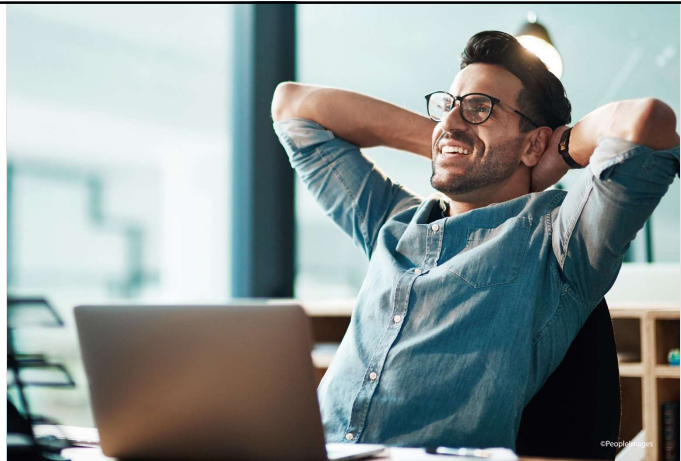




Designing Education
Connecting People

Das erwartet Sie:

- Anforderungsanalyse
- Systemdesign



Benutzerschnittstellen gestalten und konfigurieren

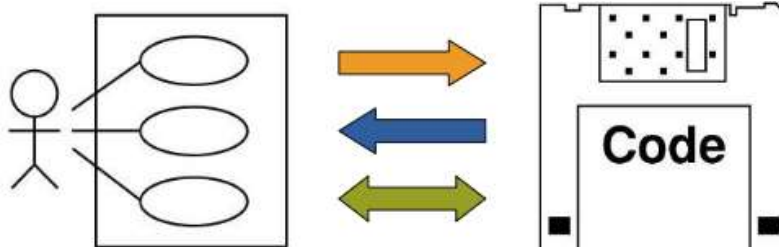
Lernfeld 10a



1

Einführung in UML

- UML ist eine standardisierte grafische Darstellungsform zur Visualisierung, Spezifikation, Konstruktion und Dokumentation von (Software-)Systemen
- Sie bietet einen Set an standardisierten Diagrammtypen, mit denen komplexe Sachverhalte, Abläufe und Systeme einfach, übersichtlich und verständlich dargestellt werden können



2

2

Einführung in UML

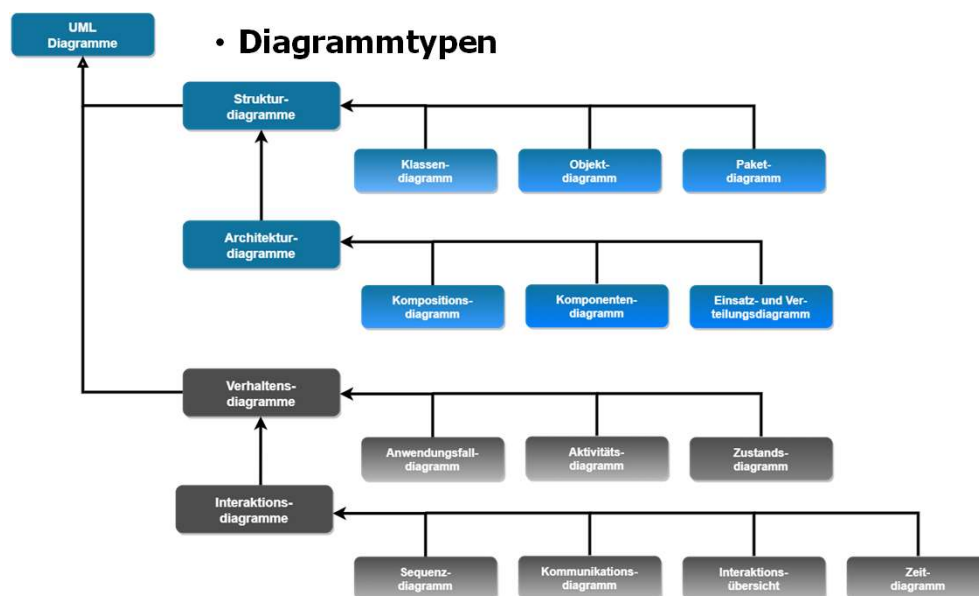
- Vorteile von UML
- Als „gemeinsame Sprache“ bei der Zusammenarbeit von Technikern und Nicht-Technikern
- Bessere Kommunikation zwischen Auftraggeber und Auftragnehmer
- Teil der Projektdokumentation
- Frühzeitiges Erkennen von Fehlern in der Analyse- und Designphase eines Projektes
- Zeit- und Kostenersparnis



3

3

Einführung in UML



4

4

Einführung in UML

Häufig gewählte Reihenfolge der Verwendung

Analysephase

1. Anwendungsfalldiagramm (Use Case Diagram)

- Welche Anwendungsfälle in der zu erstellenden Anwendung enthalten sind
- Welche Akteure diese Anwendungsfälle auslösen
- Welche Abhängigkeiten der Anwendungsfälle untereinander bestehen, z. B.
 - Ob ein Anwendungsfall in einem anderen enthalten ist
 - Ob ein Anwendungsfall eine Spezialisierung eines anderen darstellt
 - Ob ein bestehender Anwendungsfall durch einen zweiten erweitert wird

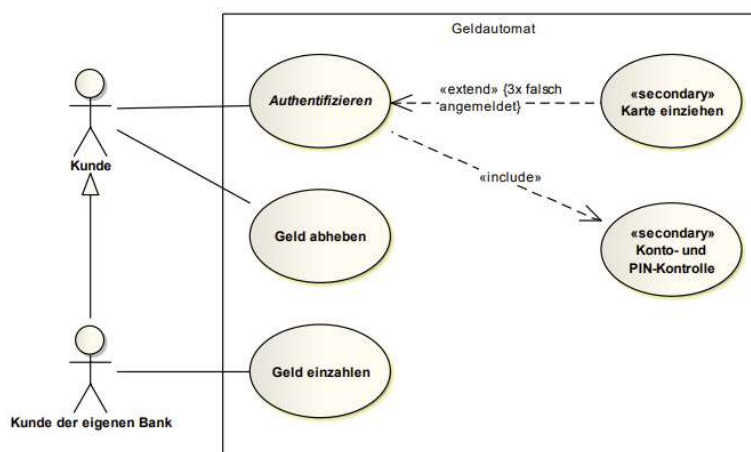


5

5

Einführung in UML

Beispiel Use Case Diagramm



6

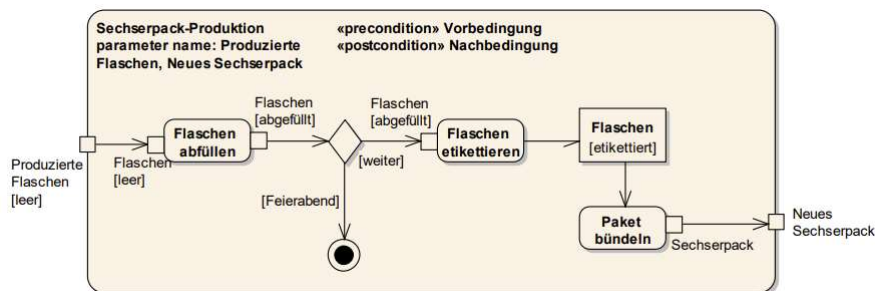
6

Einführung in UML

Analyse- und Designphase

2. Aktivitätsdiagramm (Activity Diagram)

- Welche Schritte innerhalb eines Anwendungsfalls durchlaufen werden
- Welche Zustandsübergänge die beteiligten Objekte erfahren, wenn die Abarbeitung von einer Aktivität zur nächsten wechselt



