Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

Modul Software-Entwicklung 1 (SWEN1)

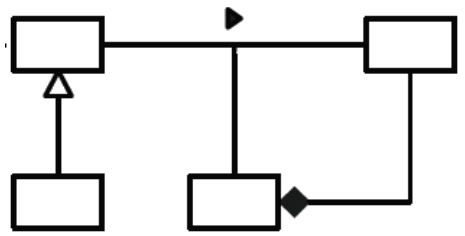
LE 06 – Softwarearchitektur und Design II

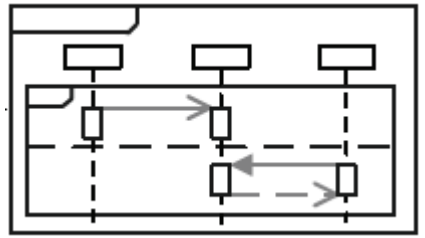
SWEN1/PM3 Team:
R. Ferri (feit), D. Liebhart (lieh), K. Bleisch (bles), G. Wyder (wydg)

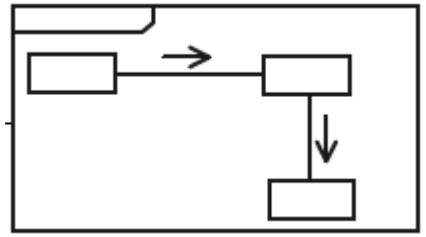
Ausgabe: HS24

Um was geht es?

- Wie realisiere ich einen Use Case mit Klassen, die klare Verantwortlichkeiten haben, wartbar und einfach erweiterbar sind?
- Wie modelliere ich mein Design mit der UML, um es diskutieren und evaluieren zu können?







Lernziele LE 06 – Softwarearchitektur und Design II

- Sie sind in der Lage:
  - den Zweck und die Anwendung von statischen und dynamischen Modellen im Design zu erläutern,
  - einen Objektentwurf zweckmässig mit UML-Klassen-, UML-Interaktions-, UML-Zustands- und UML-Aktivitätsdiagrammen darzustellen.
  - die Idee von Verantwortlichkeiten und des Responsibility-Driven Designs (RDD) für den Entwurf von Klassen zu erklären,
  - grundlegende Prinzipen und Pattern für den Klassenentwurf anzuwenden (GRASP, SOLID),

Agenda

1. Einführung in das objektorientierte Design
2. UML-Diagramme für das Design
3. Klassen mit Verantwortlichkeiten entwerfen
4. Wrap-up und Ausblick

Recap: Objektorientierung

- Was bedeutet Objektorientierung?

