



Klausur Softwaretechnologie SS 2015

Prof. Dr.rer.nat.habil.
Uwe Aßmann

Name:	
Vorname:	
Immatrikulationsnummer:	

Aufgabe	Maximale Punktzahl	Erreichte Punktzahl
1	13	
2	22	
3	55	
Gesamt	90	

Hinweise:

- In der Klausur ist als Hilfsmittel lediglich ein **A4-Blatt, beidseitig beschrieben**, zugelassen.
- Die Klammerung der Aufgabenblätter darf **nicht** entfernt werden.
- Tragen Sie bitte die Lösungen auf den Aufgabenblättern ein!
- Verwenden Sie keine roten, grünen Stifte oder Bleistifte!
- Es ist kein eigenes Papier zu verwenden! Bei Bedarf ist zusätzliches Papier bei der Aufsicht erhältlich. Bitte jedes zusätzliche Blatt mit Name, Vorname und Immatrikulationsnummer beschriften.
- Es sind alle Aufgabenblätter abzugeben!
- Ergänzen Sie das Deckblatt mit Name, Vorname und Immatrikulationsnummer!
- Halten Sie Ihren Studentenausweis und einen Lichtbildausweis zur Identitätsprüfung bereit.
- **Achtung!** Das Zeichen ☞ heißt: **Hier ist Java-Text einzufügen!**

Aufgabe 1: Flaschenrückgabeautomat (13 Punkte)

Ein typischer Flaschenrückgabeautomat kennt drei Zustände und hat nur eine begrenzte Kapazität (*kap*):

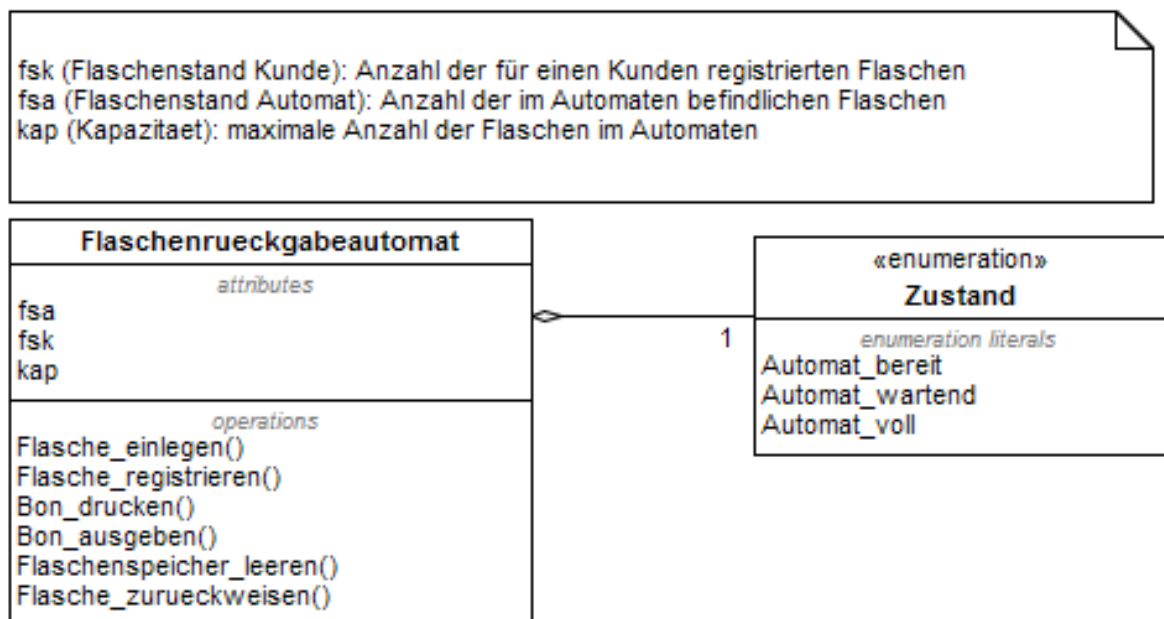
- *Automat_bereit*: Er ist bereit, neue Flaschen entgegenzunehmen
- *Automat_wartend*: Er nimmt die Flaschen für einen bestimmten Benutzer entgegen und berechnet das Pfandgeld auf Basis der für den Kunden registrierten Flaschen (*fsk*).
- *Automat_voll*: Er muss geleert werden, bevor neue Flaschen eingelegt werden können.

