der beteiligten Elemente aufeinander

Assoziation (Forts.)

Graphische Darstellung:

– Linie zwischen beteiligten Elementen

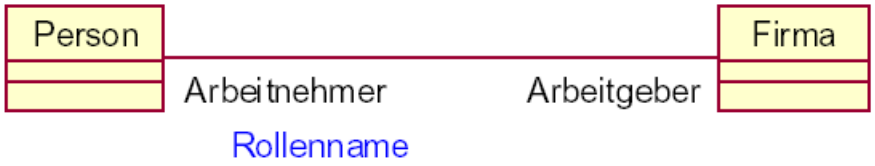
Assoziation - Namen

Assoziation kann mit einem *Namen* versehen werden, um die Beziehung zu verdeutlichen

Optional: Angabe der Richtung, in welcher der Name gelesen wird

Assoziation - Rolle

Klassen, die an einer Assoziation beteiligt sind, spielen in diesem Kontext eine gewisse *Rolle*



Dieselbe Klasse kann verschiedene Rollen in anderen Assoziationen spielen

Assoziation - Multiplizität

Multiplizität **einer Rolle = Anzahl der Verbindungen, die es für eine Rolle geben kann**

Assoziation - Navigation

Klasse *A* und Klasse *B* durch Assoziationverbunden ⇒ von Objekt der Klasse *A* kann man zu Objekt der Klasse *B* *navigieren* und umgekehrt

Einschränkung der Navigierbarkeit durch explizite Spezifikation der *Richtung* mit Pfeilspitze

Assoziation - Navigation (Forts.)

Beispiel

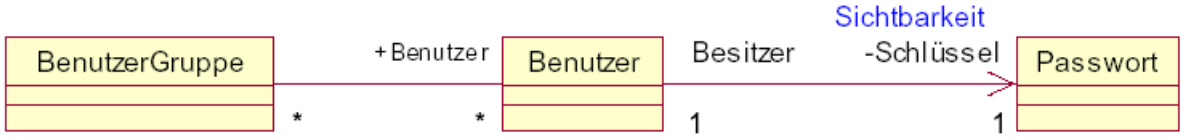
Einschränkung der Navigierbarkeit bedeutet *nicht*, dass überhaupt nicht auf das andere Objekt zugegriffen werden kann

Im Vordergrund steht direkter, einfacher Zugriff (i.a. durch Speichern von Referenz/Pointer)

Assoziation - Sichtbarkeit

Für Assoziation lässt sich - wie bei Attributen und Operationen - *Sichtbarkeit* festlegen (Kennzeichnung am Rollennamen)

Beispiel:



Nur von Objekt der Klasse `Benutzer` selbst darf auf das `Passwort` Objekt zugegriffen werden

Anwendungsbeispiel: Monopoly

Ziel: Modellierung der wichtigsten Klassen und Beziehungen im Spiel Monopoly

Sukzessive Entwicklung:

– Klassen finden

– Verantwortlichkeiten festlegen

– Attribute, Operationen und Beziehungen festlegen

Monopoly - Klassen finden

- **Konkrete Gegenstände:**
  - **Spieler**
  - **Strasse**
  - **Spielbrett**
  - **Spielfeld**
  - **Bahnhof**
- **Abstrakte Gegenstände:**
  - **MonopolySpiel**

Monopoly - Verantwortlichkeiten festlegen

- **Spieler**

– **Spielerdaten speichern**

– **Konto verwalten**

– **Besitz verwalten**

– **Aktionen veranlassen**

Monopoly - Verantwortlichkeiten festlegen (Forts.)