Recap: Objektorientierte Analyse und objektorientiertes Design

domain concept

Plane
tailNumber

visualization of domain concept

representation in an object-oriented programming language

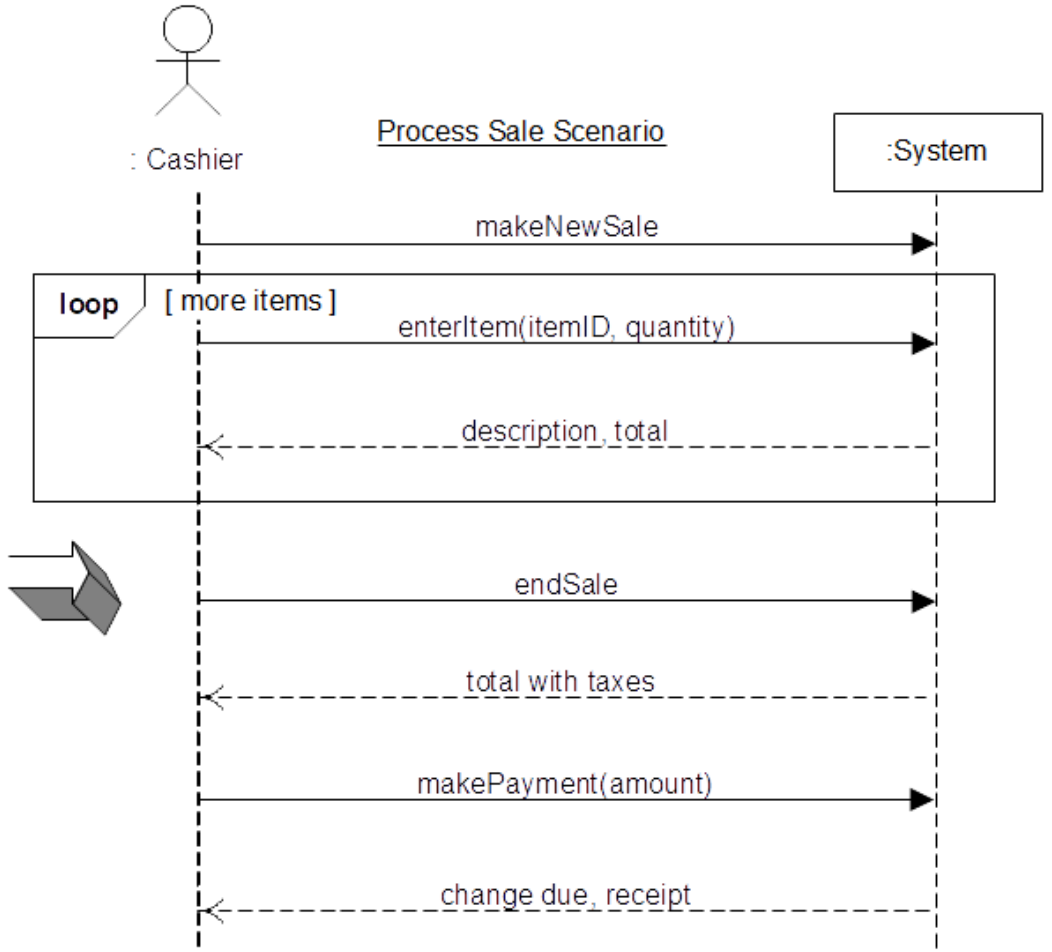
```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

Use Cases und System-Sequenzdiagramm (SSD)

- Szenarien und Systemoperationen, die in den Use Cases identifiziert wurden, zusammen mit dem Domänenmodell bilden die Basis für das Design.
- Die Systemoperation bzw. deren Antworten sind schlussendlich das, was programmiert werden muss.

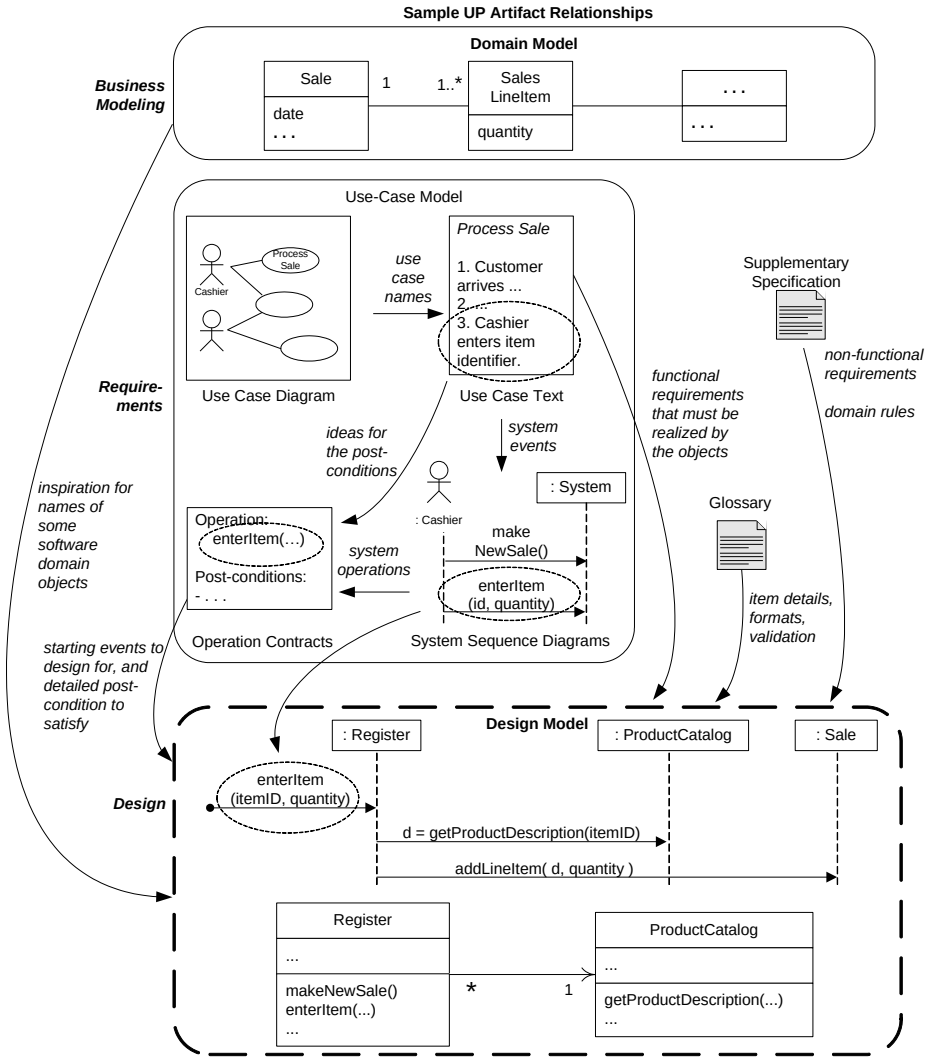
Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



Use Cases und Use-Case-Realisierung

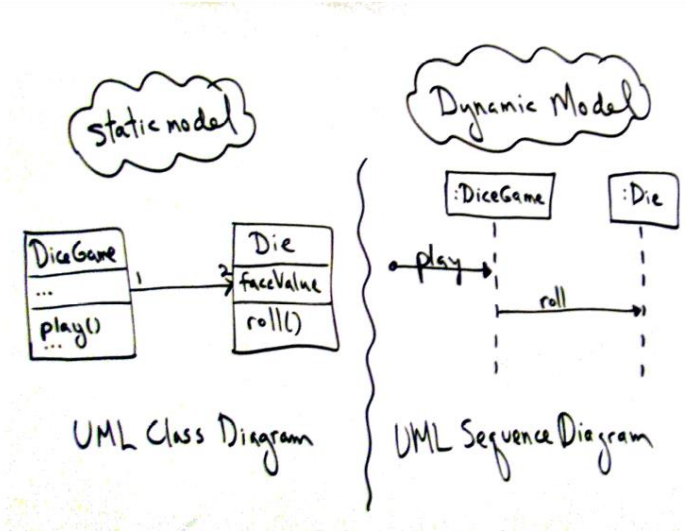
- Eine Use-Case-Realisierung beschreibt, wie ein bestimmter Use Case innerhalb des Designs mit kollaborierenden Objekten realisiert wird.
- Jedes Szenario eines Use Cases bzw. dessen Systemoperationen werden schrittweise entworfen und implementiert.
- Die UML-Diagramme sind eine gemeinsame Sprache, um Use-Case-Realisierungen zu veranschaulichen und zu diskutieren.



Klassen entwerfen: Statische und dynamische Modellierung

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

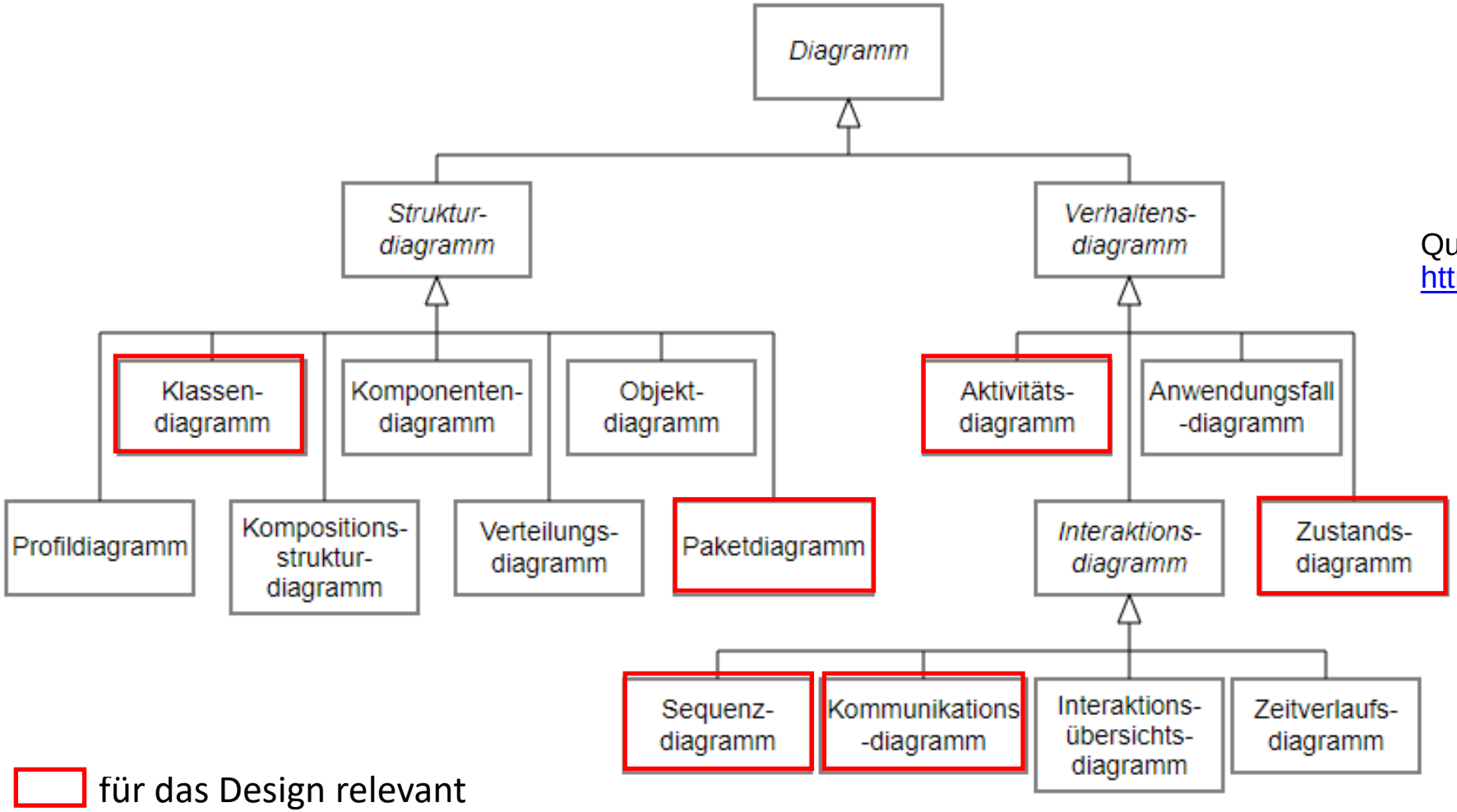
- Es gibt zwei Arten von Design-Modellen:
  - **Statische Modelle**
    - Statische Modelle, wie beispielsweise das UML-Klassendiagramm, unterstützen den Entwurf von Paketen, Klassennamen, Attributen und Methodensignaturen (ohne Methodenkörper).
  - **Dynamische Modelle**