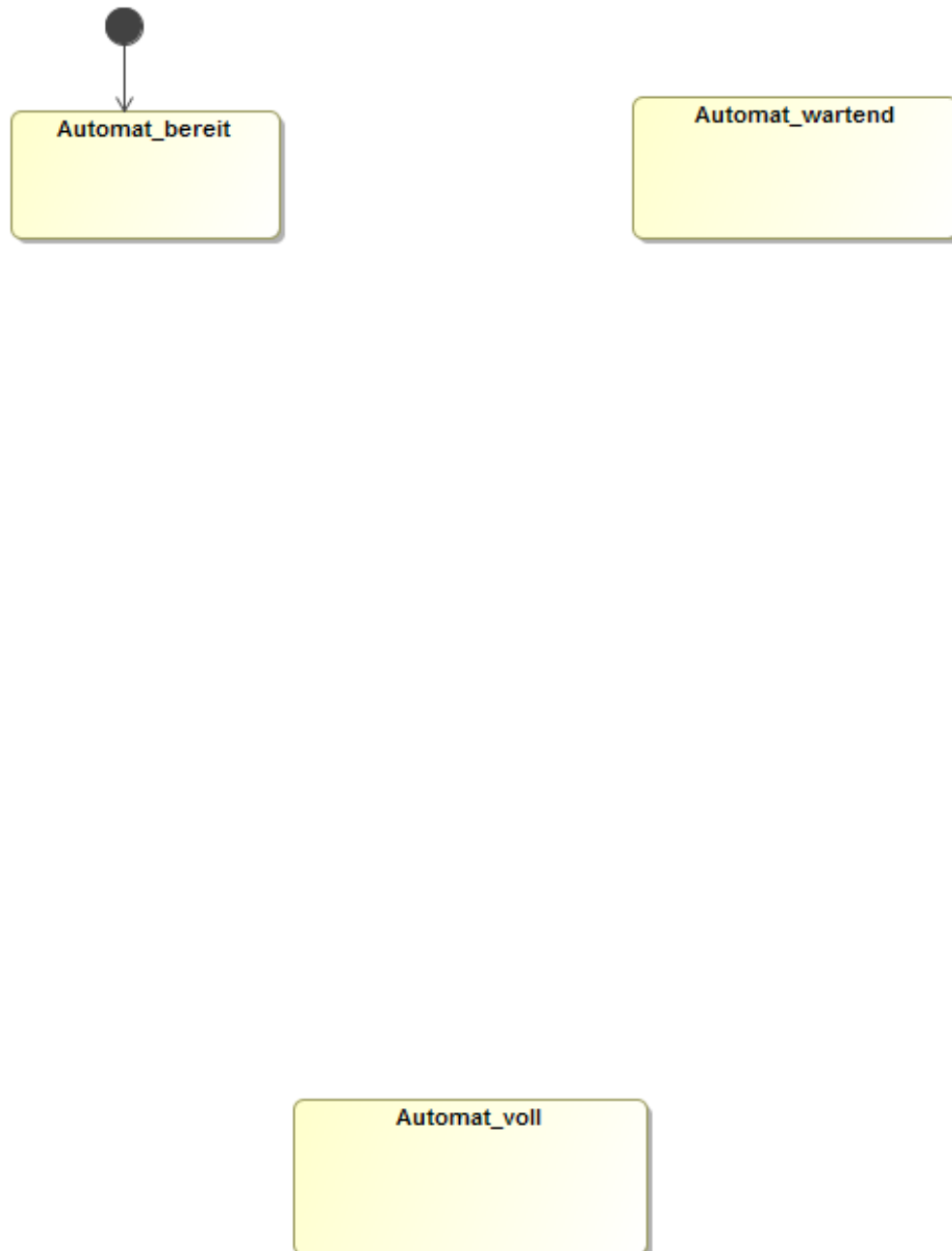
Ein Benutzer kann jederzeit eine Flasche einlegen (*Flasche_einlegen*). Sofern der Automat nicht voll ist, wird die Flasche eingezogen und für die Berechnung des Pfandgeldes registriert (*Flasche_registrieren*). Anderenfalls wird die Flasche zurückgewiesen (*Flasche_zurueckweisen*).

Ebenso kann jederzeit die Bon-Taste gedrückt werden (*Bon_drucken*). Der Bon wird nur ausgegeben (*Bon_ausgeben*), wenn bereits mindestens eine Flasche registriert ist. In diesem Fall wird der Flaschenstand des Kunden auf null gesetzt.

Der Flaschenspeicher (*fsa*) kann ebenfalls jederzeit geleert werden (*Flaschenspeicher_leeren*). Die Anzahl der registrierten Flaschen eines Kunden wird dabei nicht verändert.