7

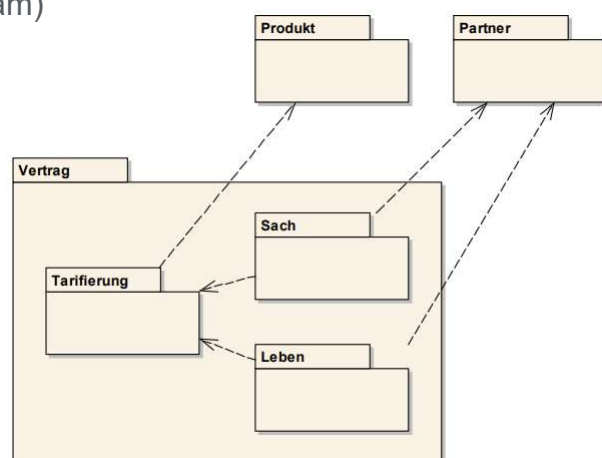
7

Einführung in UML

Analyse- und Designphase

3. Paketdiagramm (Package Diagram)

- In welche Pakete die Anwendung zerlegt werden kann
- Welche Pakete eine weitere Unterteilung ermöglichen
- Welche Kommunikation zwischen den Paketen realisiert werden muss



8

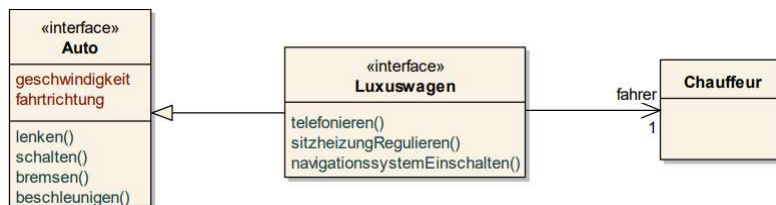
8

Einführung in UML

Analyse- und Designphase

4. Klassendiagramm (Class Diagram)

- Welche Zusammenhänge bestehen in der Aufgabenstellung (Domainmodell)
- Welche Klassen, Komponenten und Pakete beteiligt sind
- Über welche Kommunikation die Zusammenarbeit stattfindet
- Welche Methoden und Eigenschaften die Klassen benötigen
- Wie viele Objekte mindestens und höchstens in Verbindung stehen
- Welche Klassen als Container für mehrere Objekte zuständig sind



9

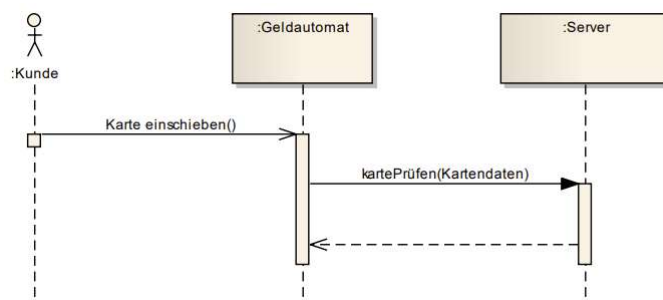
9

Einführung in UML

Designphase

5. Sequenzdiagramm (Sequence Diagram)

- Welche Methoden für die Kommunikation zwischen ausgewählten Objekten zuständig sind
- Wie der zeitliche Ablauf von Methodenaufrufen zwischen ausgewählten Objekten stattfindet
- Welche Objekte in einer Sequenz neu erstellt und / oder zerstört werden



10

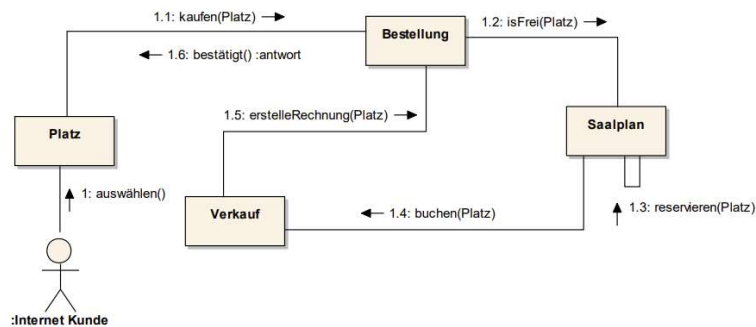
10

Einführung in UML

Designphase

6. Kommunikationsdiagramm (Communication Diagram)

- Wie ausgewählte Objekte miteinander kommunizieren
- In welcher Reihenfolge die Methodenaufufe stattfinden
- Welche alternativen Methodenaufufe gegebenenfalls existieren



11

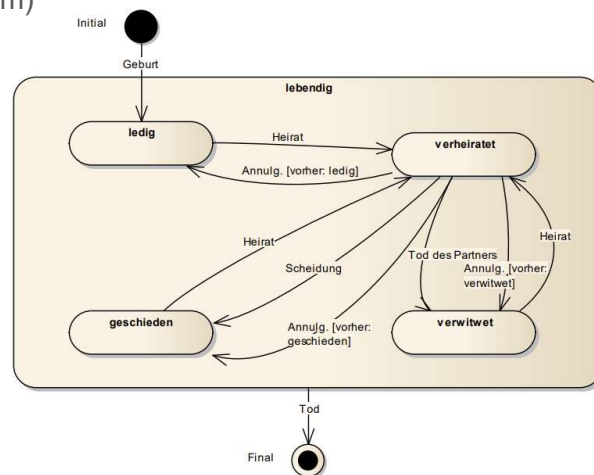
11

Einführung in UML

Designphase

7. Zustandsdiagramm (State Diagram)

- Welche Zustandsübergänge von welchen Methodenaufrufen ausgelöst werden
- Welcher Zustand nach dem Erzeugen des Objekts eingenommen wird
- Welche Methoden das Objekt zerstören



12

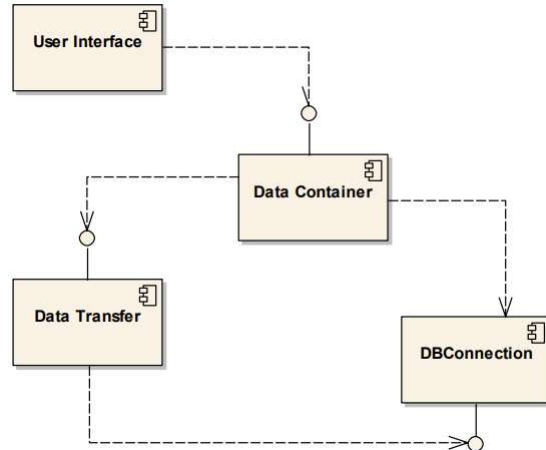
12

Einführung in UML

Designphase

8. Komponentendiagramm (Components Diagram)

- Wie werden Soft- und/oder Hardwareteile mit definierter Funktion und definierten Interfaces gekapselt
- Welche Komponenten haben Interfaces zueinander
- Welche Softwareteile erzeugen die Funktionalität in Komponenten



13

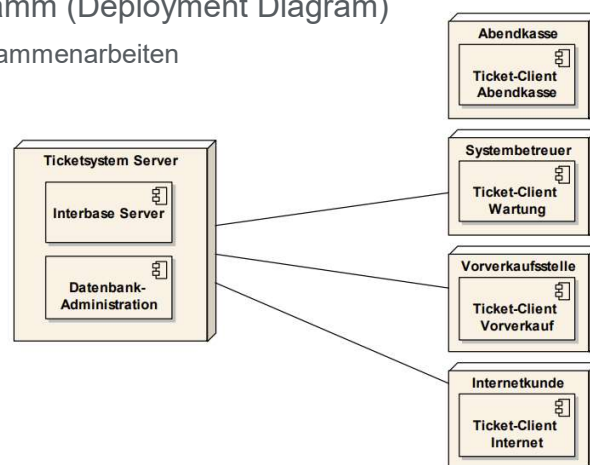
13

Einführung in UML

Designphase

9. Einsatz- und Verteilungsdiagramm (Deployment Diagram)

- Welche PCs in der Anwendung zusammenarbeiten
- Welche Module der Anwendung auf welchem PC ausgeführt werden
- Auf welchen Kommunikationsmöglichkeiten die Zusammenarbeit basiert



14

14

Das war der Überblick

- Sie haben 13 UML-Diagrammtypen kennen gelernt
- Wir durchlaufen im Anschluss die Analyse und Planung einer Software-Lösung an einem Fallbeispiel der Immobilienverwaltung
- Es werden die wichtigsten UML-Diagrammtypen detailliert betrachtet
 - Anwendungsfalldiagramm - Use Case
 - Aktivitätsdiagramm
 - Objektdiagramm
 - Klassediagramm
 - Sequenzdiagramm
 - Zustandsdiagramm



15

15

UML-Fallbeispiel

Die Firma **FairImmoPlus** soll durch ein Softwaresystem in die Lage versetzt werden, die Immobilien ihrer Kunden effizient rechnerunterstützt zu verwalten.

