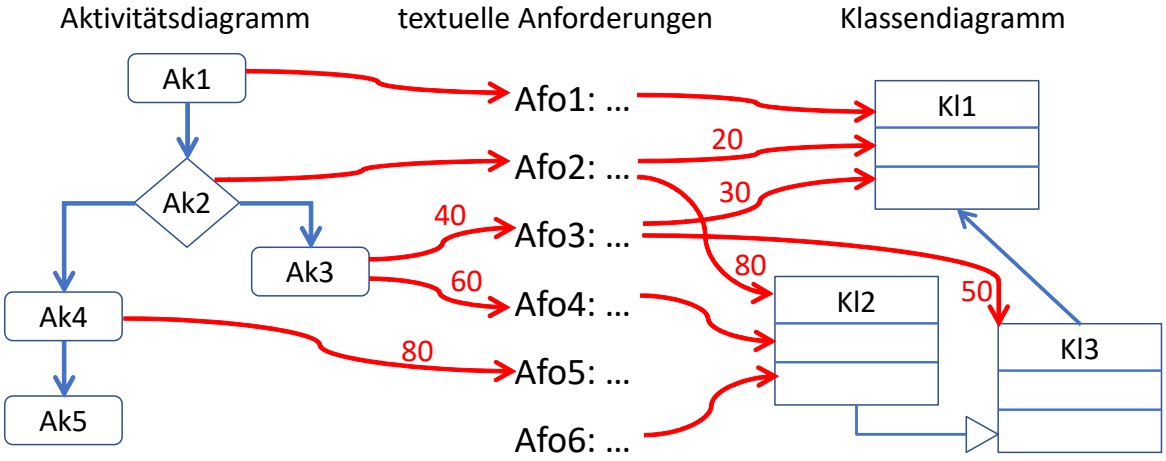

Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing-Wiss. und Informatik
- Software-Entwicklung -

Software Quality Management

Sommersemester 2025
3. Aufgabenblatt

Aufgabe 0.3
Geben Sie das Lösungswort des Quiz aus der Lernnotiz an.

Aufgabe 4
In der Vorlesung wurde gezeigt, dass es für die Nachverfolgbarkeit, also „was wurde mit was verknüpft/ wie umgesetzt“ von Entwicklungsentscheidungen sehr wichtig ist, diese als Tracing-Informationen zu dokumentieren. In dieser Aufgabe wird dies mit einer Software umgesetzt. Eventuell hilfreich für die Begriffserklärung kann es sein, erst das folgende Diagramm zu verstehen (s. Vorlesung) und sich dann den Code mit den Ausgaben auf der folgenden Seite anzusehen.



Ziel der Aufgabe ist es, die Grundlage einer einfachen Tracing-Software in Java zu erstellen, die die Zuordnung sogenannter Artefakte ermöglicht. Artefakte können z. B. Aktionen eines Aktivitätsdiagramms (links), Anforderungen in Textform (mitte) und Klassen (rechts) sein. Die obige Abbildung zeigt eine Beispielzuordnung, bei der neben der Abhängigkeit auch ein prozentualer Anteil der Abdeckung angegeben wird.
Nutzen Sie als Basistyp die Klasse Artefakt aus dem Projekt sqmAufgabeAnforderungstracing von der Veranstaltungsseite. Dabei hat jedes Artefakt eine eindeutige id, einen fachlichen Inhalt und eine Sammlung von Gruppen, denen das Artefakt zugeordnet ist. Jede Gruppe wird vereinfacht durch einen String angegeben. Eine Gruppe kann z. B. „Aktion“, „Anforderung“ oder „Klasse“ sein. Eine Zuordnung zu mehreren Gruppen soll erlaubt werden.

```
public class Artefakt {
  private int id;
  private String inhalt;
  private Set<String> gruppen;
```

Schauen Sie sich die etwas ungewöhnliche Verwaltung aller Artefakte in der Klasse Artefakt an. (Wenn Sie sie nicht mögen, können Sie das Projekt gerne refaktorieren.)
Nutzen Sie weiterhin die gegebene Verbindung, mit der ein Artefakt einem anderen Artefakt zu einem bestimmten Prozentsatz zugeordnet werden kann (rote Linien im Diagramm).

```
public class Verbindung {
  private Artefakt von;
  private Artefakt nach;
  private int prozent;
```

Implementieren Sie einen Typ Zuordnung, der eine Sammlung von Verbindungen enthält. Es soll immer nur eine Zuordnung zwischen zwei Gruppen möglich sein, z. B. von Aktionen zu Anforderungen.

