Woche 06

ISW-Tutorium

Xel Pratscher

Orga

! Stellt Fragen !

- We all want to see you succeed
- Im Testat ist zu spät
- Ihr erreicht mich meistens über Discord
  - Wenn ich nach 6h nicht geantwortet habe schreibt nochmal
  - Auf dem Sever antworten auch andere Tutor:innen

Anwesenheit Testate

- Wenn ihr nicht kommen könnt - auch kurzfristig: **Sagt Ab!**
- Wer unentschuldigt fehlt oder mehr als 10 Minuten zu spät kommt besteht das Testat nicht

Tutorium

- Umfrage auf Discord
- Vorlage für Pairprogramming-Protokoll von Nemo

Infos aus VL

- ab **19.12.** Winterpause ISW
- **Blockprojekt freiwillig**
  - 19.02. - 01.03.
  - 2 CP FÜK
  - Erweiterung App in 4er-Teams
  - Teams $\neq$ Testatpartner
  - Umfrage in Moodle **ausfüllen**
- **1. Klausur** voraussichtlich **11.03.**

Vorlesung

1. Kommunkation von Entscheidungen (Rationale)
2. Modellierung
3. Interaktionsdiagramme
4. Klassendiagramme
5. Klassenentwurf mit OOAD

Kommunikation von Entscheidungen (Rationale)

- Begründung für Gestaltungsentscheidungen
- Wenn fehlt:
  - Entscheidungen berücksichtigen nicht alles
  - Entscheidungen **nicht überzeugend**
  - Entscheidung nachträglich umgeworfen
  - Verworfene Optionen **immer wieder** durchgegangen

Rationale Beschreibung I

- Fragen
  - Konkrete Probleme
- Optionen
  - Alternativen zur Lösung von Problem

Rationale Beschreibung II

- Kriterien
  - Qualitätsanforderungen
- Argumente
  - Kondensieren Diskussionen
- Entscheidungen
  - Bezug auf eine oder mehrere offene Fragen
  - Fasst gewählte Option und Argumente zusammen
  - Danach Frage “geschlossen”
  - Kann revidiert werden

Optionen zur Erfassung Rationale

- **implizit**
  - Gesprächsnotizen, Protokolle, …
  - versteckt
- **später**
  - nach Entwicklung
  - enthält keine alternativen Optionen
- **kontinuierlich**
  - während Entwicklung
  - Überarbeitung später
- **integriert**
  - Überarbeitung während Entwicklung

Nachteile Rationale

- Hoher Aufwand, Nutzen nicht sofort sichtbar
  - muss gut motiviert werden
- Aufwand zur Konsolidierung noch höher
  - mehr Arbeitsaufwand, eventuell eigene Rolle
- freie Erfassung nicht ausreichend
- Information, Nutzung komplex

Vorteile Rationale

- Unterstützt Teamwork
- Wiederverwendung / Änderungen leichter
- Höhere Endqualität
- Unterstützt Wissenstransfer

Modellierung

- **Modell:** in Maßstab, Detailliertheit und/oder Funktionalität verkürzte/abstrahierte Darstellung
  - Abstraktion eines Systems
  - **Ziel:** Nachdenken über System vereinfachen
  - Charakterisierung durch:
    - Abbildungsmerkmal
    - Verkürzungsmerkmal
    - Pragmatisches Merkmal

Strukturdiagramme

- **Entwurf**
  - **Klassendiagramm**
  - Objektdiagramm
  - Paketdiagramm
- **Architektur**
  - Kompositionsstrukturdiagramm
  - Komponentendiagramm
  - Verteilungsdiagramm