- **Strasse**

– Informationen speichern/bereitstellen (Kaufpreis, aktueller Mietpreis, Anzahl Häuser, Hotel)

– Kauf/Verkauf

– Hypothek

- **Spielfeld**

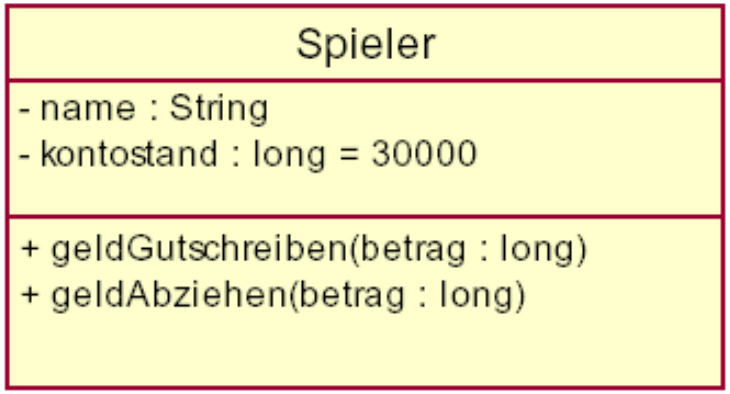
– Steuerung der Aktionen, die beim Betreten passieren

Monopoly - Verantwortlichkeiten festlegen (Forts.)

- **MonopolySpiel**
  - **Spielverlauf steuern (Hauptschleife)**
  - **Spielstand verwalten**

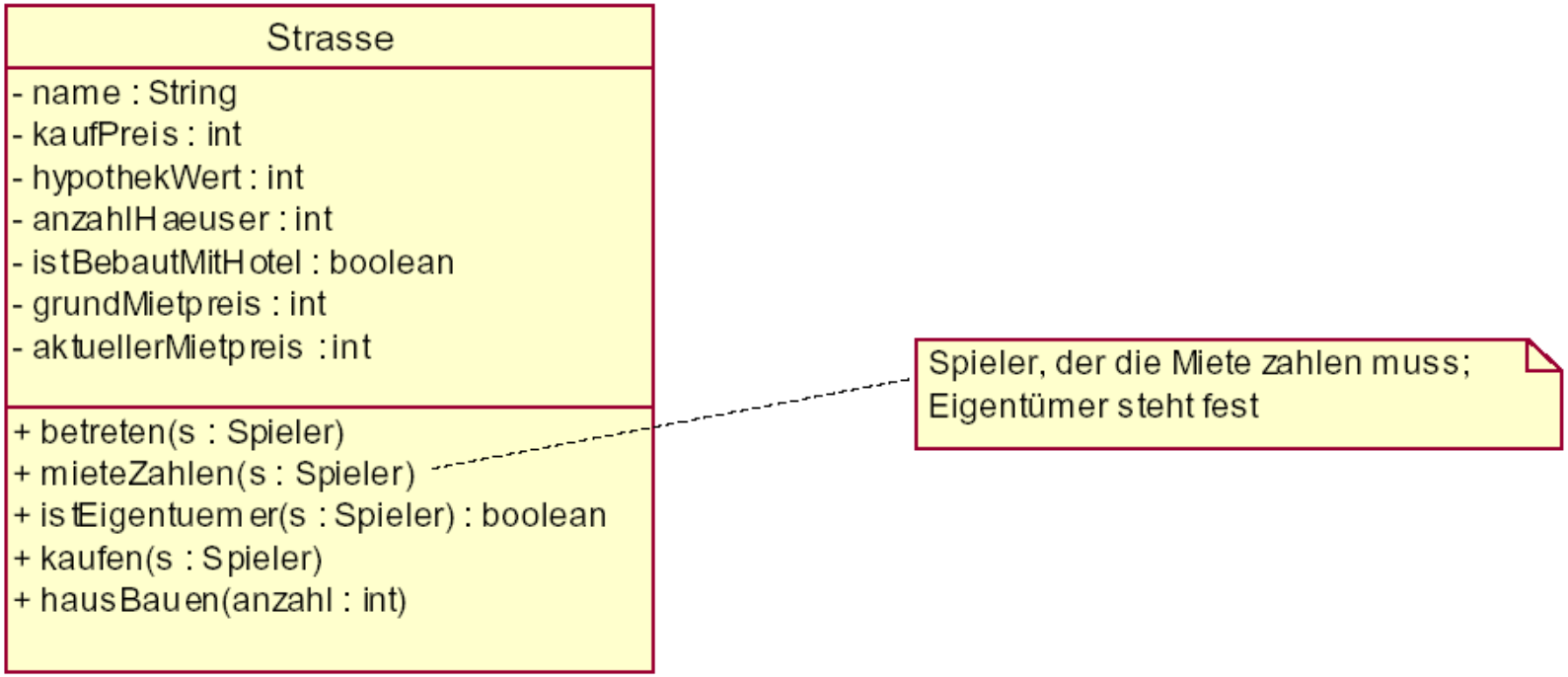
Monopoly - Attribute und Operationen festlegen

