Composite Pattern

BY: KHALED BAKEER

Structural Design Pattern

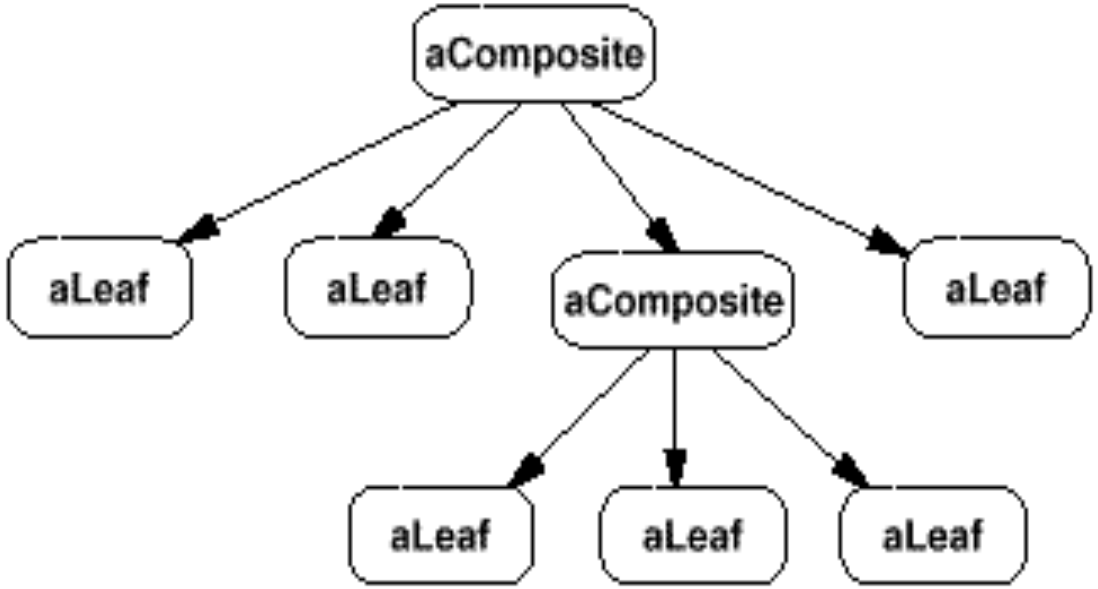
- Structural Patterns are concerned with how classes and objects are composed to form larger structures.
- Composite design pattern is a structural Design pattern.
- The composite objects let you compose primitive and other composite objects into arbitrarily complex structures.
- Use the Composite pattern when:
  - you want to represent part-whole hierarchies of objects.
  - you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

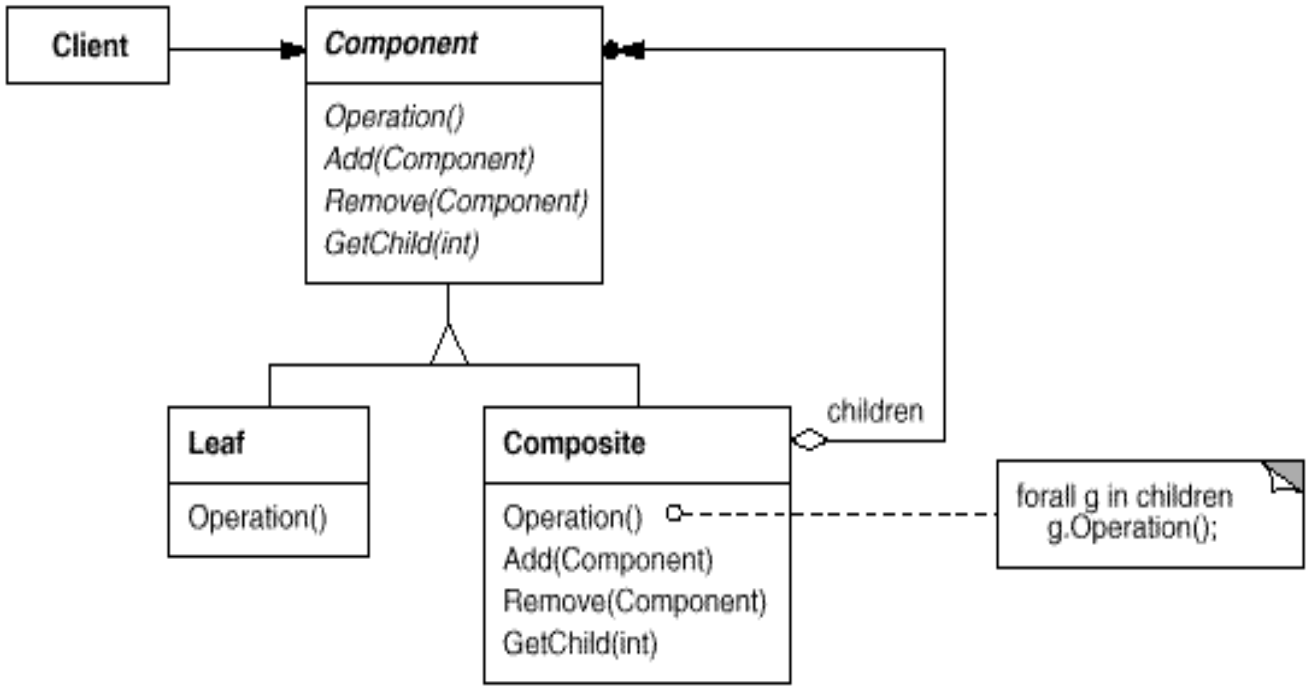
What are composite Objects?

- Objects that contains other objects.