Die Software soll ohne viel Aufwand für die Zukunft änderbar sein, z. B. im Hinblick auf eine Änderung der Benutzungsoberfläche (z. B. vom Client weg zu einer webbasierten Lösung, oder von der Konsole weg hin zu einer GUI) oder im Hinblick auf eine Änderung der Datenhaltung (z. B. von einem relationalen Datenbanksystem weg hin zu einer anderen Speicherlösung , z. B. XML).

Das Produkt Immobilienverwaltung ist eine administrative Client-Anwendung.

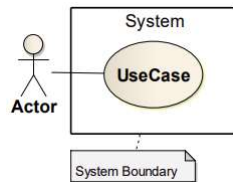


16

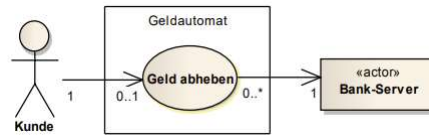
16

Anwendungsfalldiagramm (Use Case Diagram)

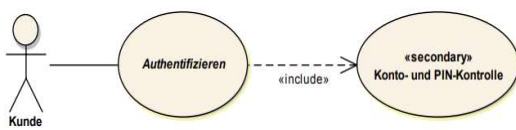
System (System Boundary)



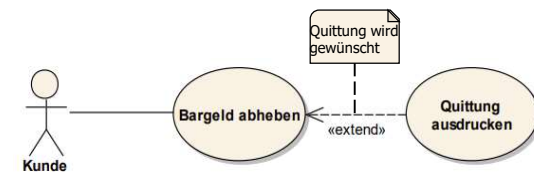
Beziehungen



Enthält-Beziehung (Include)



Erweiterungsbeziehung (extend)



17

17

Anwendungsfalldiagramm (Use Case Diagram)

Akteur



Anwendungsfall

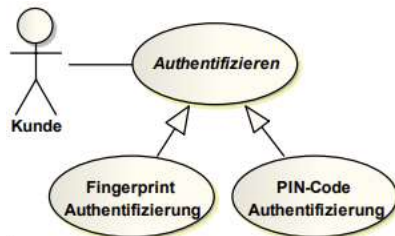


18

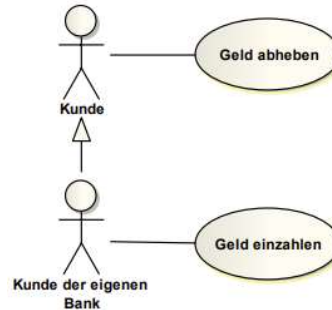
18

Anwendungsfalldiagramm (Use Case Diagram)

Generalisierung ...



... von Use Cases



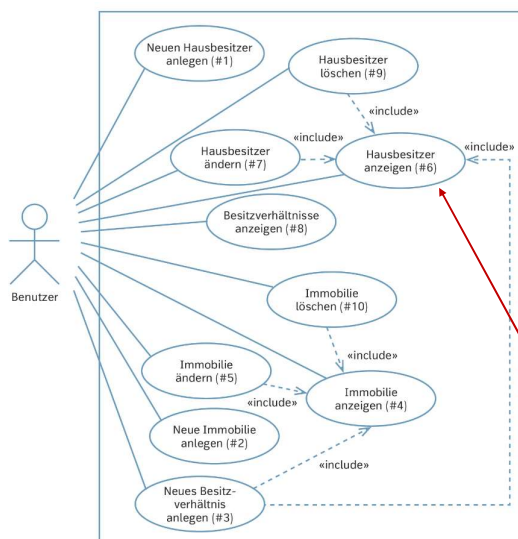
... von Akteuren



19

19

Use Case Diagramm - Fallbeispiel



System: Hausbesitzer-Verwaltung

<<include>>

Der Anwendungsfall, auf den der <<include>> Pfeil zeigt, wird in jedem Fall mit ausgeführt

Der Zeitpunkt ist nicht festgelegt, ob vor oder nach der Ausführung des inkludierenden Anwendungsfalls

Use Case #6

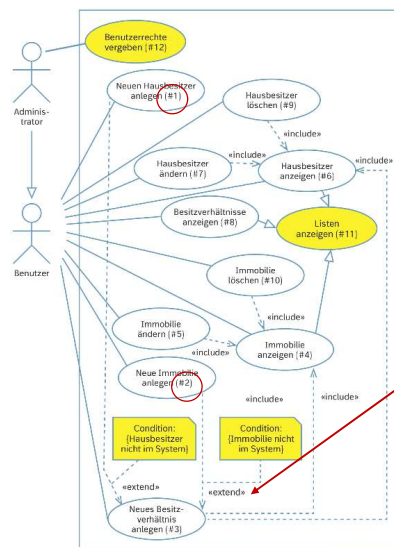
```
public static void hausbesitzerAendern(Hausbesitzer hb) {
    //...
    hausbesitzerAnzeigen(hb);
    //...
}
```



20

20

Use Case Diagramm - Fallbeispiel



<<extend>>

Unter bestimmten Bedingungen (Condition) wird ein Anwendungsfall um einen weiteren Anwendungsfall erweitert

Use Case #3

```
public static void neuesBesitzverhaeltnisAnlegen() {
    //...
    if (!istBesitzerImSystem)           // condition 1
        hb = neuenHausbesitzerAnlegen(); // extend: use case 1
    if (!istImmobilieImSystem)         // condition 2
        im = neueImmobilieAnlegen();    // extend: use case 2
    //...
}
```



21

21

Use Case Diagramm - Fallbeispiel

Textuelle Beschreibung eines Use Case

| Use Case #5 | Immobilie ändern |
|------------------------|--|
| Beschreibung/Ziel | Die Daten einer Immobilie werden geändert |
| Akteur | Benutzer (und Administrator) |
| Priorität | Wichtig |
| Status | In Arbeit |
| Auslöser | Immobilien ändern sich. Benutzer will Änderungen ins System einpflegen |
| Abhängigkeiten/include | Vorhandene Immobilien werden angezeigt (Use Case #4) und Benutzer wählt eine aus |
| Abhängigkeiten/extend | keine |
| Vorbedingung | Benutzer hat einen Account und Berechtigung. Immobilien sind vorhanden |
| Nachbedingung | Immobilien ändern sich. Andere Benutzer können aktualisierte Daten sehen |
| Hinweis | Je nachdem, ob die Immobilie ein Wohnhaus oder ein Geschäftshaus ist, müssen die Eingabefelder entsprechend gestaltet werden |



22

22

Kompetenzcheck

Finden Sie die richtigen Antworten:

1. Das Use Case Diagramm ...
 - [a] stellt Anwendungsfälle einer Software im zeitlichen Kontext dar.
 - [b] zeigt die Struktur von Elementen mit Methoden und Attributen.
 - [c] wird verwendet, um Anforderungen an eine Software zu definieren und zu kommunizieren.
2. Eine Person, die eine Software bedient,
 - [a] schlüpft in eine Rolle, die durch einen Akteur repräsentiert wird.
 - [b] wird im Use Case Diagramm namentlich genannt.
 - [c] steht in den Interaktionsdiagrammen im Mittelpunkt.
3. Ein Anwendungsfall wird repräsentiert durch
 - [a] ein Quadersymbol mit Namen oberhalb.
 - [b] eine Ellipse mit Namen innerhalb oder unterhalb.
 - [c] ein Rechteck mit einem Doppelpunkt vor dem Namen.