- Klasse Spieler



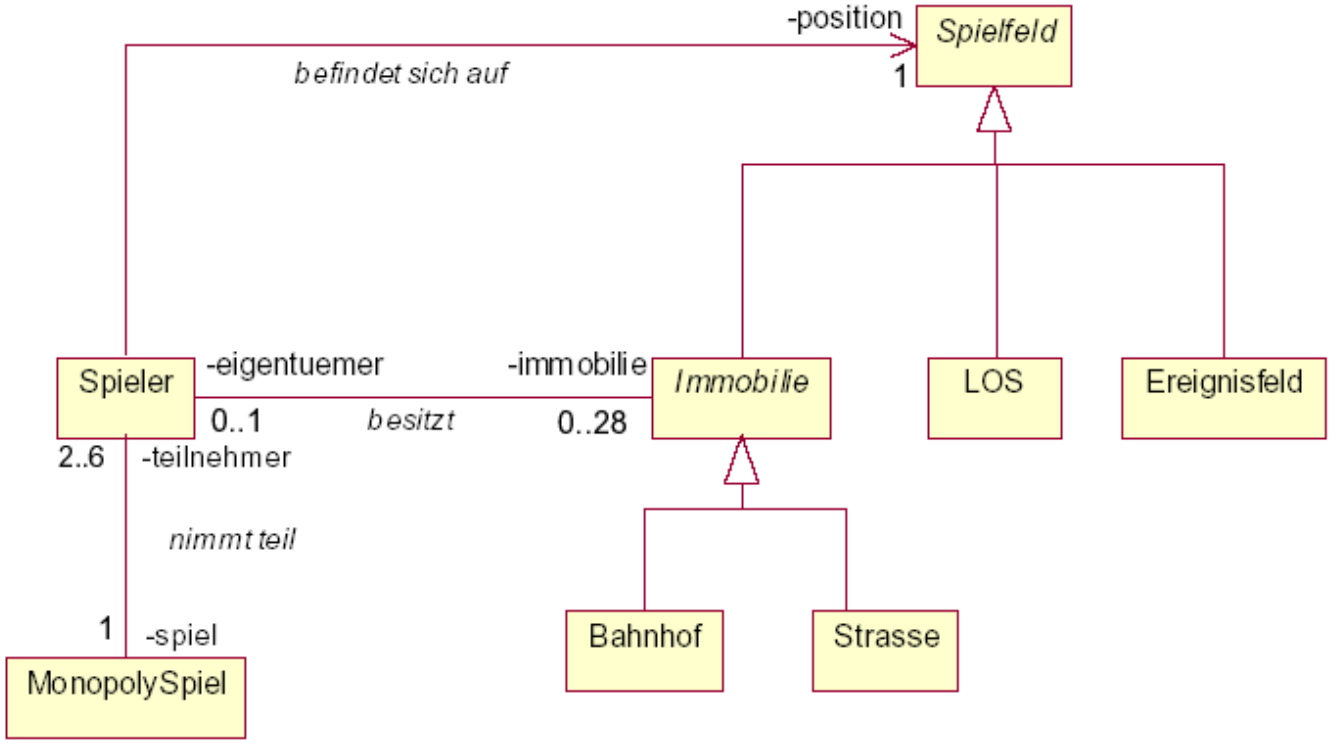
Monopoly - Attribute und Operationen festlegen (Forts.)

- Klasse Strasse



Monopoly - Beziehungen festlegen/identifizieren

- Klassendiagramm Spieler Beziehungen:



Realitätsbezug

Erzählen Sie ein Beispiel oder
eine Anekdote aus dem wirklichen Leben.

Zeigen Sie Mitgefühl mit der Situation des Publikums, wenn angemessen.

Was bedeutet das?

Machen Sie eine eindeutige Aussage, die Ihre Meinung über das Thema zusammenfasst.