    - Dynamische Modelle, wie beispielsweise UML-Interaktionsdiagramme, unterstützten den Entwurf der Logik, des Verhaltens des Codes und der Methodenkörper.
- Statische und dynamische Modelle ergänzen sich.
- Statische und dynamische Modelle werden parallel erstellt.



Agenda

1. Einführung in das objektorientierte Design
2. **UML-Diagramme für das Design**
3. Klassen mit Verantwortlichkeiten entwerfen
4. Wrap-up und Ausblick

Die Diagramme der UML

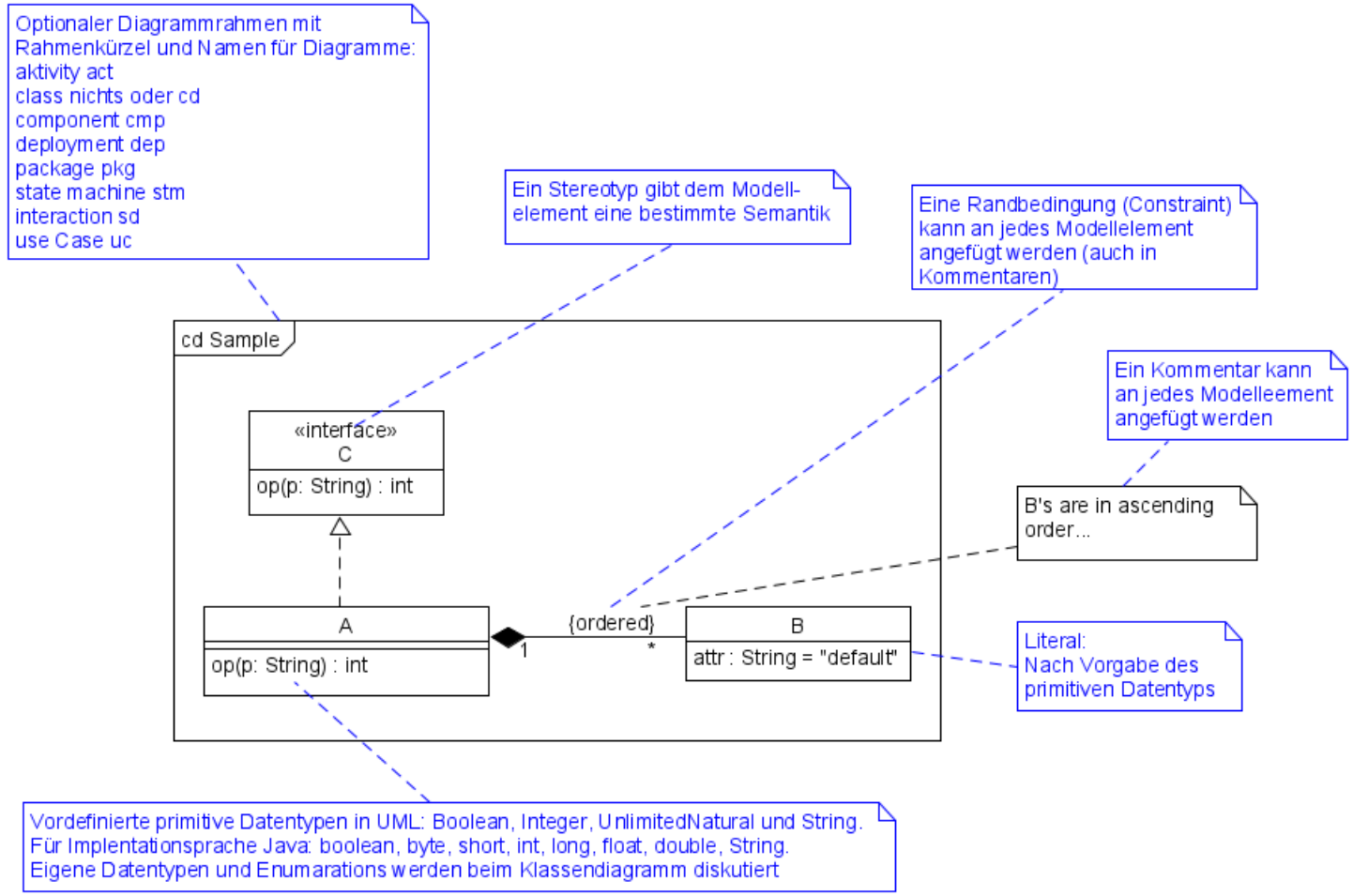


für das Design relevant

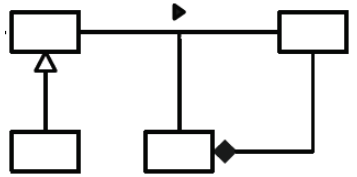
Quelle: UML Specification, https://www.omg.org/spec/UML/

Grundelemente der UML

- Grundlegende Notationselemente:
  - Primitiver Datentyp
  - Literal
  - Schlüsselwort, Stereotyp
  - Randbedingung (constraint)
  - Kommentar
  - Diagrammrahmen (optional)



UML-Klassendiagramm (1/7)



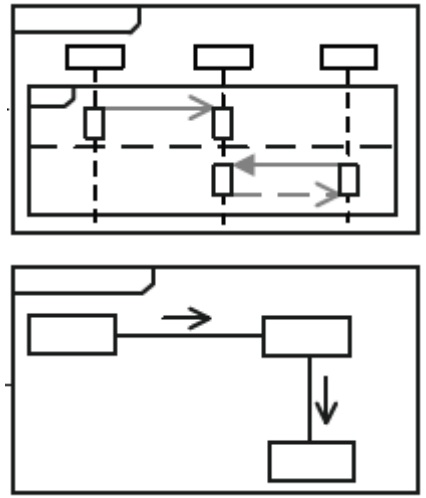
- Das Diagramm beantwortet die zentrale Frage:
  Aus welchen Klassen besteht mein System und wie sind sie miteinander verknüpft?
- Es beschreibt die statische Struktur des zu entwerfenden oder abzubildenden Systems.
  - Welche Klassen und Objekte existieren im System
  - Welche Attribute, Operationen und Beziehungen haben sie untereinander
  - Es enthält alle relevanten Strukturzusammenhänge und Datentypen.
- Es bildet die Brücke zwischen den dynamischen Diagrammen.
- Das UML-Klassendiagramm kann für mehrere Zwecke verwendet werden:
  - In der Konzeptphase als Domänenmodell mit einem vereinfachten UML-Klassendiagramm (Problemdomäne).
  - In der Designphase als Design-Klassendiagramm (DCD) mit zusätzlichen Notationselementen (Lösungsdomäne).

UML-Interaktionsdiagramme