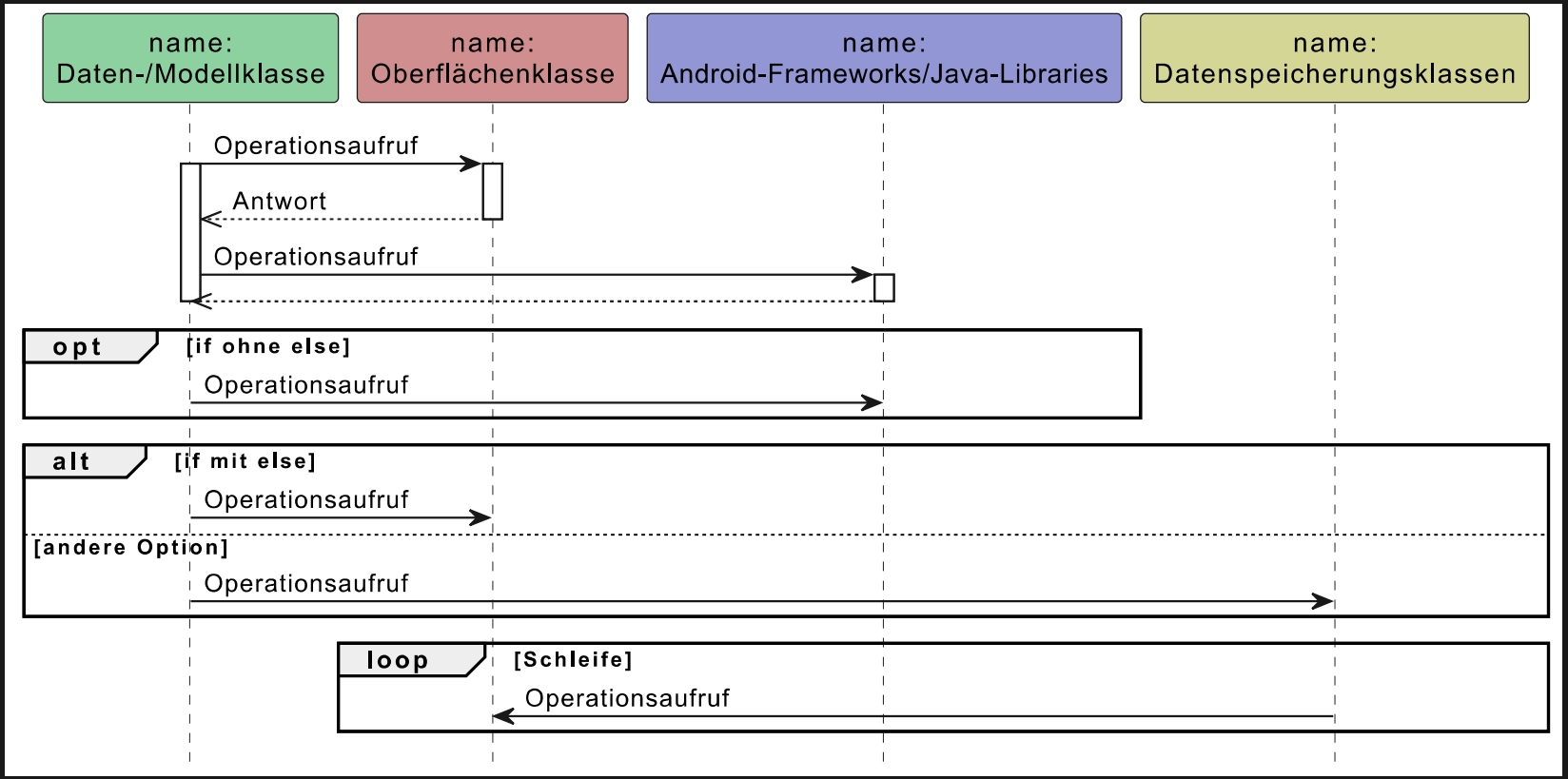
Verhaltensdiagramme

- **Abläufe**
  - Use Case Diagramm
  - Aktivitätsdiagramm
  - **Zustandsdiagramm**
- **Interaktion**
  - **Sequenzdiagramm**
  - Kommunikationsdiagramm
  - Zeitdiagramm
  - Interaktionsübersichtsdiagramm