Der Typ Zuordnung soll dabei die folgenden Interfaces realisieren:

```
public interface Zuordnen
```

Methodenübersicht

Alle Methoden | Instanzmethoden | Abstrakte Methoden

Modifizierer und Typ	Methode	Beschreibung
boolean	zuordnen(Artefakt von, Artefakt nach, int prozent)	ordnet dem Artefakt von das Artefakt nach zu prozent Prozent zu, Ergebnis zeigt an, ob Zuordnung erlaubt ist, also von und nach die richtigen Gruppen haben und der Prozentwert sinnvoll ist; falls schon eine Zuordnung von zu nach existiert, wird diese ersetzt

```
public interface Analysieren
```

Methodenübersicht

Alle Methoden | Instanzmethoden | Abstrakte Methoden

Modifizierer und Typ	Methode	Beschreibung
List<Verbindung>	abhaengig(int id)	gibt alle Verbindungen aus, in denen ein anderes Artefakt dem Artefakt mit der Id id zugeordnet wird.
List<Artefakt>	nichtAbhaengig()	gibt alle Artefakte aus, die von keinem Artefakt abhaengig sind (genauer Artefakte, die nicht auf der nach-Seite stehen und die zur Gruppe nach der das Interface implementierenden Klasse gehoeren)
List<Verbindung>	nichtVollstaendigZugeordnet()	berechtnet zu allen Artefakten, denen keine Artefakte mit der Gesamtprozenstsumme von 100 zugeordnet wurde (genauer Artefakte, die nicht auf der von-Seite stehen und die zur Gruppe von der das Interface implementierenden Klasse gehoeren) die daran beteiligten Verbindungen
List<Artefakt>	nichtZugeordnet()	gibt alle Artefakte aus, denen kein Artefakt zugeordnet wurde (genauer Artefakte, die nicht auf der von-Seite stehen und die zur Gruppe von der das Interface implementierenden Klasse gehoeren)
List<Verbindung>	zugeordnet(int id)	gibt alle Verbindungen aus, mit denen einem Artefakt mit der Id id ein weiteres Artefakt zugeordnet wird (rechte Seite der Zuordnung).

Schreiben Sie dann eine Klasse Zuordnungskette, mit der eine beliebige Anzahl von Zuordnungen sequenziell verknüpft werden kann, dabei muss die Zielgruppe „nach“ einer Zuordnung immer die Ausgangsgruppe „von“ der nachfolgenden Zuordnung der Liste sein.


Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing-Wiss. und Informatik
- Software-Entwicklung -