- Ein Interaktionsdiagramm spezifiziert, auf welche Weise Nachrichten und Daten zwischen Interaktionspartnern ausgetauscht werden.

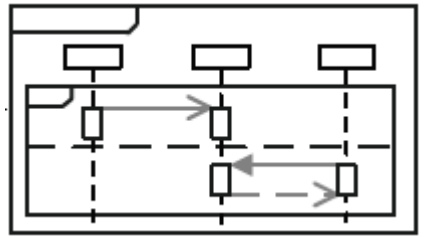
- Es gibt zwei Arten von UML-Interaktionsdiagrammen:
  - Sequenzdiagramm
  - Kommunikationsdiagramm

- Modellieren die Kollaborationen bzw. den Informationsaustausch zwischen Objekten (Dynamik).



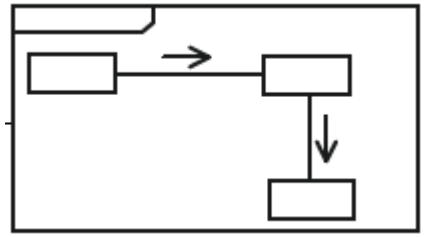
UML-Sequenzdiagramm (1/4)

- Das Diagramm beantwortet die zentrale Frage:
  Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus?
- Es stellt den zeitlichen Ablauf des Informationsaustausches zwischen Kommunikationspartnern dar.
- Es sind Schachtelung und Flusssteuerung (Bedingungen, Schleifen, Verzweigungen) möglich.



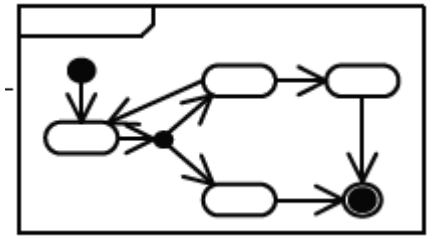
UML-Kommunikationsdiagramm (1/3)

- Das Diagramm beantwortet die zentrale Frage:
  Wer kommuniziert mit wem? Wer «arbeitet» im System zusammen?
- Es stellt ebenfalls den Informationsaustausch zwischen Kommunikationspartnern dar.
- Der Überblick steht im Vordergrund (Details und zeitliche Abfolge sind weniger wichtig).



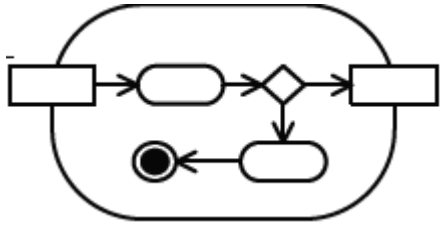
UML-Zustandsdiagramm (1/4)

- Das Diagramm beantwortet die zentrale Frage:
  Welche Zustände kann ein Objekt, eine Schnittstelle, ein Use Case, … bei welchen Ereignissen annehmen?
- Präzise Abbildung eines Zustandsmodells (endlicher Automat) mit Zuständen, Ereignissen, Nebenläufigkeiten, Bedingungen, Ein- und Austrittsaktionen.
- Zustände können wieder aus Zuständen bestehen (Schachtelung möglich).
- Das Zustandsdiagramm wird vor allem in der Modellierung von Echtzeitsystemen, Steuerungen und Protokollen verwendet.