Interaktionsdiagramme

- Kommunikation zw. versch. Aktoren beschreiben
- Sequenzdiagramm wichtig
- Folgt den Funktionen

Aufbau Interaktionsdiagramm



Beispiel

```
1  public class Sleep {
2      private void selectBedVersion(int selectedIndex,
3                                    List<Bed> beds) {
4          int id = 0;
5          for (int i = 0; i < beds.size(); i++) {
6              if (i != selectedBedVersion) {
7                  beds.get(i).setInactive();
8              }
9              else {
10                 beds.get(i).setActive();
11 }}}}
```

Beispiel

```
1  public class Sleep {
2      private void selectBedVersion(int selectedIndex,
3                                    List<Bed> beds) {
4          int id = 0;
5          for (int i = 0; i < beds.size(); i++) {
6              if (i != selectedBedVersion) {
7                  beds.get(i).setInactive();
8              }
9              else {
10                 beds.get(i).setActive();
11 }}}}
```

Beispiel

```
1  public class Sleep {
2      private void selectBedVersion(int selectedIndex,
3                                    List<Bed> beds) {
4          int id = 0;
5          for (int i = 0; i < beds.size(); i++) {
6              if (i != selectedBedVersion) {
7                  beds.get(i).setInactive();
8              }
9              else {
10                 beds.get(i).setActive();
11 }}}}
```

Beispiel

```
1  public class Sleep {
2      private void selectBedVersion(int selectedIndex,
3                                    List<Bed> beds) {
4          int id = 0;
5          for (int i = 0; i < beds.size(); i++) {
6              if (i != selectedBedVersion) {
7                  beds.get(i).setInactive();
8              }
9              else {
10                 beds.get(i).setActive();
11 }}}}
```

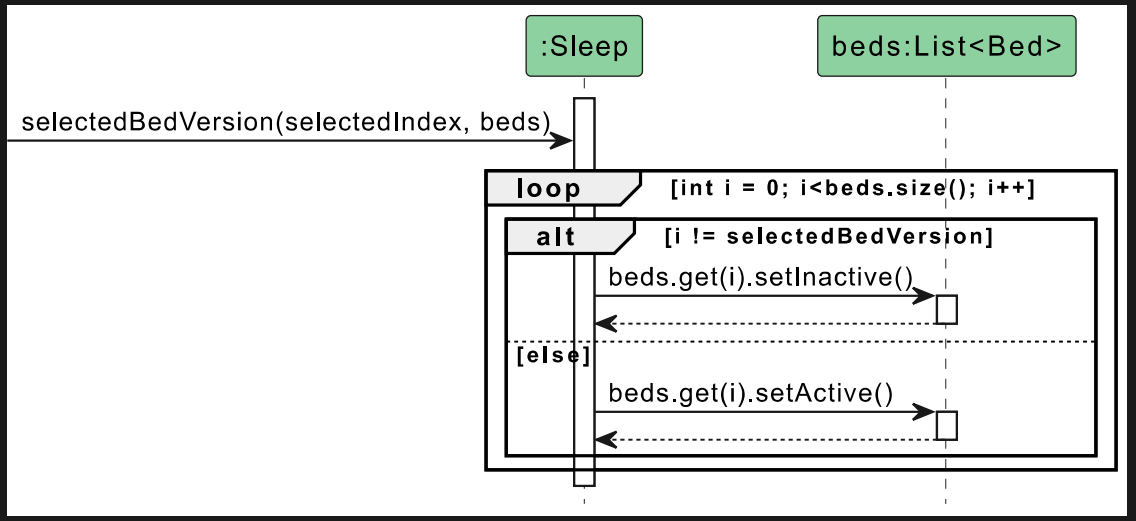
Beispiel

```
1  public class Sleep {
2      private void selectBedVersion(int selectedIndex,
3                                    List<Bed> beds) {
4          int id = 0;
5          for (int i = 0; i < beds.size(); i++) {
6              if (i != selectedBedVersion) {
7                  beds.get(i).setInactive();
8              }
9              else {
10                 beds.get(i).setActive();
11 }}}}
```