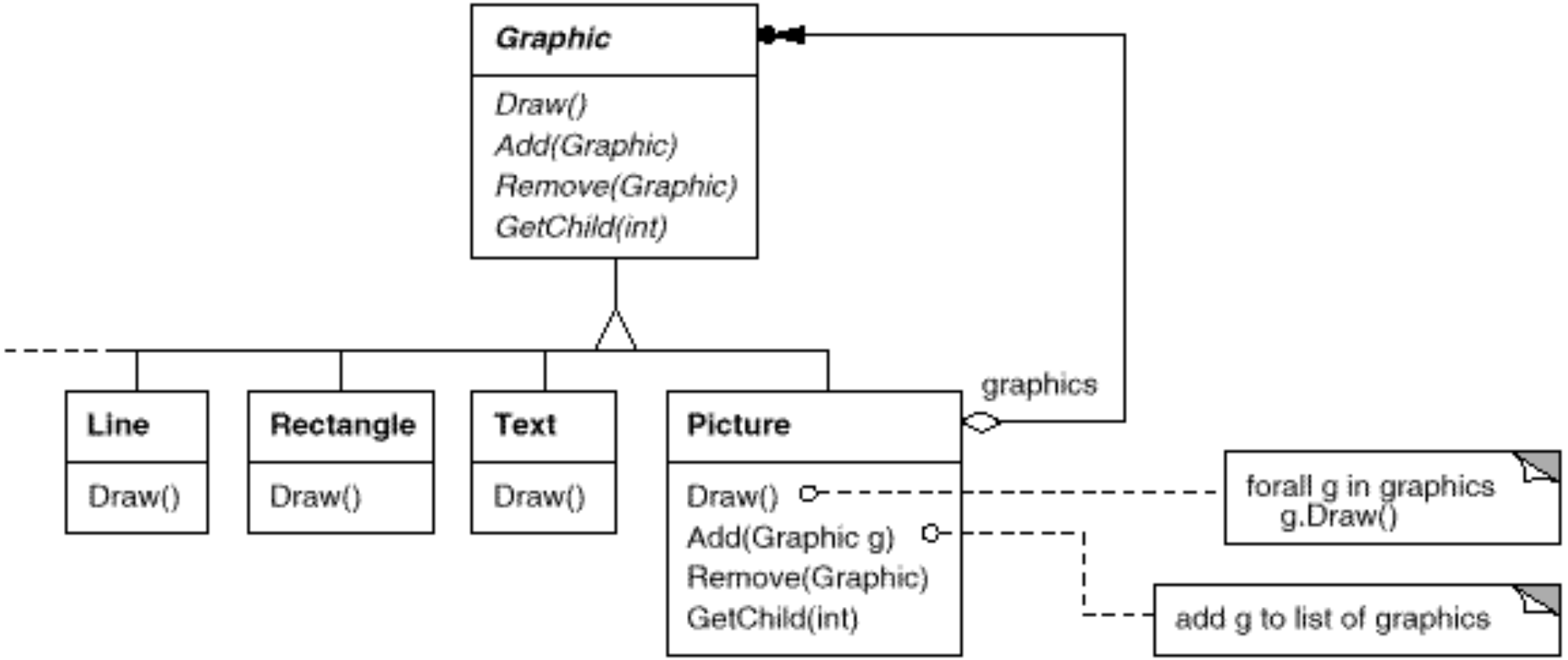
- Examples:
  - **Menus** : contains menu items.
  - **Directories**: contains files, each of which could be a directory.
  - **Containers**: contains containers, each of which could be a container.