23

23

Kompetenzcheck

Finden Sie die richtigen Antworten:

4. Eine Include-Beziehung sagt, dass
 - [a] ein Use Case unbedingt in einem anderen vorkommen muss.
 - [b] ein Use Case möglicherweise in einem anderen vorkommen kann.
 - [c] ein Use Case eine Spezialisierung eines anderen darstellt.
5. Eine Extend-Beziehung sagt, dass
 - [a] ein Use Case unter Umständen mit einem anderen Use Case vorkommen kann
 - [b] ein Use Case immer zusammen mit einem anderen Use Case ausgeführt werden muss
 - [c] ein Use Case vom anderen abhängt und daher nicht erweitert werden darf



24

24

Kompetenzcheck

Aufgaben



- Finden Sie im Web Beispiele für Use Case Diagramme, aus denen Sie die Regeln gut nachvollziehen können.
- Präsentieren Sie ihr Lieblingsdiagramm und argumentieren Sie, warum Ihnen gerade dieses Diagramm so gut gefällt.



25

25

Aktivitätsdiagramm (activity diagram)

Modellieren Abläufe von Aktionen, enthalten oft algorithmische Strukturen



Systemgrenze bzw. Name des Aktivitätsdiagramms



Aktion



Schwimmbahnen (swimlanes), um Aktionen speziellen Bereichen zuzuordnen



Startknoten



Endknoten einer Aktivität



Endknoten eines Kontrollflusses



Referenz auf ein weiteres Aktivitätsdiagramm



Verzweigungsknoten (Fallunterscheidung)



Verbindungsknoten (Zusammenführung)



Parallelisierungsknoten (hier starten parallel stattfindende Aktionen)



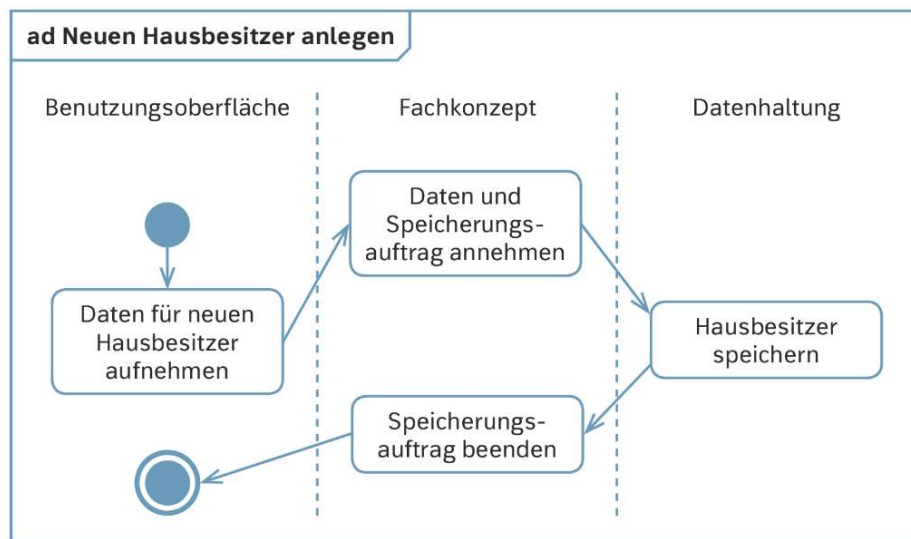
Synchronisationsknoten (hier enden parallel ausgeführte Aktionen)



26

26

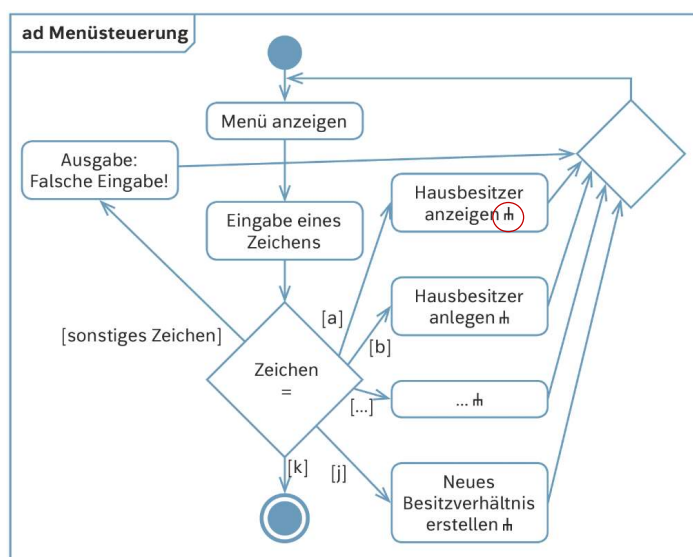
Aktivitätsdiagramm - Fallbeispiel



27


27

Aktivitätsdiagramm - Fallbeispiel



Menüsteuerung über textorientierte Benutzeroberfläche

[a] Menüpunkt „Hausbesitzer anzeigen“ bis
[k] Menüpunkt „Programm beenden“ und falsches Zeichen

Ist eine Aktion mit einem Gabelsymbol  gekennzeichnet, ist das der Verweis auf ein weiteres Aktivitätsdiagramm



28

28

Kompetenzcheck

Finden Sie die richtigen Antworten:

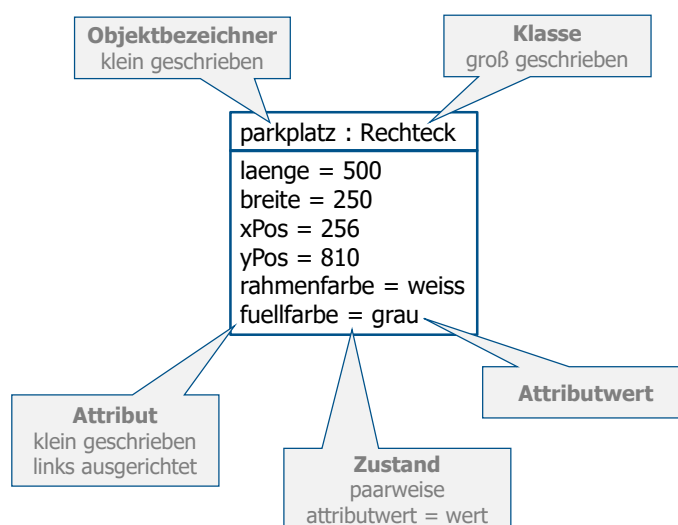
- Das Aktivitätsdiagramm
 - [a] zeigt zeitliche Abfolgen von Aktionen und Aktivitäten.
 - [b] hat sich seit UML 1.4 nicht wesentlich verändert.
 - [c] zeigt Aktionen und Aktivitäten in einer Größe relativ zu ihrer Dauer.
- Eine Aktivität
 - [a] kann hierarchisch verschachtelt weitere Aktivitäten beinhalten.
 - [b] ist ein nicht weiter unterteilbarer Schritt in einem Ablauf.
 - [c] stellt den Nachrichtenaustausch zwischen Objekten dar.
- Splitting und Synchronisation werden eingesetzt, um
 - [a] Wege aus einer Entscheidung miteinander zu vergleichen.
 - [b] die zeitliche Dauer von Aktionen zu betonen.
 - [c] Parallelität von Aktionen darzustellen.
- Swimlanes sind
 - [a] Verantwortlichkeitsbereiche, die nach ihrem zuständigen Akteur benannt werden.
 - [b] Verantwortlichkeitsbereiche, innerhalb derer Akteure dargestellt werden.
 - [c] der bedingte Übergang zwischen zwei Aktionen.