Beispiel

```
1  public class Sleep {
2      private void selectBedVersion(int selectedIndex,
3                                    List<Bed> beds) {
4          int id = 0;
5          for (int i = 0; i < beds.size(); i++) {
6              if (i != selectedBedVersion) {
7                  beds.get(i).setInactive();
8              }
9              else {
10                 beds.get(i).setActive();
11 }}}}
```

Lösung


:Sleep
beds:List<Bed>
selectedBedVersion(selectedIndex, beds)
loop
[int i = 0; i<beds.size(); i++]
alt
[i != selectedBedVersion]
beds.get(i).setInactive()
[else]
beds.get(i).setActive()

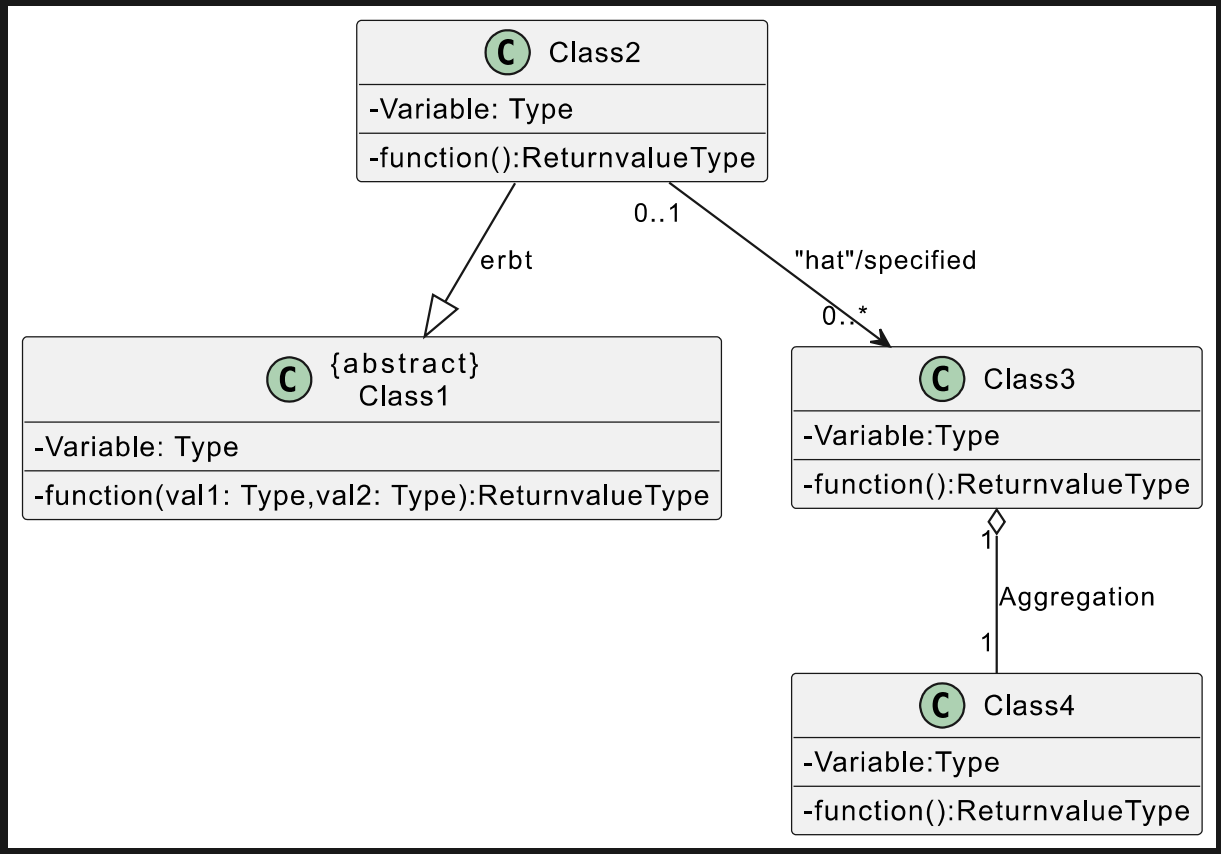
Klassendiagramme

- Komplexe Strukturen von Objekten beschreiben
- Typisierung von Objekten
- Mögliche Strukturen charakterisieren

Struktur

- Klassen (Objekte)
- Assoziationen (Aggregation, Komposition)
- Attribute
- Operationen
- Generalisierungsbeziehungen (Vererbung)
- Schnittstellen

Aufbau


Class2
-Variable: Type
-function():ReturnvalueType
0..1
erbt
"hat"/specified
0..*
{abstract}
Class1
-Variable: Type
-function(val1: Type,val2: Type):ReturnvalueType
Class3
-Variable:Type
-function():ReturnvalueType
1
Aggregation
1
Class4
-Variable:Type
-function():ReturnvalueType

Klassenentwurf mit OOAD

- Object-Oriented Analysis and Design
- Analyseklassenmodell
  - definiert Klassenstruktur auf Basis Anforderungen
  - abstrakte Modellierung Systemverhaltens