Structure - Link

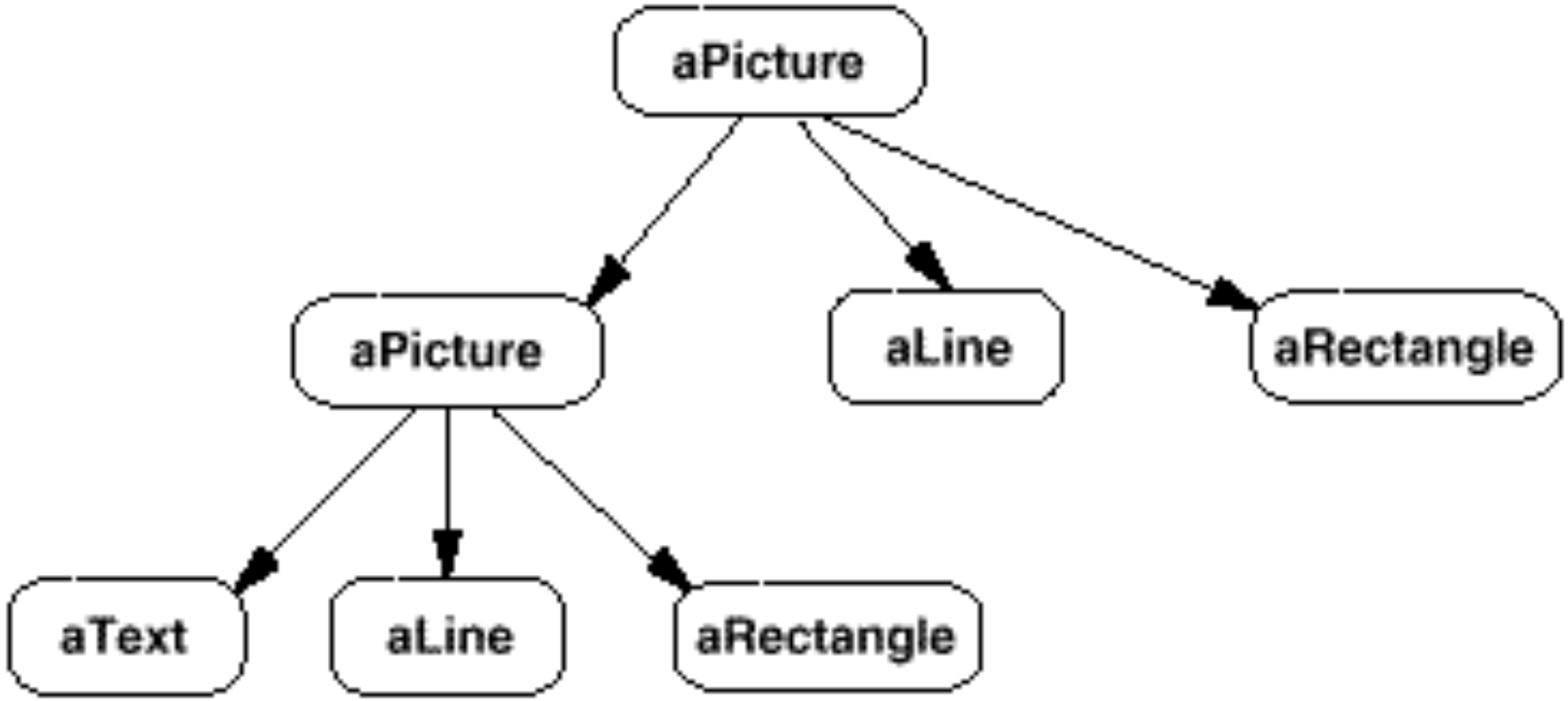




Graphics Example



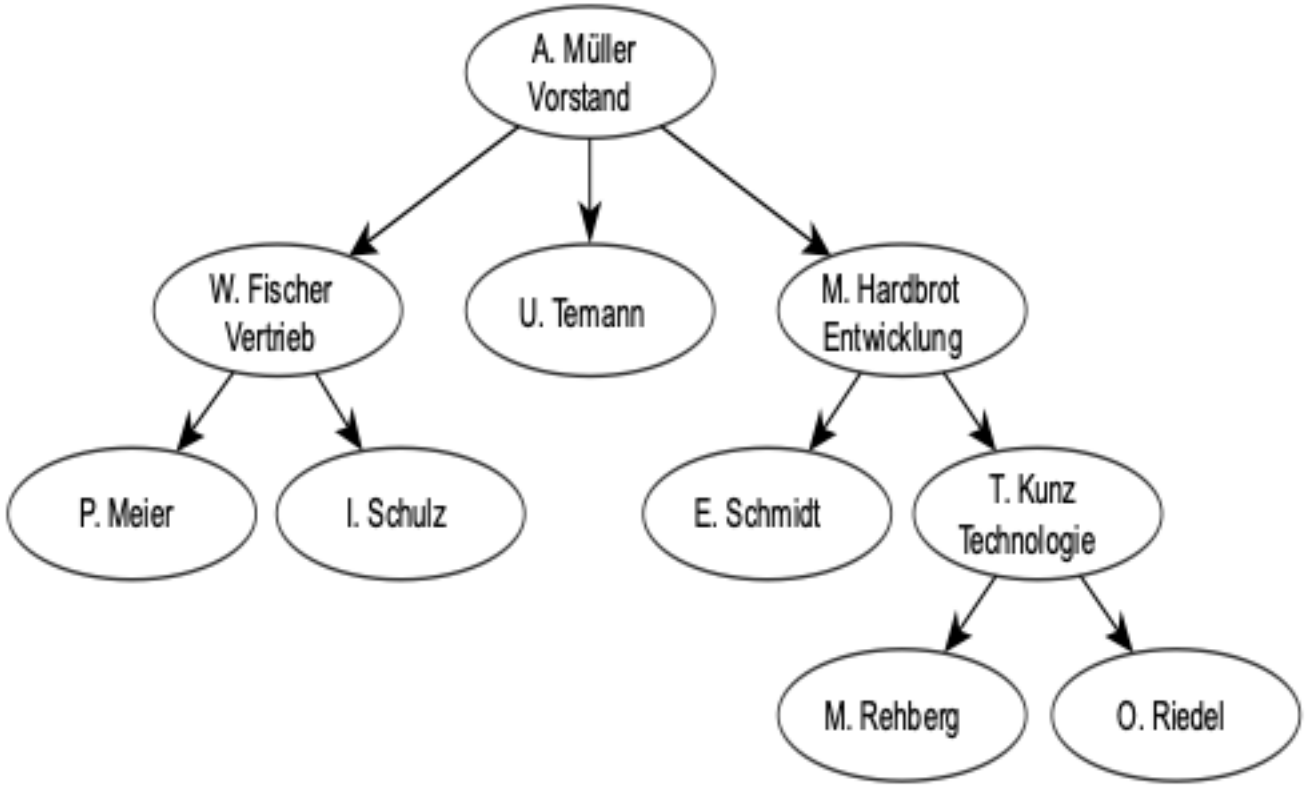
Graphics Example



Practical Example