29

29

Objektdiagramm

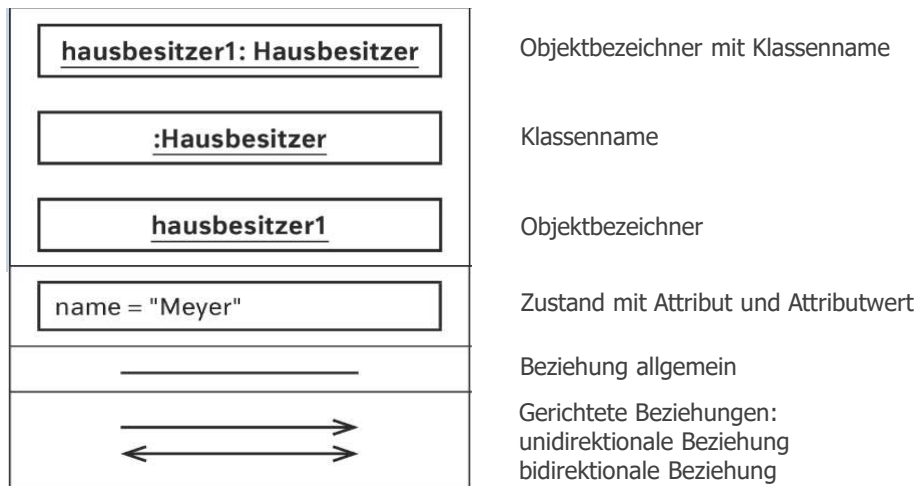


30

30

Objektdiagramm

Basiselemente im Objektdiagramm



31

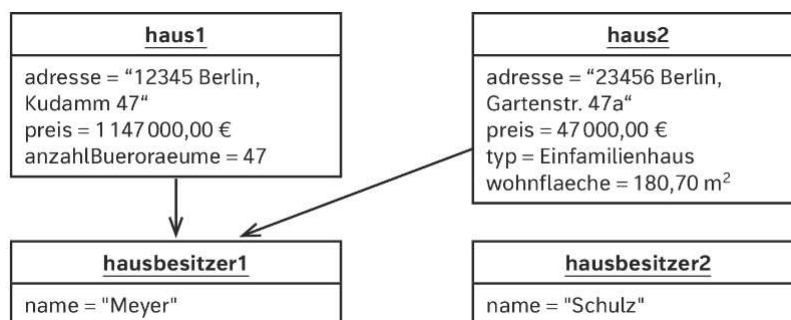
31

Objektdiagramm - Fallbeispiel

Brainstorming-Phase

hausbesitzer1 besitzt haus1 und haus2.

Was müsste im Entwurf noch verändert werden?



32

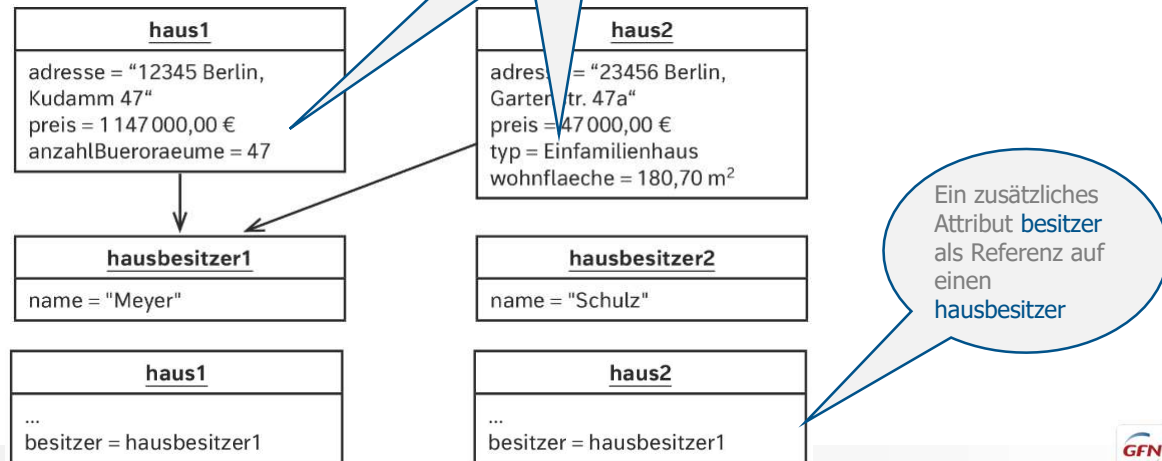
32

Objektdiagramm - Fallbeispiel

Brainstorming-Phase

hausbesitzer1 besitzt haus1 und haus2.

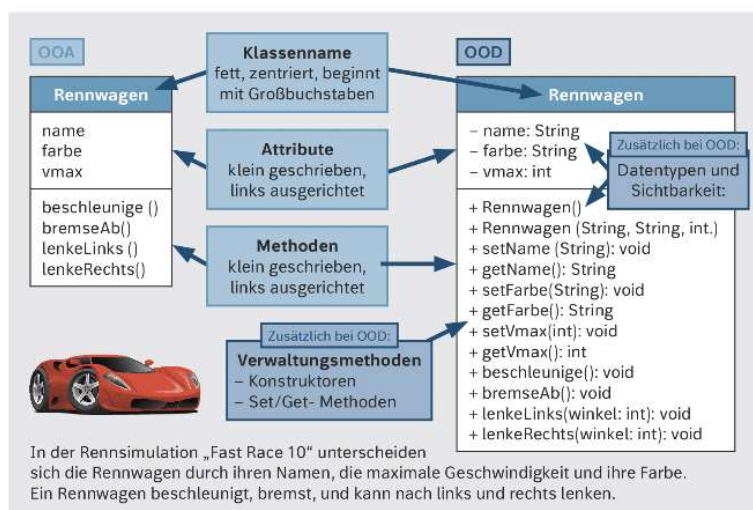
Was müsste im Entwurf noch verändert werden?



33

33

Klassendiagramm – class diagram



Klassendiagramm in OOA- und OOD-Ansicht (mit Java-Bezug)



34

34

Klassendiagramm

| Element | Bedeutung |
|--|---|
| Hausbesitzer | Klasse (Name) |
| – name: String – immobilien: ArrayList | Klasse (Attribute) OOD-Ansicht: mit Zugriffsspezifizierern und Datentypen |
| + Hausbesitzer + Hausbesitzer(String) + setName(String): void + getName(): String + neueImmoblie(int): void + löscheImmoblie(int): void | Klasse (Konstruktoren, Verwaltungsmethoden, Methoden) OOD-Ansicht: mit Zugriffsspezifizierern, Rückgabewerten, Parametern und Datentypen |
| Immoblie | Abstrakte Klasse (keine Instanziierung von Objekten möglich); kann statt kursiver Schreibweise auch durch {abstract} gekennzeichnet werden |



35

35

Klassendiagramm

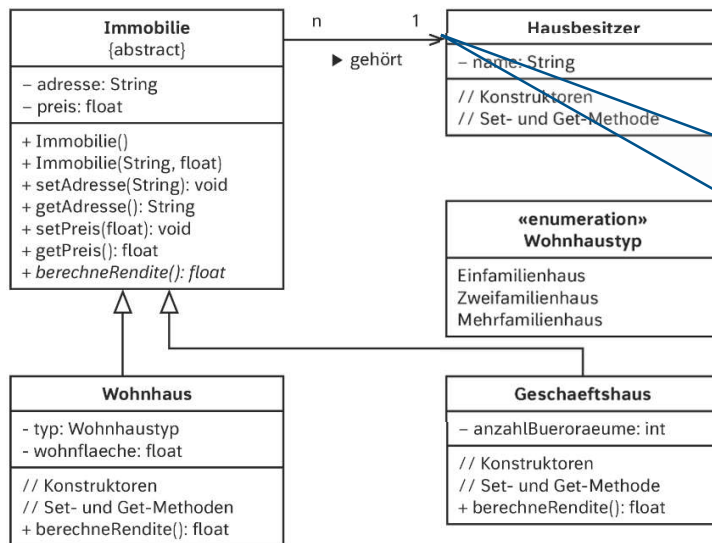
| Element | Bedeutung |
|--|---|
| – anzahlImmobilien: int | Klassenattribut (statisches Attribut) |
| + getAnzahlImmobilien(): int + berechneRendite(): float | Klassenmethode (statische Methode), Aufruf mit Immobilie.getAnzahlImmobilien() Abstrakte Methode (muss in der Unterklasse implementiert werden); kann statt kursiver Schreibweise auch durch {abstract} gekennzeichnet werden |
| — | Beziehung (allgemein) |
| n — 1 ► gehört | Gerichtete Beziehung inkl. Multiplizität und Assoziationsnamen |
| | Aggregation: schwache „Ist-Teil-Von-Beziehung“ Wird das Gesamtojekt gelöscht, können die Teilobjekte weiter existieren. |
| | Komposition: starke „Ist-Teil-Von-Beziehung“ Wird das Gesamtojekt gelöscht, werden auch die Teilobjekte gelöscht. |
| —> | Vererbung (zwischen Klassen) |
| -----> | Implementierung einer Schnittstelle |