UML-Aktivitätsdiagramm (1/3)

- Das Diagramm beantwortet die zentrale Frage:
  Wie läuft ein bestimmter Prozess oder ein Algorithmus ab?
- Es kann eine sehr detaillierte Visualisierung von Abläufen mit Bedingungen, Schleifen und Verzweigungen modelliert werden.
- Es sind Parallelisierung und Synchronisation von Aktionen möglich.



Weitere Informationen zur Modellierung mit der UML

- Diese Einführung in die Modellierung mit der UML und in die verschiedenen Diagramme für das Design ist eine kompakte Zusammenfassung, ohne detaillierte Erläuterung der Semantik.
- Sie umfasst der wichtigsten Notationselemente, mit denen ca. 80% der Problemstellungen modelliert werden können.

- *Achtung: Um damit in der Praxis modellieren zu können, müssen die Diagramme und Notationselemente in verschiedenen Problemstellungen angewendet und eingeübt werden!*

Tipps zur Modellierung

- Modellieren-im-Team: Skizzieren Sie erste Versionen im Team am Whiteboard. Verzichten Sie (vorläufig) auf komplexe Grafik- oder UML-Werkzeuge.
- Gerade-gut-genug: Vermeiden Sie das Streben nach Vollständigkeit Ihrer Modelle – meistens genügen Ausschnitte oder Teile des Systems für das Verständnis!
- Gerade-rechtzeitig: Verzögern Sie die Erstellung der ausgelieferten Dokumentation – es könnten sich ja noch Dinge ändern!
- Dokumentieren Sie kontinuierlich. Am Ende der letzten Iteration oder des Projekts können Sie sich nicht mehr an alle relevanten Dinge erinnern.
- Diagramm-plus-Text: Ergänzen Sie grafische Modelle um kurze textuelle Erläuterungen. Gute technische Dokumentation kombiniert Bild mit Text!