Modellieren Sie diesen Flaschenrückgabeautomaten als Verhaltenszustandsmaschine mit allen möglichen Zustandsübergängen! Nutzen Sie zur Modellierung die Attribute und Methoden der gegebenen Klasse Flaschenrückgabeautomat und denken Sie daran, das Verhalten des Automaten exakt zu spezifizieren. Der Automat hört nicht auf zu existieren.





Aufgabe 2: Schachprogramm DokChess¹ (22 Punkte)

Schach wird zwischen zwei Gegnern gespielt, die abwechselnd ihre Figuren auf einem quadratischen Spielbrett, Schachbrett genannt, ziehen.

Erstellen Sie aus der nachfolgenden Beschreibung ein Domänenmodell in Form eines UML-Analyseklassendiagramms!

- **Denken Sie an Klassen, Enumerationen, Attribute, Methoden und Klassenbeziehungen.**
- **Das Domänenmodell soll aktuelle (gültige) Spielsituationen auf dem Schachbrett beschreiben. Das bedeutet, dass mindestens zwei Figuren auf dem Brett stehen.**

Eine Schachfigur ist gekennzeichnet durch ihre Farbe (schwarz oder weiß) und ihre Art (König, Dame, Turm, Läufer, Springer, Bauer). Insgesamt gibt es 32 Figuren.

Ein Feld kann maximal von einer Figur besetzt sein. Ein Feld ist durch seine Koordinaten (auf dem Schachbrett), d.h. durch Angabe seiner Linie und Reihe gekennzeichnet.

Ein Zug gibt an, von welchem Feld eine Figur nach welchem Feld gezogen wird. In einer Ausnahme kann eine Figur in eine andere Figur umgewandelt werden.

Die aktuelle Spielsituation wird durch die Stellung der Figuren (d. h. durch die Zuordnung zu den einzelnen Feldern) beschrieben. Zur Komplettierung der Spielsituation gehört die Information, wer am Zug (schwarz oder weiß) ist.

In einer Stellung kann ein Zug ausgeführt werden (*führeZugAus*). Ebenso kann eine Figur von einem Feld genommen werden, wenn diese geschlagen wird (*schlageFigur*).

Aus einer Stellung können gültige Züge abgeleitet werden sowie ob ein

- Schach (der König der angegebenen Farbe steht im Schach)
- Patt (aktueller Spieler hat keinen gültigen Zug, steht aber nicht im Schach)
- Matt (aktueller Spieler am Zug steht im Schach und kein Zug führt ihn aus diesem Angriff heraus)

vorliegt.

¹ Nach Stefan Zörner: Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten. Hanser Fachbuch, 2. Auflage Mai 2015

Aufgabe 3: Struktur eines Unternehmens² (55 Punkte)

Gegeben ist das Entwurfsklassendiagramm auf Seite 7. Es bildet die Struktur eines Unternehmens in Form eines Baumes ab. Die Knoten in diesem Baum sind sowohl Unternehmenseinheiten (`AbstractEnterpriseUnit`) als auch Mitarbeiter (`StaffMember`).