36

36

Klassendiagramm - Fallbeispiel



Man beachte die Modellierungsentscheidungen, dass eine Immobilie nur genau einen Besitzer hat und dass eine Immobilie ihren Besitzer „kennt“, aber ein Besitzer nicht seine Immobilien

Klassendiagramm in OOD-Ansicht

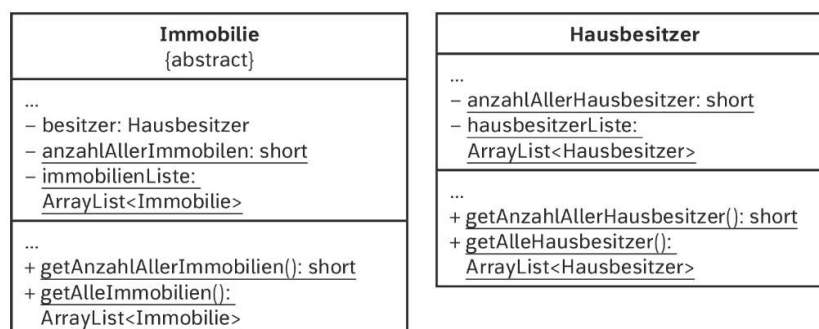


37

37

Klassendiagramm - Fallbeispiel

Nächste Modellierungsrunde



Hinweis: Unterstrichene Attribute und Methoden sind Klassenattribute bzw. Klassenmethoden

```
ArrayList<Immobilie> immobilienListe;
immobilienListe = Immobilie.getAllImmobilien();
```



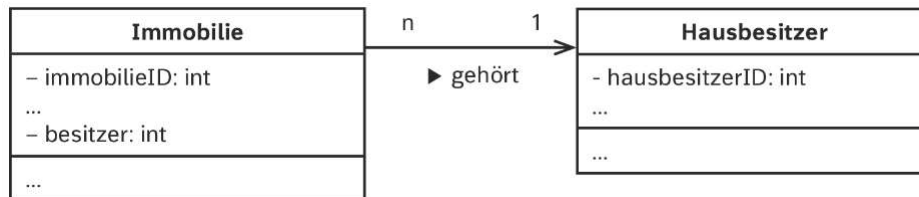
38

38

Klassendiagramm - Fallbeispiel

Vorbereitung auf den Einsatz einer relationalen Datenbank

Spätestens, wenn man mit einem relationalen Datenbanksystem arbeitet, bekommen die Immobilien und die Hausbesitzer jeweils einen künstlichen Primärschlüssel, der sich auch in den Klassen als Attribut wiederfindet



Das Attribut **besitzer** der Klasse **Immobilie** mit dem ursprünglichen Datentyp **Hausbesitzer** erhält jetzt den Datentyp **int** und bekommt den Wert von **hausbesitzerID** zugewiesen

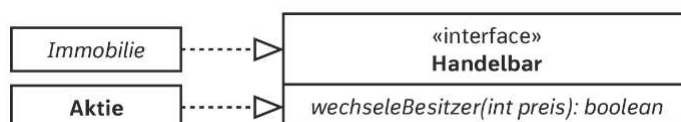


39

39

Klassendiagramm - Fallbeispiel

Interface



Klassen können auch Schnittstellen (Interfaces) implementieren, quasi beerben.

Interfaces enthalten Methoden-Deklarationen, die in der erbenden Klasse einen Methodenkörper erhalten müssen (im Beispiel die Klasse **Aktie**), oder die erbende Klasse muss selbst wieder abstrakt sein (im Beispiel die Klasse **Immobilie**, kursiv geschrieben)

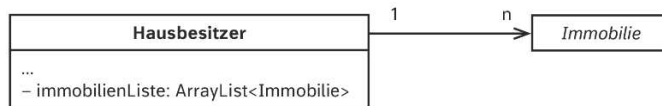


40

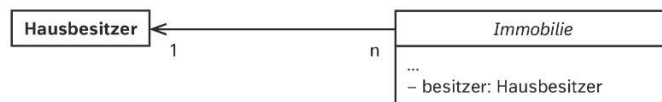
40

Klassendiagramm - Fallbeispiel

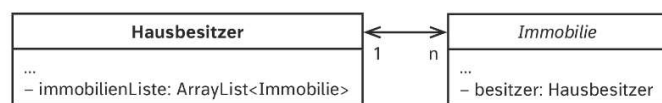
Assoziationen



Die Hausbesitzer kennen ihre Immobilien, aber die Immobilien haben keine Referenz auf ihre Besitzer. Trotzdem kann man algorithmisch den Besitzer einer Immobilie herausfinden



Die Immobilien kennen ihre Besitzer, die Besitzer kennen aber ihre Immobilien nicht. Trotzdem kann man algorithmisch die Immobilien eines Besitzers herausfinden



Beide Seiten kennen sich. Hausbesitzer hat alle Referenzen auf seine Immobilien in einer Liste. Immobilie hat die Referenz auf ihren Hausbesitzer



41

41

Klassendiagramm - Fallbeispiel

Assoziationen

○ Aggregation



Aggregation wird auch bezeichnet als schwache **Ist-Teil-Von-Beziehung**.

Wenn das Gesamtobjekt **Raum** gelöscht wird, existieren die Teilobjekte **Inventar** weiter.

○ Komposition



Komposition wird auch bezeichnet als starke **Ist-Teil-Von-Beziehung**.

Wenn das Gesamtobjekt **Immobilie** gelöscht wird, werden auch die Teilobjekte **Raum** gelöscht.



42

42

Kompetenzcheck

Finden Sie die richtigen Antworten:

1. Eine Klasse wird in der UML dargestellt durch
 - [a] ein Rechteck mit drei Bereichen: Klassenname, Methoden, Zustände.
 - [b] ein dreifärbiges Rechteck mit Klassenname, Parametern, Methoden.
 - [c] ein Rechteck mit drei Bereichen: Klassenname, Attribute, Methoden.
2. Das einem Attribut vorangestellte Symbol
 - [a] „#“ kennzeichnet Attribute, die als Properties nach außen sichtbar sind.
 - [b] „+“ bedeutet generelle Sichtbarkeit (auch von außerhalb der Klasse).
 - [c] „-“ kennzeichnet Attribute mit einfachen Datentypen.
3. Ein Domainmodell ist
 - [a] die Darstellung der Wirklichkeit in der Analysephase.
 - [b] ein Modell basierend auf Attributen und Methoden von Klassen.
 - [c] eine Basis für die Erzeugung von Source Code.
4. In einer Komposition
 - [a] sind die Klassen als lose Teile miteinander verbunden.
 - [b] ist eine Klasse ein untrennbarer Bestandteil der anderen.
 - [c] erbt eine Klasse Attribute und Methoden von der anderen.
5. Bei der Generalisierung
 - [a] zeigt der Pfeil der Verbindung auf die spezielle Klasse.
 - [b] erbt die spezielle Klasse Methoden und Attribute der allgemeinen Klasse.
 - [c] werden gleichwertige Attribute zwischen Klassen übergeben und gefüllt.