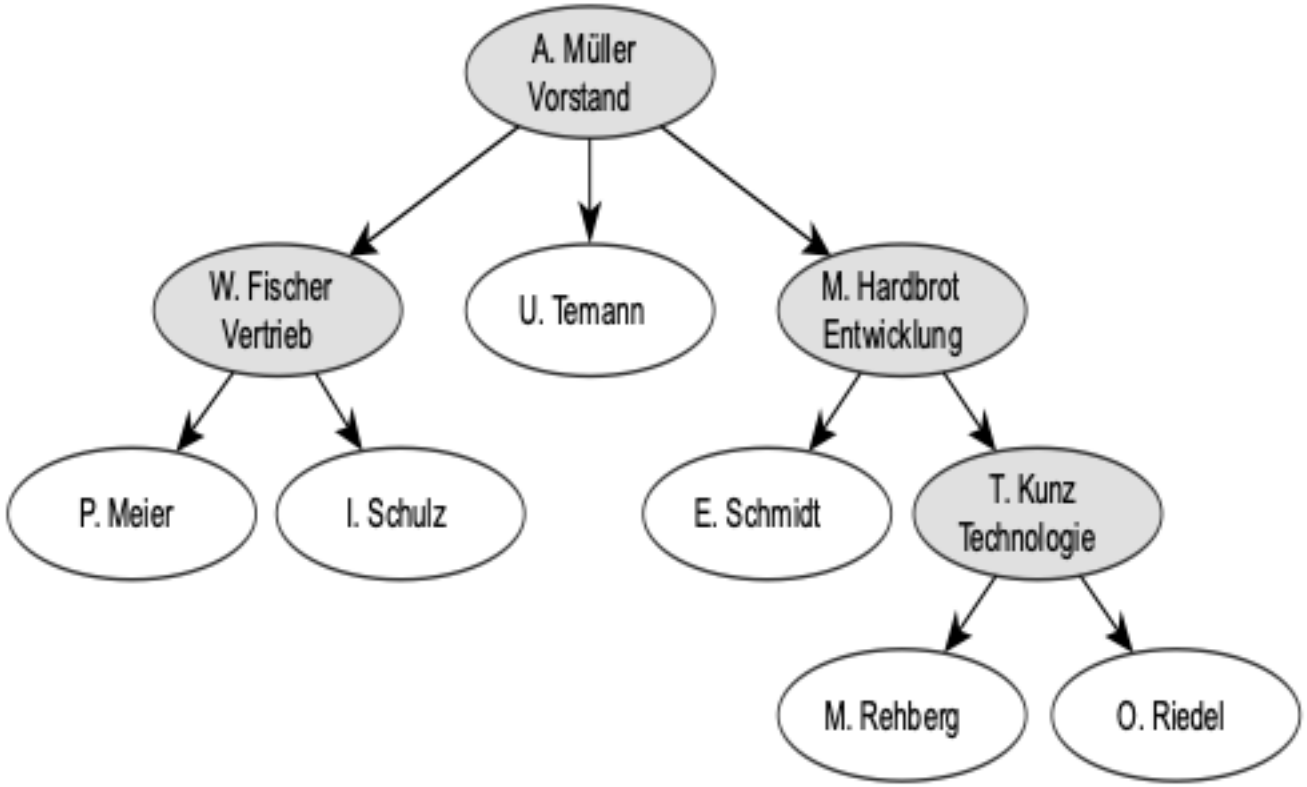
Describe the Problem

- We want to model the employee hierarchy of the company XY. The hierarchy tree looks like this



Describe the Problem