Software Quality Management

Sommersemester 2025
3. Aufgabenblatt

Die Klasse Zuordnungsliste soll wieder das Interface Analysieren realisieren, dabei muss zugeordnet() nur für den Anfang und abhaengig() nur für das Ende sinnvolle Ergebnisse liefern, die sich aber auf die gesamte Zuordnungsliste beziehen sollen. Die folgende im Projekt vorhandene folgende Methode nutzt die obigen Beispieldaten.

```
public static void main(String[] args) {
  Artefakt ak1 = new Artefakt("Ak1", "Aktion");
  Artefakt ak2 = new Artefakt("Ak2", "Aktion");
  Artefakt ak3 = new Artefakt("Ak3", "Aktion");
  Artefakt ak4 = new Artefakt("Ak4", "Aktion");
  Artefakt ak5 = new Artefakt("Ak5", "Aktion");

  Artefakt afo1 = new Artefakt("Afo1", "Anforderung");
  Artefakt afo2 = new Artefakt("Afo2", "Anforderung");
  Artefakt afo3 = new Artefakt("Afo3", "Anforderung");
  Artefakt afo4 = new Artefakt("Afo4", "Anforderung");
  Artefakt afo5 = new Artefakt("Afo5", "Anforderung");
  Artefakt afo6 = new Artefakt("Afo6", "Anforderung");
  Artefakt kl1 = new Artefakt("Kl1", "Klasse");
  Artefakt kl2 = new Artefakt("Kl2", "Klasse");
  Artefakt kl3 = new Artefakt("Kl3", "Klasse");

  Zuordnung  aktanf = new Zuordnung("Aktion", "Anforderung");
  aktanf.zuordnen(ak1, afo1, 100);
  aktanf.zuordnen(ak2, afo2, 100);
  aktanf.zuordnen(ak3, afo3, 40);
  aktanf.zuordnen(ak3, afo4, 60);
  aktanf.zuordnen(ak4, afo5, 80);

  Zuordnung anfkla = new Zuordnung("Anforderung", "Klasse");
  anfkla.zuordnen(afo1, kl1, 100);
  anfkla.zuordnen(afo2, kl1, 20);
  anfkla.zuordnen(afo2, kl2, 80);
  anfkla.zuordnen(afo3, kl1, 30);
  anfkla.zuordnen(afo3, kl3, 50);
  anfkla.zuordnen(afo4, kl2, 100);
  anfkla.zuordnen(afo6, kl2, 100);

  Zuordnungskette  zlk = new Zuordnungskette();

  System.out.println("aktanf hinzu: " + zlk.hinzu(aktanf));
  System.out.println("anfkla hinzu: " + zlk.hinzu(anfkla));
  System.out.println("ak3 zugeordnet: " + zlk.zugeordnet(ak3.getId()));
  System.out.println("ak5 zugeordnet: " + zlk.zugeordnet(ak5.getId()));
  System.out.println("kl1 abhaengig: " + zlk.abhaengig(kl1.getId()));
  System.out.println("kl3 abhaengig: " + zlk.abhaengig(kl3.getId()));
  System.out.println("nicht zugeordnet: " + zlk.nichtZugeordnet());
  System.out.println("nicht abhaengig: " + zlk.nichtAbhaengig());
  System.out.println("nicht vollstaendig zugeordnet: " +
      zlk.nichtVollstaendigZugeordnet());
}
```

Die resultierende Ausgabe soll fachlich etwa wie folgt aussehen, andere Formatierungen und Reihenfolgen sind erlaubt. Weiterhin dürfen auch Pfade und weitere Methoden frei umbenannt werden.

```
aktanf hinzu: true
anfkla hinzu: true
ak3 zugeordnet: [[(3) Ak3 [Aktion] -> (8) Afo3 [Anforderung] 40%], [(3) Ak3
[Aktion] -> (9) Afo4 [Anforderung] 60%], [(8) Afo3 [Anforderung] -> (12)
Kl1 [Klasse] 30%], [(8) Afo3 [Anforderung] -> (14) Kl3 [Klasse] 50%], [(9)
Afo4 [Anforderung] -> (13) Kl2 [Klasse] 100%]]
ak5 zugeordnet: []
kl1 abhaengig: [[(6) Afo1 [Anforderung] -> (12) Kl1 [Klasse] 100%], [(7)
Afo2 [Anforderung] -> (12) Kl1 [Klasse] 20%], [(8) Afo3 [Anforderung] ->
(12) Kl1 [Klasse] 30%], [(1) Ak1 [Aktion] -> (6) Afo1 [Anforderung] 100%],
[(2) Ak2 [Aktion] -> (7) Afo2 [Anforderung] 100%], [(3) Ak3 [Aktion] -> (8)
Afo3 [Anforderung] 40%]]
kl3 abhaengig: [[(8) Afo3 [Anforderung] -> (14) Kl3 [Klasse] 50%], [(3) Ak3
[Aktion] -> (8) Afo3 [Anforderung] 40%]]
nicht zugeordnet: [(5) Ak5 [Aktion], (10) Afo5 [Anforderung]]
nicht abhaengig: [(11) Afo6 [Anforderung]]
nicht vollstaendig zugeordnet: [[(4) Ak4 [Aktion] -> (10) Afo5
[Anforderung] 80%], [(8) Afo3 [Anforderung] -> (12) Kl1 [Klasse] 30%], [(8)
Afo3 [Anforderung] -> (14) Kl3 [Klasse] 50%]]
```

Hinweis: Wichtig ist hier nur, dass die angedeutete Funktionalität umgesetzt wird. Das Testen kommt später.