43

43

Sequenzdiagramm

Symbole

| Element | Bedeutung |
|--------------------------------------|---|
| <u>einHausbesitzer</u> | Objekt (verschiedene Darstellungsmöglichkeiten) Hinweis: Manche UML-Zeichentools lassen den Unterstrich weg. |
| <u>einHausbesitzer: Hausbesitzer</u> | |
| <u>:Hausbesitzer</u> | |
| → | Synchroner Methodenaufruf (Nachricht) |
| ---> | Objekterzeugung |
| <--- | Antwortnachricht (Rücksprung) |



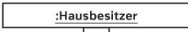
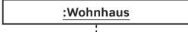
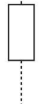


44

44

Sequenzdiagramm

Symbole

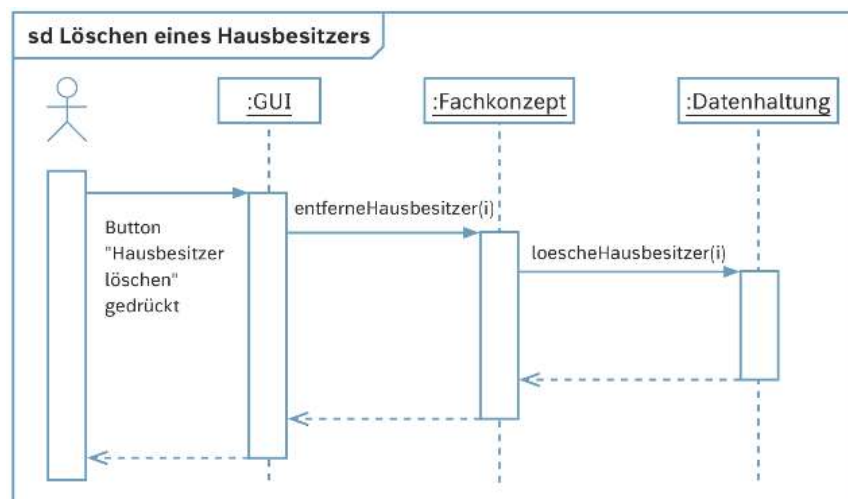
| Element | Bedeutung |
|---|---|
|  | Der (immer aktive) Akteur |
|  | Aktives Objekt |
|  | Konstruktor |
|  | Lebenslinie |
|  | Aktivierung der Lebenslinie (Methodenabarbeitung bzw. Ausführungssequenz) |



45

45

Sequenzdiagramm - Fallbeispiel

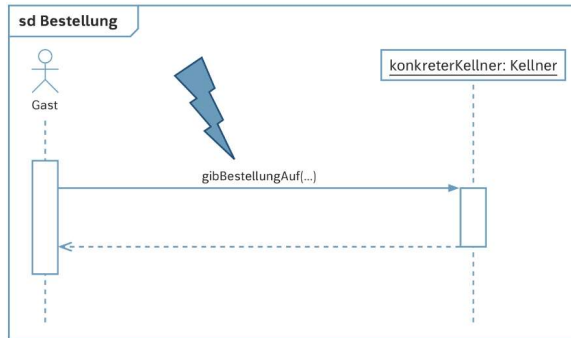


46

46

Sequenzdiagramm

Falsch modelliertes Sequenzdiagramm



Der Methodenaufruf `gibBestellungAuf(...)` mag zwar im ersten Moment als logisch erscheinen, ist aber falsch. Die Methode muss zwingend eine Methode der Klasse Kellner sein. Der spätere falsche Quelltext würde nämlich wie folgt aussehen:

```
konkreterKellner.gibBestellungAuf(...); // falsch!
```

Ein Kellner gibt aber keine Bestellungen auf, sondern nimmt Bestellungen an, somit muss ein richtig modellierter Methodenaufruf wie folgt lauten:

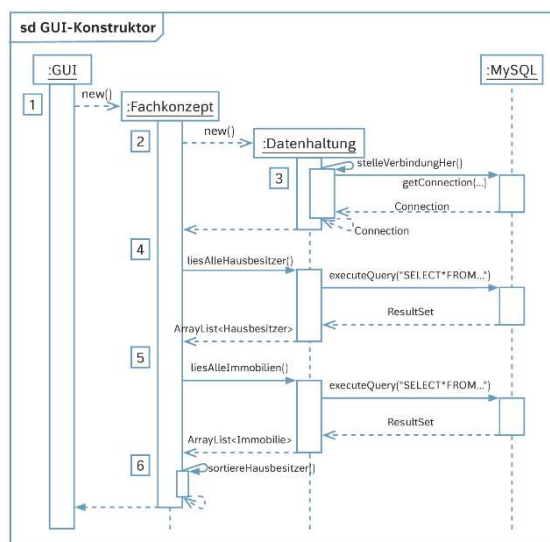
```
konkreterKellner.nimmBestellungAn(...); // richtig!
```



47

47

Sequenzdiagramm – Fallbeispiel – 3 Schichten Architektur



Das GUI-Objekt braucht eine Referenz auf das Fachkonzept-Objekt. Daher wird im GUI-Konstruktor einfach das Fachkonzept-Objekt erzeugt (Position 1)

Das Fachkonzept-Objekt braucht eine Referenz auf das Datenhaltungs-Objekt. Daher wird im Fachkonzept-Konstruktor einfach das Datenhaltungs-Objekt erzeugt (Position 2)

Zusätzlich liest der Fachkonzept-Konstruktor alle Hausbesitzer und alle Immobilien aus dem Datenhaltungs-Objekt und sortiert dabei die Hausbesitzerliste nach Namen (Positionen 4, 5 und 6)

Der Datenhaltungs-Konstruktor muss eine Verbindung zur konkreten Datenbank herstellen (Position 3)

Aus diesem Sequenzdiagramm lassen sich leicht die konkreten Implementierungen ableiten



48

48

Implementierung aus Sequenzdiagramm - Fallbeispiel

```
// Konstruktor in der Klasse GUI
public GUI() {

    einFachkonzept = new Fachkonzept();

}

// Konstruktor in der Klasse Fachkonzept
public Fachkonzept() {
    // s. Abb. 3.18: Position 2
    eineDatenhaltung = new Datenhaltung();
    // s. Abb. 3.18: Position 4
    hausbesitzerListe = eineDatenhaltung.liesAlleHausbesitzer();
    // s. Abb. 3.18: Position 5
    immobilienListe = eineDatenhaltung.liesAlleImmobilien();
    // s. Abb. 3.18: Position 6
    sortiereHausbesitzer();
}
```

```
// Klasse Datenhaltung
//
public class Datenhaltung() {

    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    // Konstruktor
    public Datenhaltung() {
        con = stelleVerbindungHer();
    }

    // Methode stelleVerbindungHer()
    public Connection stelleVerbindungHer() {
        // ...
        Class.forName("com.mysql.jdbc.Driver");
        return DriverManager.getConnection("DB_URL", "USER", "PASSWORD");
    }

    // Methode liesAlleHausbesitzer()
    public ArrayList<Hausbesitzer> liesAlleHausbesitzer() {
        ArrayList<Hausbesitzer> hausbesitzerListe = new ArrayList();
        stmt = con.createStatement();
        rs = stmt.executeQuery("SELECT * FROM T_Hausbesitzer");
        // Umwandlung des ResultSet in die ArrayList
        // ...
        return hausbesitzerListe;
    }

    // Methode liesAlleImmobilien
    // ...
} // Klasse Datenhaltung
```