- Entwurfsklassenmodell
  - berücksichtigt vorhandene Klassen
  - optimiert auf Entwurfsziele

Entwurf Ziele

- Gliederung System in **überschaubare Einheiten**
  - Wiederverwendung vorhandener Komponente
- Festlegen der **Lösungsstruktur**
  - selbststabilisierend, lange haltbar
- **Hierarchische Gliederung**
  - Möglichkeit zur Abstraktion durch sinnvolle Gliederung

Umsetzung

- Klassen für **Datenverwaltung**
  - Domänen-/Interaktionsdatendiagramm
- Klassen für **interne Verarbeitung**
  - Systemfunktionen
- Klassen für **Benutzungsoberfläche**
  - Views und Navigation aus UI-Struktur

	Kohäsion	Kopplung
Bedeutung	Abhängigkeit zw. Elementen einer Komponente	Abhängigkeit zw. Komponenten
Ziel	hoch	niedrig
Vorteil	Wartbarkeit einfacher	Performance höher
Wie erreichen	Verwendung geeigneter Muster	Schnittstellenkopplung

!Daten- und Strukturkopplung vermeiden!

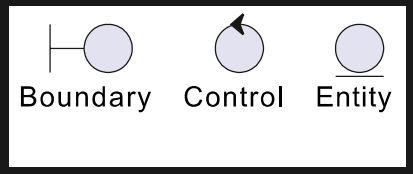
Wichtige Unterscheidungen

- **Direkter** vs. **Indirekter** Zugriff
  - Direkt: Zugriff auf eigene Attribute
  - Indirekt: Zugriff auf Attribute anderer Objekte
- **Grund-** vs. **komplexe** Operationen
  - Grund: liest und ändert Attribute nur direkt
  - komplex: liest und ändert Attribute auch indirekt

Analyseklassenmodell

- **Objekte:** Fachgegenstände
- **Klassen:** Fachbegriffe
- **Vererbung:** Begriffsstruktur
- **Funktionale** Essenz
- Projektspezifisch
- Grobe Strukturskizze