Fassen Sie die Hauptargumente zusammen, an die sich Ihr Publikum später erinnern soll.

Nächste Schritte

Fassen Sie zusammen, welche Schritte von Ihrer Zielgruppe gefordert sind.

Fassen Sie alle Aktionsschritte zusammen, die von Ihnen noch zu erledigen sind.

UML Einführung
Inhalte
1.1 Motivation
Motivation (Forts.)
Motivation (Forts.)
Motivation (Forts.)
1.2 Was ist UML?
Visualisierung
Spezifikation
Konstruktion
Konstruktion (Forts.)
Dokumentation
Was ist UML? (Forts.)
2.1 Strukturierung der UML-Konstrukte
Sichten
Personengruppen im Softwareentwicklungsprozess
Sichten - grobe Unterteilung
Sichten - grobe Unterteilung (Forts.)
Sichten - grobe Unterteilung (Forts.)
2.2 Überblick über UML-Konstrukte
Statische Sicht (Static View)
Statische Sicht (Forts.)
Statische Sicht (Forts.)
Klassendiagramm Theaterkasse
Anwendungsfall-Sicht (Use Case View)
Anwendungsfall-Sicht (Forts.)
Anwendungsfalldiagramm Theaterkasse
Interaktions-Sicht (Interaction View)
Sequenzdiagramm
Sequenzdiagramm (Forts.)
Sequenzdiagramm kaufe Eintrittskarten
Kollaborationsdiagramm
Kollaborationsdiagramm (Forts.)
Kollaborationsdiagramm reserviere Eintrittskarten
Zustandsmaschinen-Sicht (State Machine View)
Zustandsmaschinen-Sicht (Forts.)
Zustandsübergangsdiagramm Eintrittskarte
Aktivitäts-Sicht (Activity View)
Aktivitäts-Sicht (Forts.)
Aktivitätsdiagramm Planung/Ausführung einer Show
Physische Sichten (Physical Views)
Implementations-Sicht (Implementation View)
Implementations-Sicht (Forts.)
Komponentendiagramm Theaterkasse
Verteilungs-Sicht (Deployment View)
Verteilungs-Sicht (Forts.)
Verteilungsdiagramm Theaterkasse
Modell Management Sicht
Organisation des Modells Theater
Erweiterbarkeitskonstrukte (Extensibility Constructs)
Einschränkung (Constraint)
Eigenschaftswert (Tagged value)
Stereotyp
Notizen (Notes)
Notizen (Forts.)
Zusammenfassung - Sichten, Diagramme und UML-Konstrukte
2. Einleitung und Überblick über UML
3. UML im Detail: Klassendiagramme
Klassen - Findungsprozess
Klassen - Verantwortlichkeiten (Responsibilities)
Klassen - Attribute und Operationen - Sichtbarkeit (Visibility)
Klassen - Attribute und Operationen - Sichtbarkeit (Forts.)
Klassen - Attribute
Klassen - Attribute (Forts.)
Klassen - Attribute (Forts.)
Klassen - Operationen
Klassen - Operationen (Forts.)
Klassen - Operationen (Forts.)
Klassen - Operationen (Forts.)
Klassen - Abstrakte Klassen und Operationen
Klassen - Abstrakte Klassen und Operationen (Forts.)
Klassen - Abstrakte Klassen und Operationen (Forts.)
Generalisierung (Generalization)
Generalisierung (Forts.)
Generalisierung (Forts.)
Assoziation (Association)
Assoziation (Forts.)
Assoziation - Namen
Assoziation - Rolle
Assoziation - Multiplizität
Assoziation - Navigation
Assoziation - Navigation (Forts.)
Assoziation - Sichtbarkeit
Anwendungsbeispiel: Monopoly
Monopoly - Klassen finden
Monopoly - Verantwortlichkeiten festlegen
Monopoly - Verantwortlichkeiten festlegen (Forts.)
Monopoly - Verantwortlichkeiten festlegen (Forts.)
Monopoly - Attribute und Operationen festlegen
Monopoly - Beziehungen festlegen/identifizieren
Realitätsbezug
Was bedeutet das?
Nächste Schritte