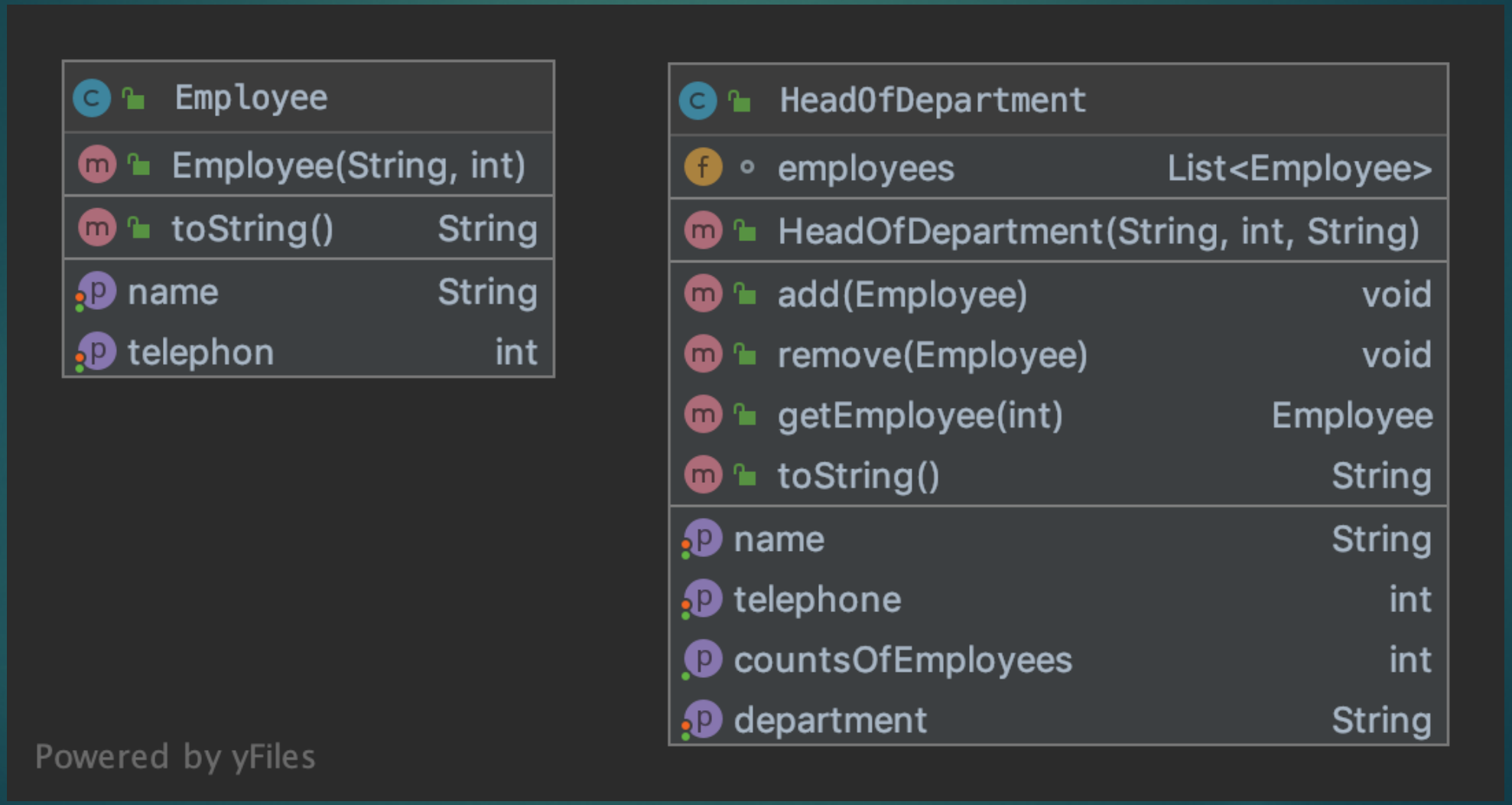
- Obviously, there are "normal" employees and department heads who have other employees (or department heads) under them.