Stereotypen Analyseklassen

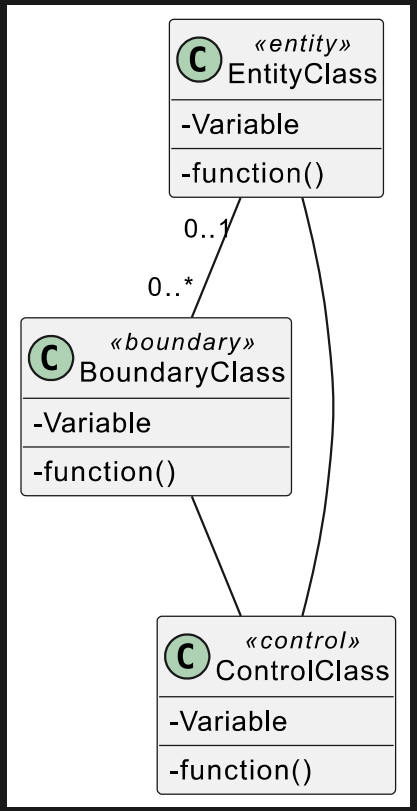


- Entitätsklasse: Beschreibt Objekte mit dauerhafter Existenz
- Steurungsklasse: Beschreibt Vorgänge und Reihenfolgen
- Dialogklasse: Erwähnt Nutzerinteraktionen und Inhalt

Vorgehen Analyseklassenmodell

1. Klassen, Attribute, Assoziationen bestimmen
2. (Relevante) Operationen der Klassen bestimmen
3. Vererbung nutzen und komplexe Assoziationen (z.B. Aggregationen) beschreiben
4. Klassendiagramm konsolidieren