- Halten Sie Modelle möglichst redundanzfrei. Versuchen Sie, benötigte Dokumente aus einer einheitlichen Informations- oder Modellbasis zu generieren.

Agenda

1. Einführung in das objektorientierte Design
2. UML-Diagramme für das Design
3. **Klassen mit Verantwortlichkeiten entwerfen**
4. Wrap-up und Ausblick

Verantwortlichkeiten und Responsibility-Driven-Design

- Denken in Verantwortlichkeiten, Rollen und Kollaborationsbeziehungen für den Entwurf von Softwareklassen.
- Dieser Ansatz ist das Responsibility-Driven-Design (RDD).
- Softwareobjekte werden ähnlich wie Personen betrachtet, mit Verantwortlichkeiten und einer Zusammenarbeit mit anderen Personen, um eine Aufgabe zu erledigen.
- Verantwortlichkeiten werden durch Attribute und Methoden implementiert.
  - Evtl. in Zusammenarbeit mit Operationen von anderen Klassen bzw. Objekten.
- RDD kann auf jeder Ebene des Designs angewendet werden (Klasse, Komponente, Schicht).

Ausprägungen von Verantwortlichkeiten

- «Doing»-Verantwortlichkeiten (oder Algorithmen, Code)
  - Selbst etwas tun
  - Aktionen anderer Objekte anstossen
  - Aktivitäten anderer Objekte kontrollieren und steuern

- «Knowing»-Verantwortlichkeit (oder Daten, Attribute)
  - Private eingekapselte Daten
  - Verwandte Objekte kennen
  - Dinge kennen, die es ableiten oder berechnen kann
  - Daten/Objekte zur Verfügung stellen, die aus den bekannten Daten/Objekten abgeleitet oder berechnet werden können

GRASP: Ein methodischer Ansatz für das OO-Design

- GRASP (General Responsibility Assignment Software Patterns) bezeichnet eine Menge von grundlegenden Prinzipen bzw. Pattern, mit denen die Zuständigkeit bestimmter Klassen objektorientierter Systeme festgelegt wird.
