Programmierung von Systemen

Erik Neller — Dozent: Matthias Tichy

24. August 2020

ulm university universität ulm

Inhaltsverzeichnis

1 UML-Klassendiagramme

1.1 Klassen und Schnittstellen

Methoden und Attribute werden wie in Java in der Form `$TYP $NAME` angegeben, ist die Methode vom Typ `void` entfällt der Rückgabetyp.

- `private` wird durch "-" symbolisiert
- `package` wird durch "~" symbolisiert
- `protected` wird durch "#" symbolisiert
- `public` wird durch "+" symbolisiert
- `static` wird unterstrichen

{`readonly`} ist ein UML Attribut das für Konstanten verwendet wird. Die Multiplizität wird genutzt um Listen und Mengen von Objekten anzugeben:

1. Typ
2. Größe begrenzt {1..x} oder unendlich [*]
3. Attribute wie {order} und {unique}

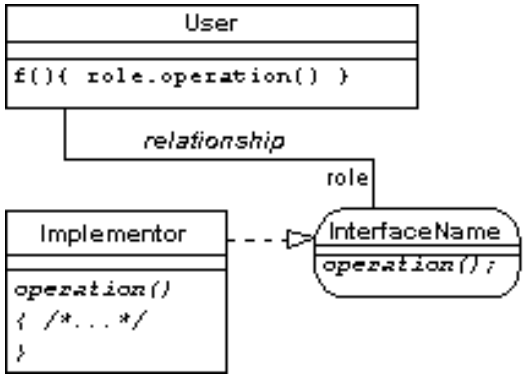
Der Block einer Klasse / eines Interfaces besteht aus drei Feldern:

1. Name mit Modifikatoren
2. Attribute (Variablen)
3. Methoden

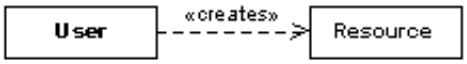
Modifikatoren für die Klasse können sein: «`interface`» oder {`abstract`} bzw. der Name in *kursiv*.

1.2 Relationen

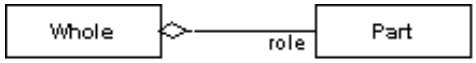
`extends` oder `implements` wird dargestellt durch einen nicht ausgefüllten Pfeil, wobei die Linie bei gleichen Typen (Interface erbt von Interface, Klasse von Klasse) durchgezogen ist, wenn eine Klasse ein Interface implementiert, gestrichelt.



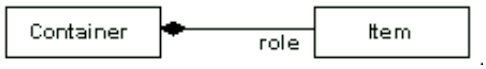
- Abhängigkeit (Dependency): User nutzt Ressource, aber die Ressource ist nicht Teil der User Klasse. Wird die Ressource modifiziert, muss auch der User modifiziert werden.



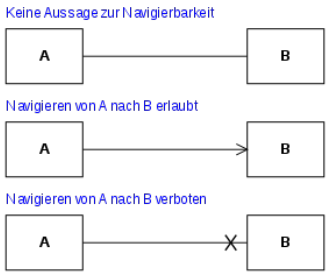
- Aggregation: "ist Teil von", wird symbolisiert durch einen Pfeil mit leerer Raute



- Komposition "besteht aus", wird symbolisiert durch einen Pfeil mit ausgefüllter Raute



- Assoziation: Die Klassen sind auf eine beliebige Weise verbunden, nutzen Methoden der anderen aber nicht auf die oben genannten Weisen



Kommentare werden mit einer gestrichelten Linie mit dem eigentlichen Objekt verbunden.

2 Version Control Systems

2.1 Grundlegende Funktionen

Version Control Systems (VCS) erlauben die Verwaltung von mehreren Teilen und Versionen eines Projekts und damit die Zusammenarbeit von mehreren Teilnehmern.

- Rechteverwaltung (z.B. Entwickler von Front-und Backend, Projektmanagement)
- Archivierung in verschiedenen Versionen (einfach anhand der gemachten Änderung, anstatt vollständige Backups zu machen)
- Speicherung von Metadaten: Historie von Änderungen mit Datum, Autor, etc.
- Backup zur Wiederherstellung lokal gelöschter Daten oder versehentlicher Änderungen
- Zentralisierung auf Server

2.2 Git

Das Git System besteht aus vier Teilen, von denen drei durch Git selbst implementiert werden. Jede Änderung durchläuft sie in dieser Reihenfolge:

1. Der eigentlichen Arbeitsplatz / *Workspace* in dem Änderungen an Dateien vorgenommen werden
2. Die *staging area*, in der *commits* aus einzelnen Änderungen an Dateien feingranular (bis zu einzelnen Zeilen) zusammengesetzt werden
3. Das *lokalen Repository*
4. Das *remote Repository* auf einem Server, beispielsweise GitHub oder GitLab

Auf den einzelnen Bereichen existieren verschiedene Befehle. Für die staging area:

- `git init` erstellt ein neues Git-Repository im aktuellen Verzeichnis
- `git add` um Dateien der staging area hinzuzufügen
- In einer `.gitignore` Datei können Regeln für Dateien angegeben werden, die generell nicht mit in die staging area aufgenommen werden sollen
- `git status` um aktuell geänderte und getrackte Dateien zu sehen
- `git rm --cached` um Dateien aus der staging area zu entfernen

Für das lokale Repository:

- `git commit -m $MESSAGE` um den Inhalt der staging area in das lokale Repository hinzuzufügen
- `git checkout $COMMIT-HASH` erlaubt das Wiederherstellen vorheriger Zustände von commits
- `git log` zeigt den Verlauf von commits an
- `git remote add $NAME $ADDRESS` verknüpft das lokale Repository mit einem remote Repository

Und für das remote Repository:

- `git push $REPOSITORY $BRANCH` um den lokalen commit auf den Server zu legen

- `git pull` um das lokale Repository mit den Änderungen aus dem remote Repository zu aktualisieren
- `git clone $REPOSITORY` um das ganze Repository lokal zu speichern

Für verschiedene Themengebiete / Zuständigkeitsbereiche existiert das Konzept von Branches(Verzweigungen), die mit `git branch $NAME` erstellt werden können, um anschließend mit `git checkout $NAME` in den Branch zu wechseln, oder in Kurzform: `git checkout -b $NAME`. Mit `git merge $NAME` kann dann ein Branch in den jeweils Aktuellen integriert, oder mit `git branch -d $NAME` gelöscht werden. Ist keine automatische Vereinigung der Branches möglich, müssen die angezeigten Dateien manuell geändert und anschließend mit `git add $FILE` Alternativ kann der Branch in das Repository aufgenommen werden: `git push $REMOTE $BRANCH`.

3 Objektorientierung

3.1 Interfaces

Analog zur abstrakten Klasse erlaubt ein Interface keine Instanziierung. Es enthält keine tatsächliche Implementierung, sondern nur Methodenrümpfe und evtl. Konstanten und muss dementsprechend in jeder Klasse mit dem Schlüsselwort `implements` implementiert werden. Im Gegensatz zu abstrakten Klassen können mehrere Interfaces von einer Klasse implementiert werden.

3.2 Sichtbarkeit

Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

3.3 Annotations

Verschiedene Standardannotationen, eigene definierbar. Durch Reflection zur Laufzeit auslesbar.

- `@Override` gibt an dass hier ein Element überschrieben werden soll
- `@Deprecated` gibt eine Warnung aus dass Element veraltet ist
- `@SuppressWarnings()` unterdrückt die angegebene Warnung
- `@Documented` nimmt nachfolgende Annotationen in die JavaDoc mit auf

JavaDoc:

- `@author`
- `@version`
- `@param` zur Beschreibung von Methodenparametern
- `@return` beschreibt den Rückgabewert
- `@exception`, `@throws` Beschreibt Fehlermeldungen, die diese Methode produzieren kann
- `@link` Verknüpfung zu anderem Symbol

3.4 Enumerations

Mit einem `enum` kann eine Menge von Konstanten definiert werden, entweder auf Klassenlevel oder innerhalb einer Klasse. Sie sind auch selbst eine Klasse, deren Attribute `public static final` definiert sind.
Methoden:

- `int ordinal()` gibt die Position in der Liste des Enums zurück
- `String name()` gibt den Namen der Konstanten zurück
- `int valueOf` gibt den zugeordneten Wert der Variablen zurück

4 Collections-Generics

4.1 Datenstrukturen

4.1.1 Array

Statische Größe, wird mit Länge und Datentyp gespeichert, zB int[5]. Objekt x erhalten: array[x]. Länge: array.length

4.1.2 ArrayList

Klasse die zur Speicherung Arrays verwendet, diese aber ersetzt wenn die Methoden `add()` oder `remove()` aufgerufen werden. Mit `toArray()` kann der Array erhalten werden, mit `size()` die Größe, da diese dynamisch ist. Objekt erhalten: arraylist.get(x).

4.1.3 Linked List

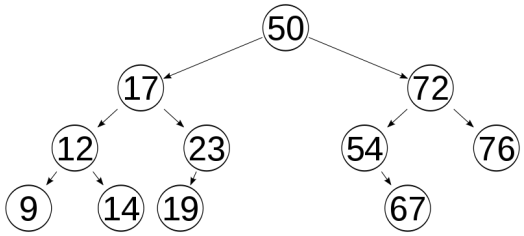
Jedes Listenelement enthält einen Zeiger auf das nächste Element, bei dem letzten Element ist der Zeiger `NULL`.

4.1.4 Double Linked List

Wie linked list, nur dass jedes Element zusätzlich einen Zeiger auf das vorherige Element enthält (beim ersten Element `NULL`).

4.1.5 Sorted Tree

Setzt die Sortierbarkeit der Objekte voraus: Objekte mit kleinerem Wert sind links des Vaterknotens, mit größerem rechts. Beim Einfügen / Entfernen reorganisieren: rot-schwarz Bäume, B Baum, B+Baum



4.1.6 HashTable

Objekte werden gehasht, also ein Wert aus ihnen berechnet, anhand dessen sie in in eine Tabelle eingeordnet werden.

4.1.7 Map

Speichert nicht einzelne Werte, sondern Tupel aus (Key, Value), nutzt dann zB eine Liste aus Tupeln oder eine Hashtabelle anhand der Keys.

4.2 Collections API

Stellt effiziente Datenstrukturen für viele Anwendunge bereit, sodass Datenstrukturen und Algorithmen nicht selbst implementiert werden müssen.

- `LinkedList`: add, remove, get, clear, set, indexOf, size, contains
- `TreeSet`: add, remove, contains, size