Aufbau

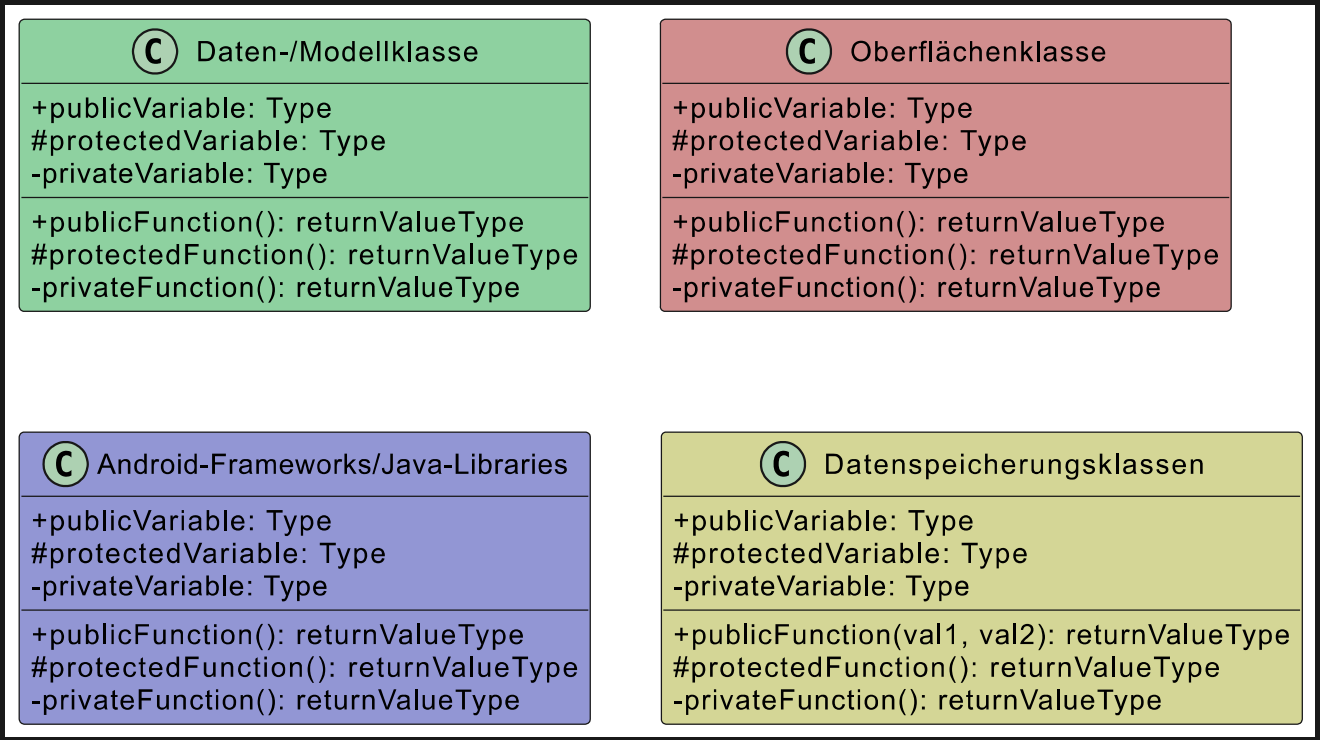


!Kontrollklassen vermeiden!

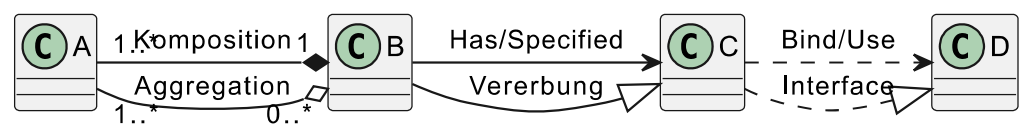
Entwurfsklassenmodell

- **Objekte:** Softwareeinheiten
- **Klassen:** Schemata
- **Vererbung:** Programmableitung
- **konkrete Implementierungsziele** erfüllen
- Gesamtstruktur System
- Genaue Strukturdefinition
- Verfeinerung der Analyseklassen

Aufbau


Daten-/Modellklasse
+publicVariable: Type
#protectedVariable: Type
-privateVariable: Type
+publicFunction(): returnValueType
#protectedFunction(): returnValueType
-privateFunction(): returnValueType
Oberflächenklasse
+publicVariable: Type
#protectedVariable: Type
-privateVariable: Type
+publicFunction(): returnValueType
#protectedFunction(): returnValueType
-privateFunction(): returnValueType
Android-Frameworks/Java-Libraries
+publicVariable: Type
#protectedVariable: Type
-privateVariable: Type
+publicFunction(): returnValueType
#protectedFunction(): returnValueType
-privateFunction(): returnValueType
Datenspeicherungsklassen
+publicVariable: Type
#protectedVariable: Type
-privateVariable: Type
+publicFunction(val1, val2): returnValueType
#protectedFunction(): returnValueType
-privateFunction(): returnValueType

Pfeilarten



- **Aggregation:** Objekte gemeinsam komplexeres Objekt (z.B. Company o- Department; also Department so weniger komplex, unter Company gemeinsam kompexer)
- **Komposition:** Objekt existiert nur mit anderem Objekt

Pfeilarten

- **Has/specifies:** Objekt1 spezifiziert Objekt2, funktioniert aber auch ohne vollständig
- **Vererbung:** Objekt extends otherObject
- **Bind:** Template wird genutzt
- **Use:** Objekt1 braucht Objekt2 um vollständig zu funktionieren
- **Interface:** Verbindung zw. Objekt und Interface

Übungsblatt 07

7.1 (Testat, Team)

- SF **Swap Pokemon** implementieren
- Testfälle dazu schreiben
  - Spezifisch sein
- Nutzt entwedder das Template oder eure Lösung
  - Im Template hilfreiche ToDos
- Folgt den Hinweisen zur Implementierung

7.2 (Testat, Einzeln)

- Klassendiagramm UI-Code
- Jira befolgen

7.3 (Testat, Team)

- Ab jetzt auch programmieren im MovieManager
- Implementierung von Sortierung nach WatchDate
- Testfälle überlegen

7.4 (Testat, Einzeln)

- Sequenzdiagramm
- SF Unlink Performer

7.5 (Nicht Testat)

- Vorbereitung auf Zustandsdiagramme