What requirements are placed on our system?

- Each employee and departmental manager has a name and phone number (*getName ()*, *getTelefonnumber ()*).
- Operations are to be provided to place, retrieve, or remove employees in the tree (*add ()*, *getemployees ()*, *remove ()*).
- Each departmental head should be able to count the number of employees / department heads in his department (*getCountsOfEmployees()*).

Step 01 - Link


Employee
Employee(String, int)
toString() String
name String
telephon int
HeadOfDepartment
employees List<Employee>
HeadOfDepartment(String, int, String)
add(Employee) void
remove(Employee) void
getEmployee(int) Employee
toString() String
name String
telephone int
countsOfEmployees int
department String
Powered by yFiles

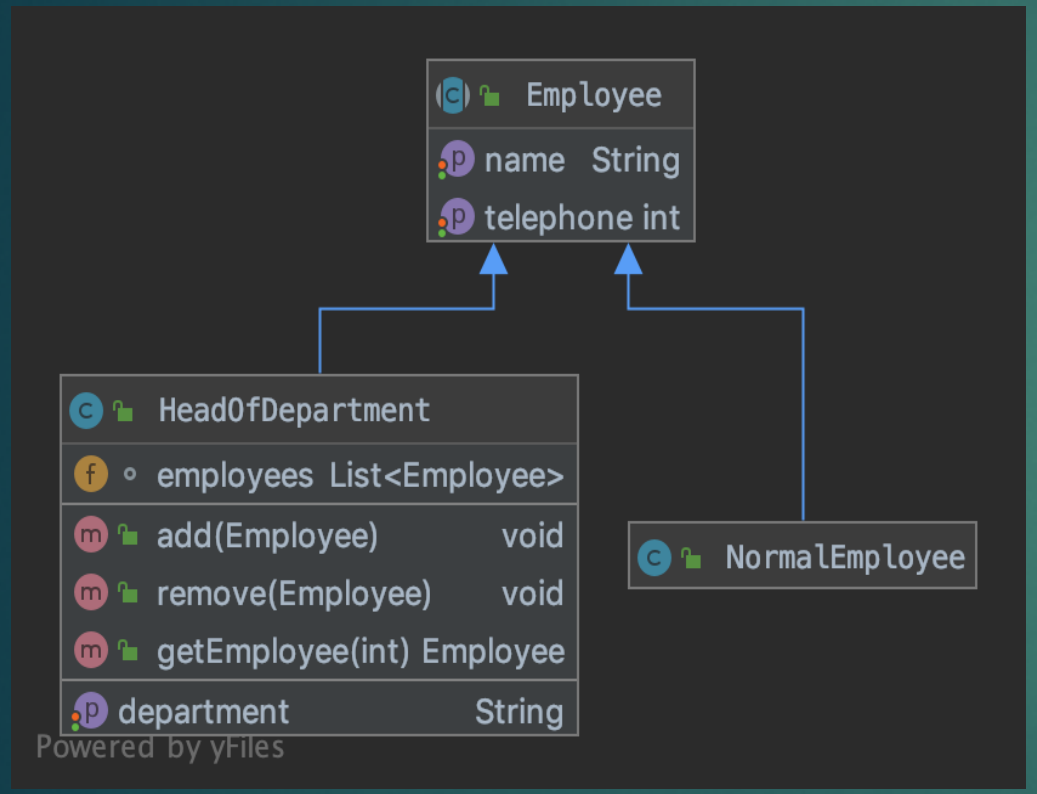
Wickedness of step 01

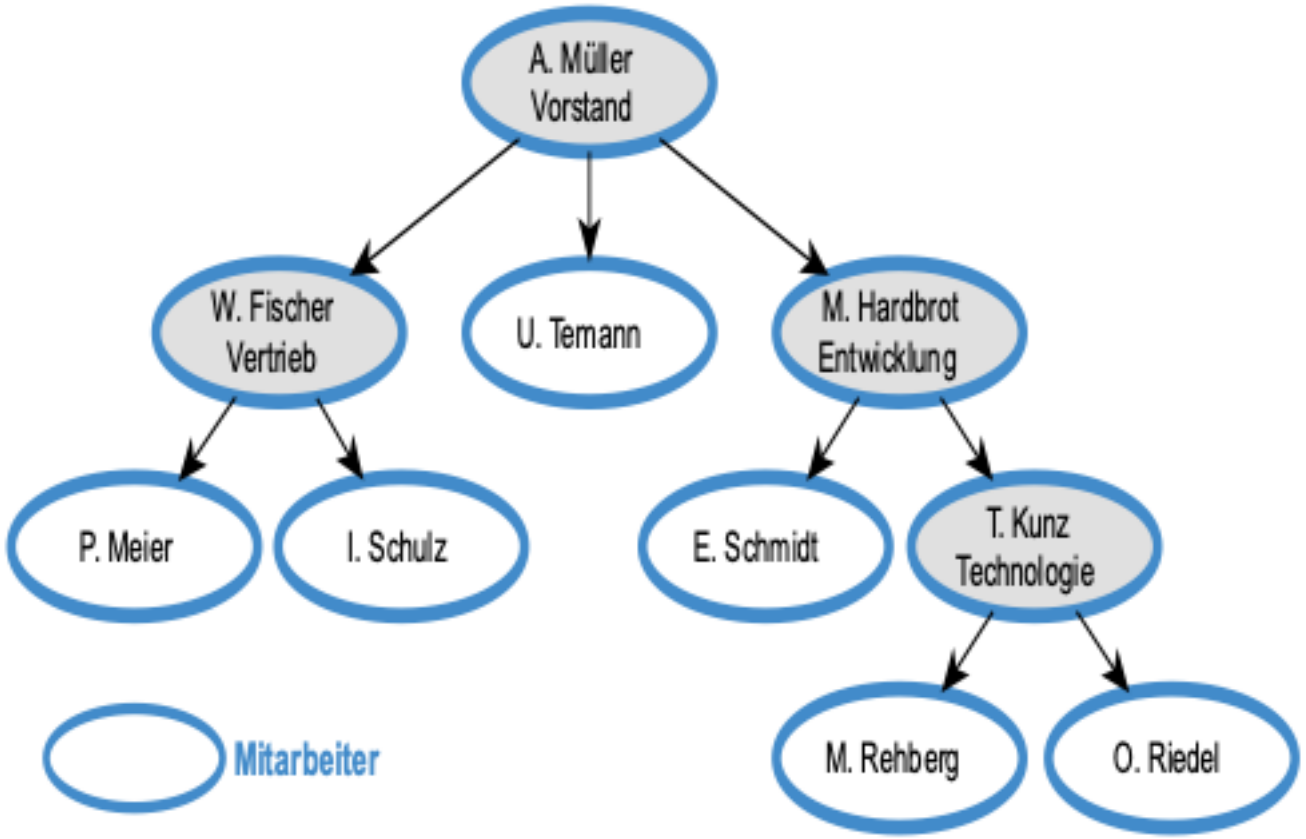
- before we fix the design, we want to look at its wickedness with all the disadvantages.
- Code duplication: The attributes for the name and phone number, as well as the associated getters and setters are completely redundant.
- Case **distinctions** and **tight coupling**. Both the client and the department manager must constantly distinguish between employees and department heads in their code. This creates a lot of hard-to-maintain code and prevents generality. Client and department heads are hard-wired to implementations.

Step 02 - Link

- First, we introduce abstraction in the form of superclass employees, inheriting both regular employees and department heads.
- From now on, every employee (whether a normal employee or a department manager) is also an employee - as in reality. This shows how the reality of OO modeling can help us.
- Employees who do not lead a department are called normal employees.

Step 02


Employee
name String
telephone int
HeadOfDepartment
employees List<Employee>
add(Employee) void
remove(Employee) void
getEmployee(int) Employee
department String
NormalEmployee
Powered by yFiles


A. Müller
Vorstand
W. Fischer
Vertrieb
U. Temann
M. Hardbrot
Entwicklung
P. Meier
I. Schulz
E. Schmidt
T. Kunz
Technologie
M. Rehberg
O. Riedel
Mitarbeiter