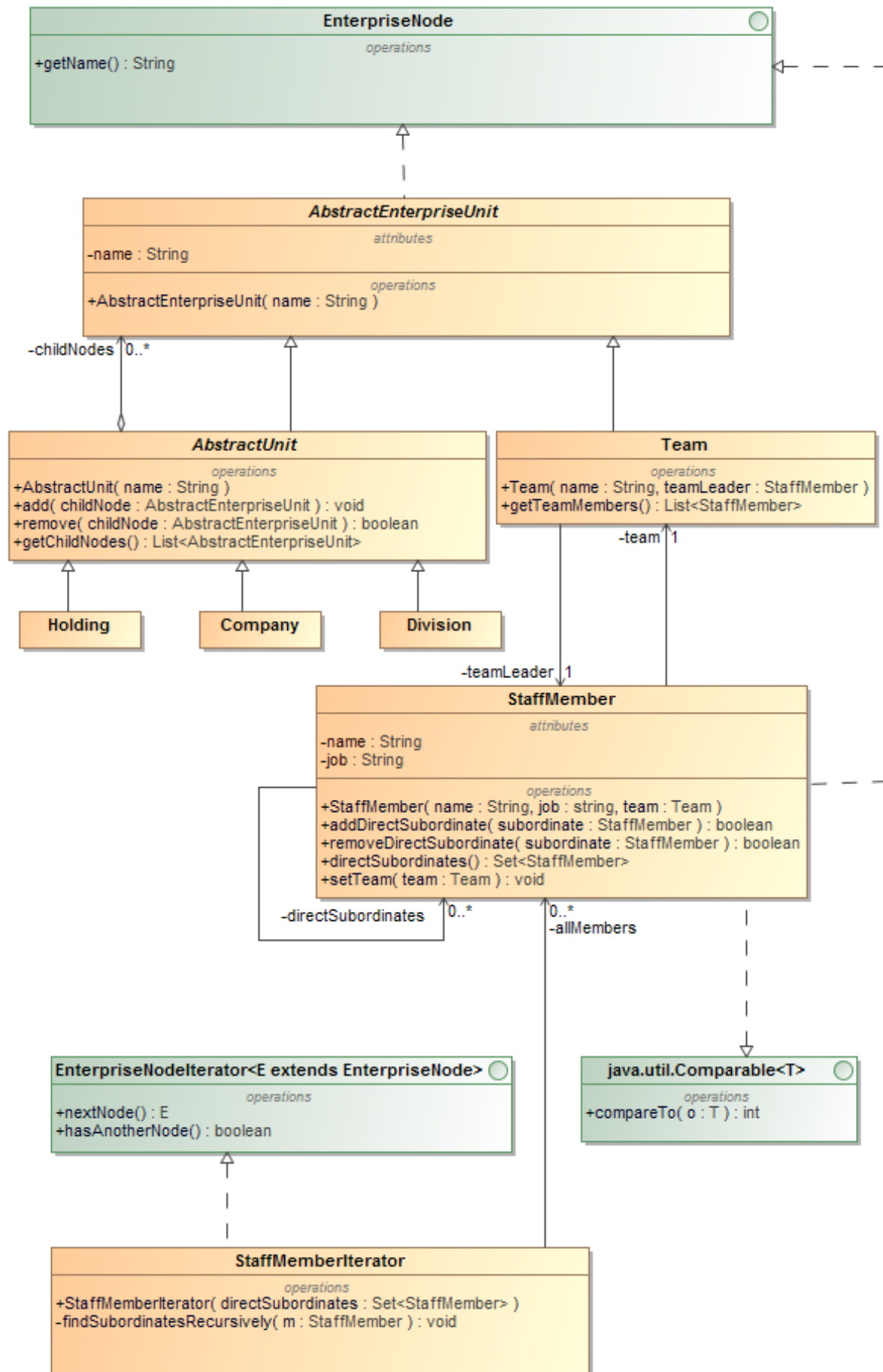
Die Unternehmenseinheiten sind hierarchisch strukturiert: Holdings (`Holding`) in Firmen (`Company`), Firmen in Abteilungen (`Division`) und Abteilungen in Teams (`Team`). Zu den Unternehmenseinheiten, die strukturiert sein können (`AbstractUnit`), kann eine untergeordnete Einheit (`childNode`) im Baum mit der Methode `add()` hinzugefügt und mit `remove()` entfernt werden. Die Methode `getChildNodes()` stellt alle unmittelbar untergeordneten Unternehmenseinheiten eines Knotens im Unternehmensbaum bereit. Teams sind die elementaren Unternehmenseinheiten und haben einen Teamleiter (`teamLeader`).

Für jeden Teamleiter werden ebenfalls in einer Baumstruktur alle Unterstellungsbeziehungen im Team abgebildet. Das bedeutet, für jeden Mitarbeiter werden die direkt unterstellten Mitarbeiter (`directSubordinates`) referenziert. Mit der Methode `addDirectSubordinates()` kann ein direkt unterstellter Mitarbeiter hinzugefügt und mit `removeDirectSubordinates()` entfernt werden. Außerdem hat jeder Mitarbeiter eine Referenz auf sein Team (`team`).

Für das Iterieren über die Baumstrukturen gibt es das Interface `EnterpriseNodeIterator`. Die Methode `hasAnotherNode()` prüft, ob es einen weiteren Knoten im Baum gibt, die Methode `nextNode()` stellt den nächsten Knoten bereit.

Die Klasse `StaffMemberIterator` implementiert den `EnterpriseNodeIterator` für die Unterstellungsbeziehungen in einem Team. Dazu müssen zunächst alle Mitarbeiter (`allMembers`) eines Teams in einer Kollektion (`List` oder `Set`) zusammengestellt werden. Der Baum wird rekursiv durchlaufen. Dem Konstruktor `StaffMemberIterator()` wird die Menge aller unmittelbar untergeordneten Knoten übergeben (`directSubordinates`). Die Hilfsmethode `findSubordinatesRecursively()` findet für einen Mitarbeiter (beginnend beim Teamleiter) rekursiv alle unterstellten Mitarbeiter. In der entstehenden Kollektion (`allMembers`) soll jeder Mitarbeiter nur genau einmal enthalten sein. Die Liste soll zudem alphabetisch nach dem Namen des Mitarbeiters sortiert sein.