- Sie beschreiben allgemein welche Klassen und Objekte wofür zuständig sein sollten (Verantwortlichkeiten und Kollaborationen).
- Diese allgemein anerkannten Regeln wurden von Craig Larman [1] systematisch beschrieben.
- Dies erleichtert die Kommunikation zwischen Softwareentwicklern und erleichtert Einsteigern als Lernhilfe das Entwickeln eines Bewusstseins für guten bzw. schlechten Code.

Prinzipien und Pattern

- Ein Prinzip ist in diesem Kontext ein Grundsatz oder Postulat, das zu einem guten objektorientierten Design führen soll.

- Ein Pattern ist ein benanntes Problem-Lösungspaar, das in neuen Kontexten angewendet werden kann.

- Weitere Design-Patterns siehe Gang-of-Four (GoF)
  - Oft wiederkehrende Problemstellungen mit detaillierten Lösungsmustern

GRASP – Neun Prinzipien bzw. Patterns

- Information Expert
- Creator
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

GRASP – Neun Prinzipien bzw. Patterns

- Bitte die Klasse durchnummerieren und Nummer merken
- Bilden Sie den Modulo 9 mit Ihrer Nummer
- Erarbeiten Sie das/die Pattern/Prinzipien selbständig. Recherchieren Sie!
  - Wozu wird das Pattern/Prinzip verwendet?
  - Wie erkenne ich das Pattern/Prinzip?
  - Wie wende ich das Pattern/Prinzip an?
  - Zeigen Sie das Pattern/Prinzip in einem UML-Diagramm

GRASP – Neun Prinzipien bzw. Patterns

- Information Expert (0)
- Creator (1)
- Controller (2)
- Low Coupling (3)
- High Cohesion (4)
- Polymorphism (5)
- Pure Fabrication (6)
- Indirection (7)
- Protected Variations (8)