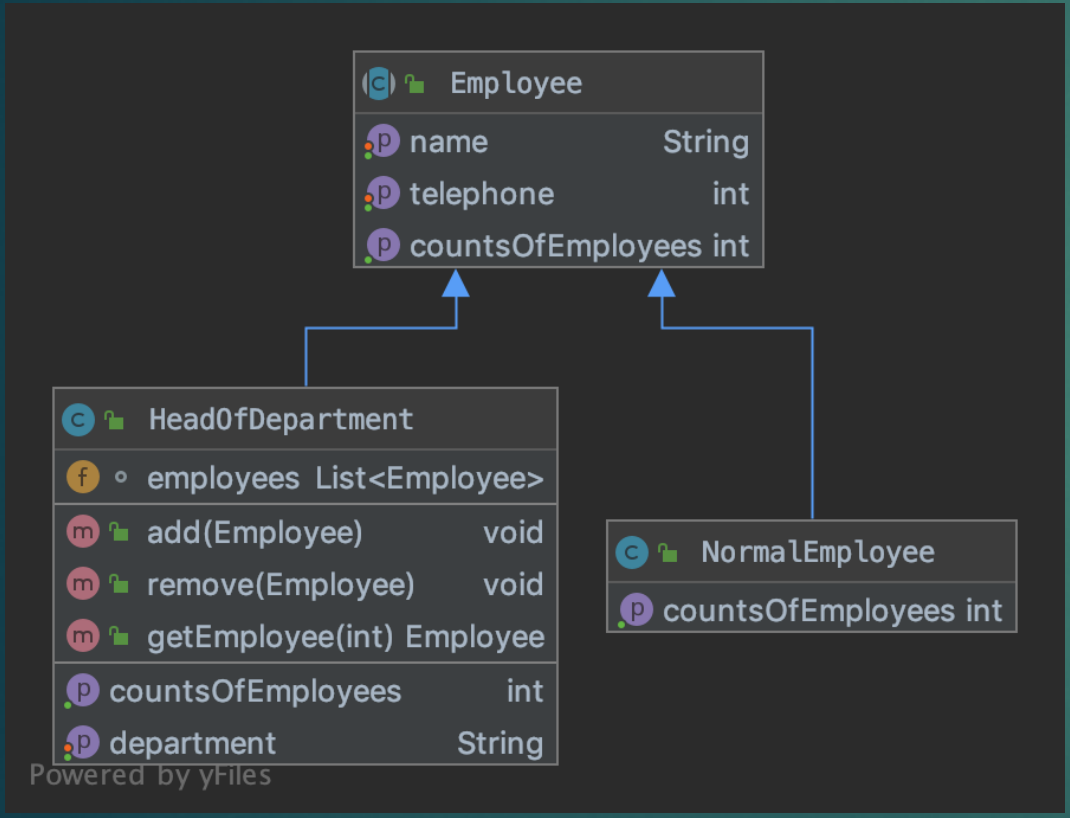
Implement *getCountsOfEmployees()*

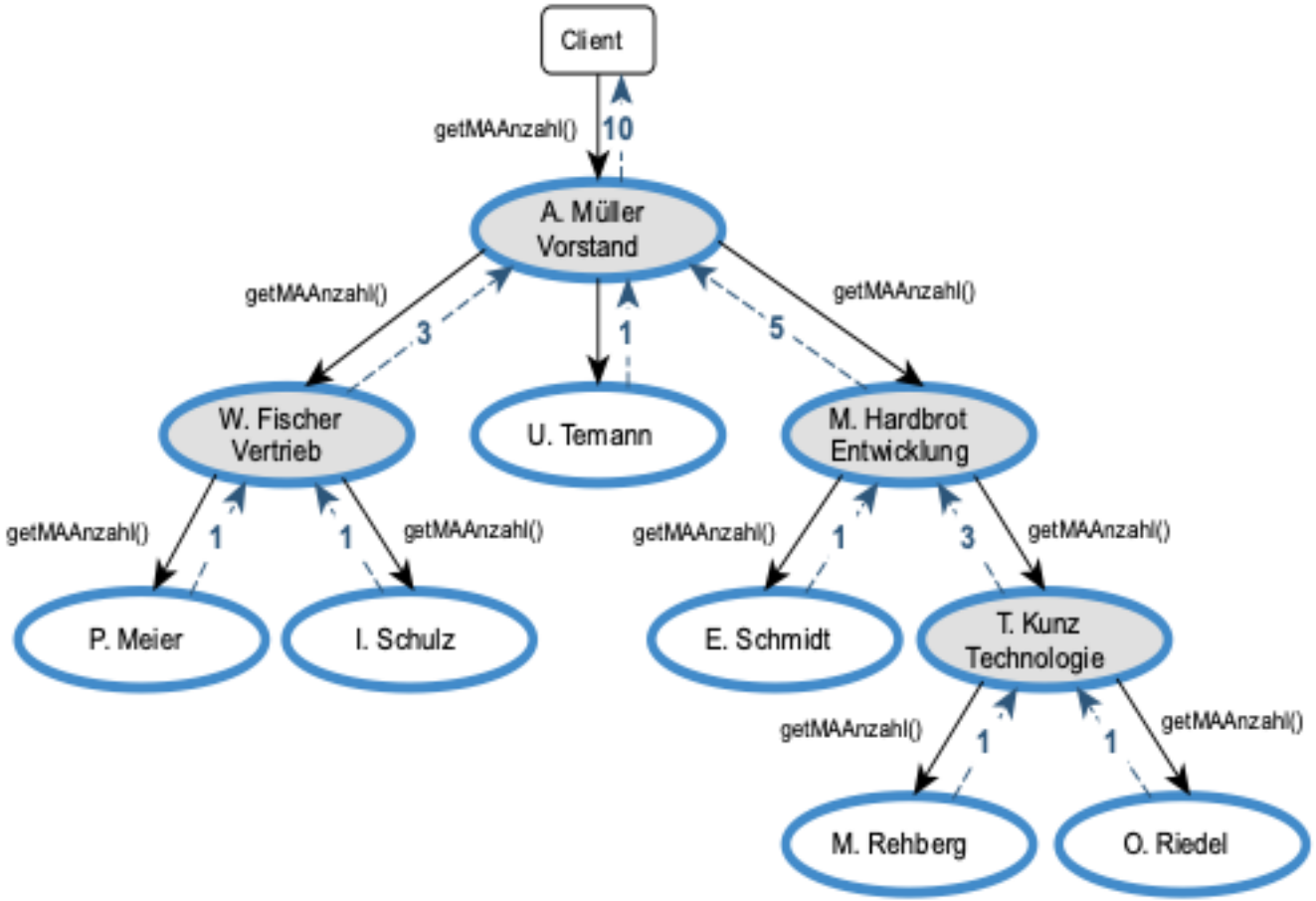
In the next step we want to implement the method *getCountsOfEmployees()*.

Show how elegantly these can be realized.

We extend the employee class by this method, and define *getCountsOfEmployees()* abstractly, forcing the subclasses to implement the method.

Implement *getCountsOfEmployees()*





Implement *getCountsOfEmployees()*

- What is the desired behavior?
- If you call *getCountsOfEmployees()* on a normal employee, it returns 1. Sure, he has no one among himself.
- Called on a department manager, the method should output the number of all its employees (plus 1 for itself). This includes subordinate department heads and their employees.
- What would be easier than to iterate over the list of employees and call on each employee *getCountsOfEmployees()* and cumulate the results?