² Codebeispiel nach: Patterns Kompakt, Verlag Springer Vieweg, Copyright 2013 Karl Eilebrecht



Der folgende Java-Codeausschnitt zeigt die beispielhafte Erstellung einer Holding:

```
h1 = new Holding("Holding");
c1 = new Company("Company 1");
c2 = new Company("Company 2");
d1 = new Division ("Division 1");
d2 = new Division ("Division 2");
d3 = new Division ("Division 3");
m1 = new StaffMember ("Member 1", "CEO", null);
m2 = new StaffMember ("Member 2", "CEO-S1", null);
t1 = new Team ("Team 1",m1);
t2 = new Team ("Team 2", m2);
m1.setTeam(t1);
m2.setTeam(t2);
h1.add(c1);
h1.add(c2);
c1.add(d1);
c2.add(d2);
d1.add(d3);
d3.add(t1);
d3.add(t2);
m3 = new StaffMember ("Member 3", "CEO-S2", t1);
m4 = new StaffMember ("Member 4", "CEO-S3", t1);
m5 = new StaffMember ("Member 5", "CEO-S11", t1);
m6 = new StaffMember ("Member 6", "CEO-S31", t1);
m1.addDirectSubordinate(m3);
m1.addDirectSubordinate(m4);
m3.addDirectSubordinate(m5);
m3.addDirectSubordinate(m6);
```

Teilaufgaben:

- a) Erstellen Sie für die Struktur dieser Holding ein Objektdiagramm! Beachten Sie ALLE Links (Java-Referenzen), die zwischen den Objekten erzeugt werden. (Attributwerte (Slots)/Rollennamen sollen im Diagramm nicht erscheinen.)

- b) Welches Entwurfsmuster außer dem Iterator ist im Entwurf enthalten? Zeichnen Sie das Muster und die Rollen der beteiligten Klassen in UML-Notation in das Klassendiagramm ein!
- c) Erkennen Sie ein Problem im Modell der Unternehmensstruktur? Beschreiben Sie es in kurzer verbaler Form!
-

- d) Implementieren Sie den Entwurf, in dem Sie den folgenden Java-Code vervollständigen. Achten Sie auf Konsistenz des Java-Codes zu den Variablen und Methoden im Entwurfsklassendiagramm! Verzichten Sie der Einfachheit halber auf den Test von Parametern auf Nullwerte.

```
import java.util.*; // gilt für alle Klassen

public abstract class AbstractUnit extends AbstractEnterpriseUnit {

    protected List<AbstractEnterpriseUnit> childNodes;

    public AbstractUnit(String name) {

    }

    // Jeder childNode darf nur einmal in der Liste childNodes erscheinen
    public void add(AbstractEnterpriseUnit childNode) {

    }

    public boolean remove(AbstractEnterpriseUnit childNode) {

    }

    public List<AbstractEnterpriseUnit> getChildNodes() {

    }
```

```

}
public class StaffMember implements EnterpriseNode, Comparable<StaffMember> {

    // alle unmittelbar untergeordneten Mitarbeiter
    private Set<StaffMember> directSubordinates;

    public StaffMember(String name, String job, Team team) {

        directSubordinates = new TreeSet<StaffMember>();
    }

    public String getName() {

    }

    public int compareTo(StaffMember m) {

    }

    public boolean addDirectSubordinate(StaffMember subordinate) {

    }

    public boolean removeDirectSubordinate(StaffMember subordinate) {

    }

    public Set<StaffMember> directSubordinates (){

    }

    public void setTeam(Team team) {

    }
}

```

```

public class Team extends AbstractEnterpriseUnit {
    public Team(String name, StaffMember teamLeader) {

    }

    public List<StaffMember> getTeamMembers() {

    }
}

```


```

public class StaffMemberIterator implements EnterpriseNodeIterator<StaffMember> {


    // Konstruktor berechnet allMembers
    public StaffMemberIterator(Set<StaffMember> directSubordinates) {

    }
}


```

```
private void findSubordinatesRecursively(StaffMember m) {  
    
```

```
    }
```

```
    public boolean hasAnotherNode() {  
        
```

```
    }
```

```
    public StaffMember nextNode() {  
        
```

```
    }  
}
```