- Alle Experten zu einem Thema finden sich zusammen, sichten die einzelnen Lösungsteile und erarbeiten eine gemeinsame Aussage, die vorgetragen werden kann.
- Ein Experte wird bestimmt, das Pattern/Prinzip allen vorzutragen und zu erklären.

Agenda

1. Einführung in das objektorientierte Design
2. UML-Diagramme für das Design
3. Klassen mit Verantwortlichkeiten entwerfen
4. **Wrap-up und Ausblick**

Wrap-up

- Das wichtigste Ziel des objektorientierten Designs ist es, Klassen mit klaren Verantwortlichkeiten und Kollaborationen zu entwerfen.
- Eine Use-Case-Realisierung wird mit einem Design-Klassendiagramm (DCD) und mehreren Interaktionsdiagrammen modelliert, um das Design zu diskutieren und evaluieren zu können.
- Dabei werden das Design-Klassendiagramm und die weiteren Modellierungsartefakte schrittweise erweitert und ergänzt durch jedes entworfene und implementierte Use-Case-Szenario.
- GRASP ist eine Lernhilfe, um beim Design bewusst die Verantwortlichkeiten und Kollaborationen zwischen Objekten festzulegen.
- GRASP sind grundlegende Prinzipien und Patterns, die zu einem guten objektorientierten Design führen und das Design nachvollziehbar begründen.

Ausblick

- In der nächsten Lerneinheit werden wir:
  - wichtige Aspekte für die Implementation einer Use-Case-Realisierung diskutieren.

Quellenverzeichnis

[1] Larman, C.: UML 2 und Patterns angewendet, mitp Professional, 2005

[2] Seidel, M. et al.: UML @ Classroom: Eine Einführung in die objektorientierte Modellierung, dpunkt.verlag, 2012

[3] Martin, R. C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design, mitp Professional, 2018