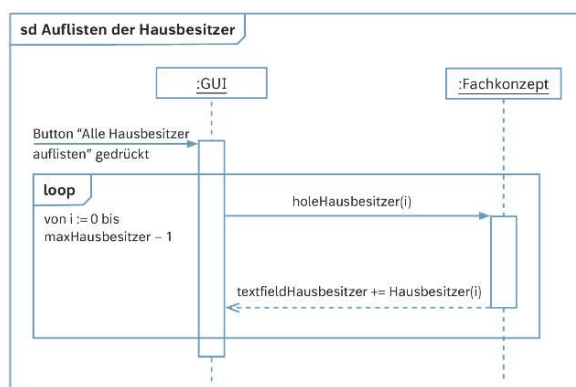


49

49

Sequenzdiagramm - Fallbeispiel

Kontrollstruktur Loop



```
// Methode holeHausbesitzer in der Klasse Fachkonzept
public Hausbesitzer holeHausbesitzer(int stelle) {
    return hausbesitzerListe.get(stelle);
}

// Methode listeHausbesitzerAuf in der Klasse GUI
public void listeHausbesitzerAuf() {
    for (int i = 0; i < einFachkonzept.getMaxAnzahlHausbesitzer(); i++) {
        textfieldHausbesitzer.add(einFachkonzept.holeHausbesitzer(i));
    }
}
```

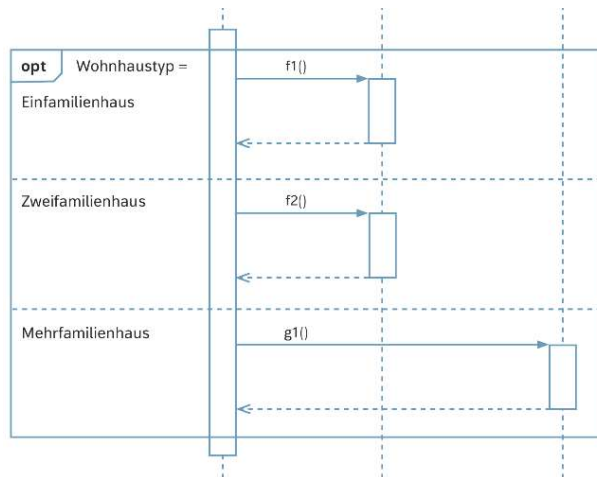


50

50

Sequenzdiagramm - Fallbeispiel

Kontrollstruktur Fallunterscheidung - switch



Gekennzeichnet als **opt**-Fragment
(Option: Fallunterscheidung)
Abhängig vom Wohnhaustyp werden
unterschiedliche Methoden aufgerufen

Es gibt noch weitere Fragmente z. B.:
alt, assert, break, consider, critical, ignore,
loop, neg, opt, par, ref, seq, strict

Quelle:
de.wikipedia.org/wiki/Sequenzdiagramm



51

51

Kompetenzcheck

Finden Sie die richtigen Antworten:

1. Neue Objekte im Sequence Diagram werden
 - [a] nur am oberen Blattrand modelliert, sie leben während der gesamten Interaktion.
 - [b] an der Position des ersten Methodenaufrufs modelliert.
 - [c] mit Hilfe der „lost message“ modelliert.
2. Eine Nachricht, bei der der Aufrufer nicht auf einen Rückgabewert wartet,
 - [a] heißt „asynchrone“ Nachricht.
 - [b] wird im UML-Diagramm nicht modelliert.
 - [c] wird durch ein „X“ auf der Lebenslinie des Objekts markiert.



52

52

Zustandsdiagramm (state diagram)

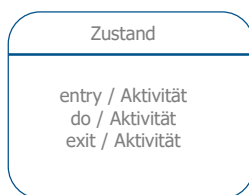
Modellieren Abläufe von Aktionen, enthalten oft algorithmische Strukturen



Systemgrenze des Zustands-Diagramm z. B. **stm Mieter**



Zustand z. B.
„Mieter hat keine Mietschulden“



Startknoten

Endknoten

Ein **Zustand** (z. B. „Mieter hat Mietschulden“) kann durch eine waagrechte Linie unterteilt werden. Unterhalb der Linie kann das Verhalten eines Objekts in dem Zustand angegeben werden.

Beispiel :

entry / Schreiben an Mieter: „Haben Sie evtl. vergessen, die Miete zu bezahlen?“

do / nothing

exit / Schreiben an Mieter: „Danke für die Überweisung!“

Die Beschreibungselemente sind optional.

Die Beschriftung am **Transition-Pfeil** gibt das auslösende Ereignis (**trigger**) an, die Bedingung, die erfüllt sein muss (**guard**), und die **Aktivität**, die beim Übergang ausgeführt wird.

Beispiel:

Kontoüberprüfung [Miete nicht eingegangen] / **Schreiben an Mieter**

Die Beschreibungselemente sind optional.

Trigger [guard] / Aktivität

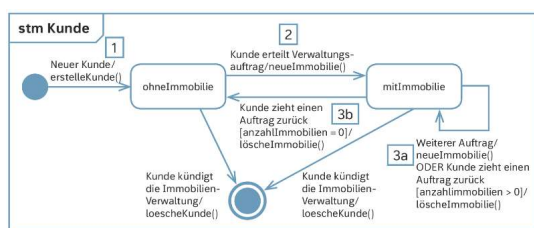
Transition (Zustandsübergang von einem Quellzustand zu einem Zielzustand)



53

53

Zustandsdiagramm - Fallbeispiel



```
// Die Methode erstelleKunde() findet sich in der Klasse
// TUI bzw. GUI
public void erstelleKunde() {
    tmpHausbesitzer.setName(...);
    tmpHausbesitzer.setZustand(Kundenstatus.ohneImmobilie);
    // Hinweis: hausbesitzerNr wird automatisiert vom Fachkonzept
    // vergeben
    einFachkonzept.speichereNeuenHausbesitzer(tmpHausbesitzer);
}
```

```
// Methode realisiert indirekt einen Verwaltungsauftrag.
// Dem Kunden, der die neue Immobilie besitzt, wird, unabhängig
// von seinem vorherigen Zustand, der Kundenstatus "mitImmobilie"
// zugewiesen
public void neueImmobilie(Immobilie eineImmobilie) {
    int tmpHausbesitzerNr;
    Hausbesitzer tmpHausbesitzer;
    tmpHausbesitzerNr = eineImmobilie.getHausbesitzerNr();
    if (gibtEsHausbesitzer(tmpHausbesitzerNr)) {
        immobilienListe.add(eineImmobilie);
        int i = 0;
        for (Hausbesitzer hb: hausbesitzerListe) {
            if (hb.getHausbesitzerNr() == tmpHausbesitzerNr) {
                tmpHausbesitzer = hb;
                mitgliederListe.remove(i);
                tmpHausbesitzer.setZustand(Kundenstatus.mitImmobilie);
                mitgliederListe.add(i, tmpHausbesitzer);
                return;
            } // if
            i++;
        } // for
    } // if
} // neueImmobilie

// Methode realisiert indirekt die Zurücknahme eines
// Verwaltungsauftrags. Allerdings ändert sich der Kundenstatus
// nur, wenn es die letzte Immobilie des Kunden war.
public void loescheImmobilie(int immobilienNr) {
    // ...
    // Wenn es die Immobilie gibt, dann ...
    // ... Lies Hausbesitzer-Nummer aus der Immobilie aus.
    // Gehe alle Immobilien durch, ob der Hausbesitzer noch in
    // einer anderen Immobilie eingetragen ist.
    // Wenn ja, bleibt der Status (s. Abb. 3.22: Position 3a).
    // Wenn nicht:
    tmpHausbesitzer = hb;
    mitgliederListe.remove(i);
    // s. Abb. 3.22: Position 3b
    tmpHausbesitzer.setZustand(Kundenstatus.ohneImmobilie);
    mitgliederListe.add(i, tmpHausbesitzer);
    // Lösche die Immobilie aus der Immobilien-Liste
} // loescheImmobilie
```



54

54

Kompetenzcheck

Finden Sie die richtigen Antworten:

1. Ein Zustand ohne notierten Namen ist
 - [a] nicht UML-konform.
 - [b] ein unvollständiger Zustand.
 - [c] ein „anonymer“ Zustand.

2. Der Startpunkt des State Diagram
 - [a] ist das erste eintreffende Ereignis
 - [b] definiert einen Guard für den ersten Zustandswechsel.
 - [c] ist gleichbedeutend mit der Objekterzeugung.



55