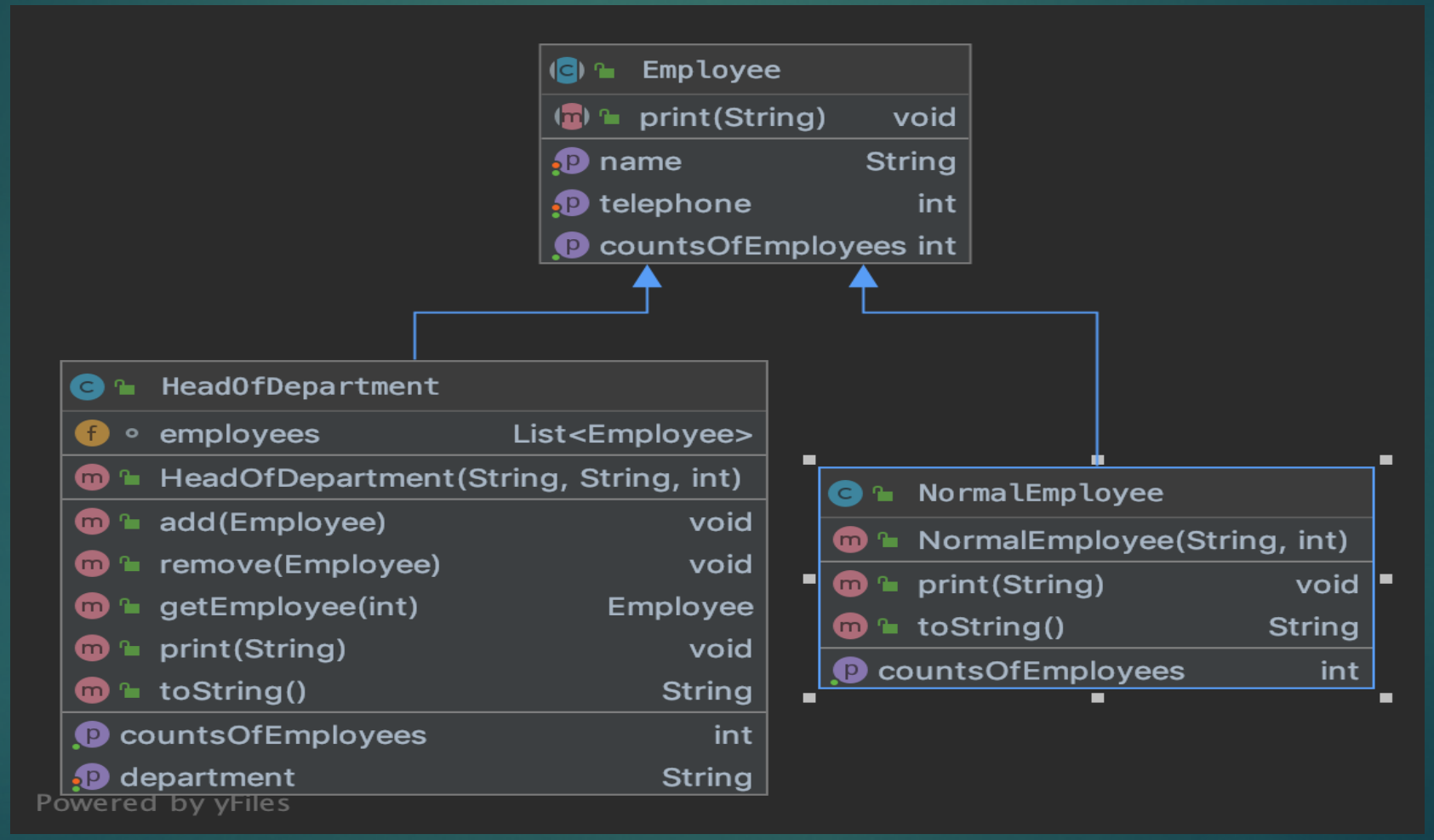
Implement *print()* method

Now for the last request - the textual printing of the entire hierarchy. The realization of this print () method is analogous to *getCountsOfEmployees()*.

The general employee class is extended by the abstract method *print()*.

Each employee delivers his name, phone number and department when calling. Of course, supervisors must also delegate the *print()* call to their subordinates.

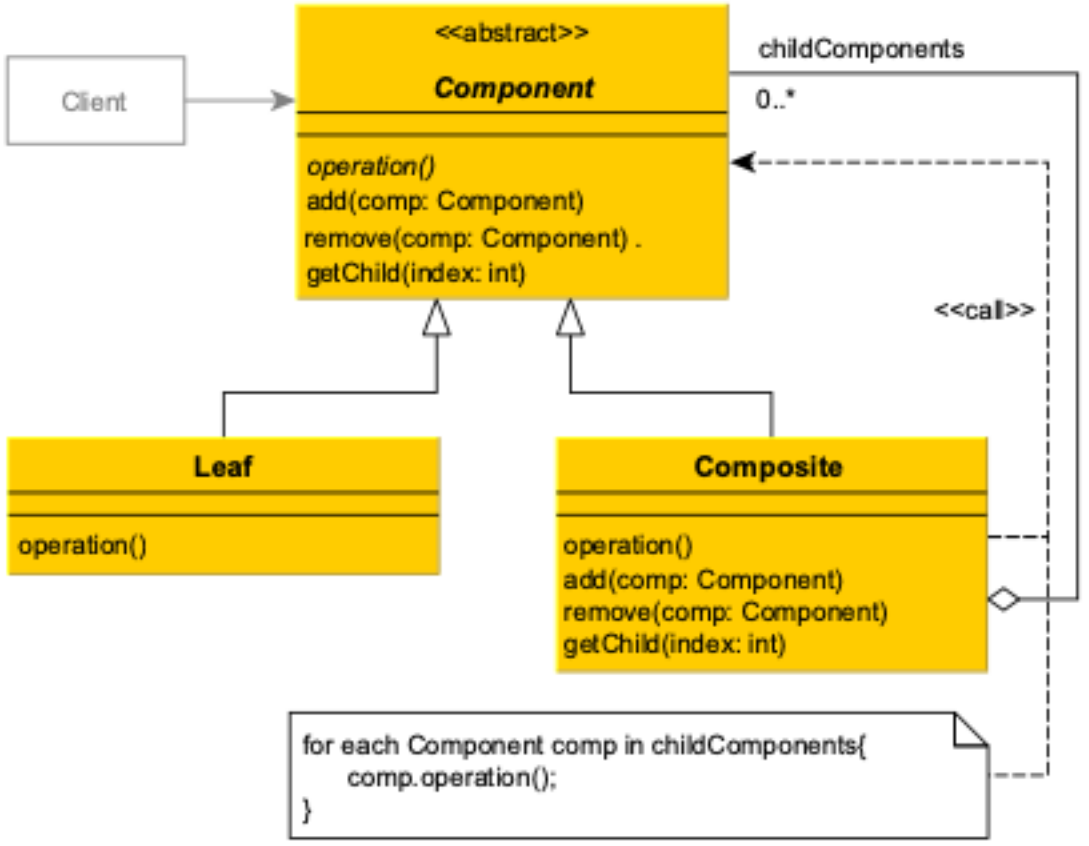
Implement *print()* method

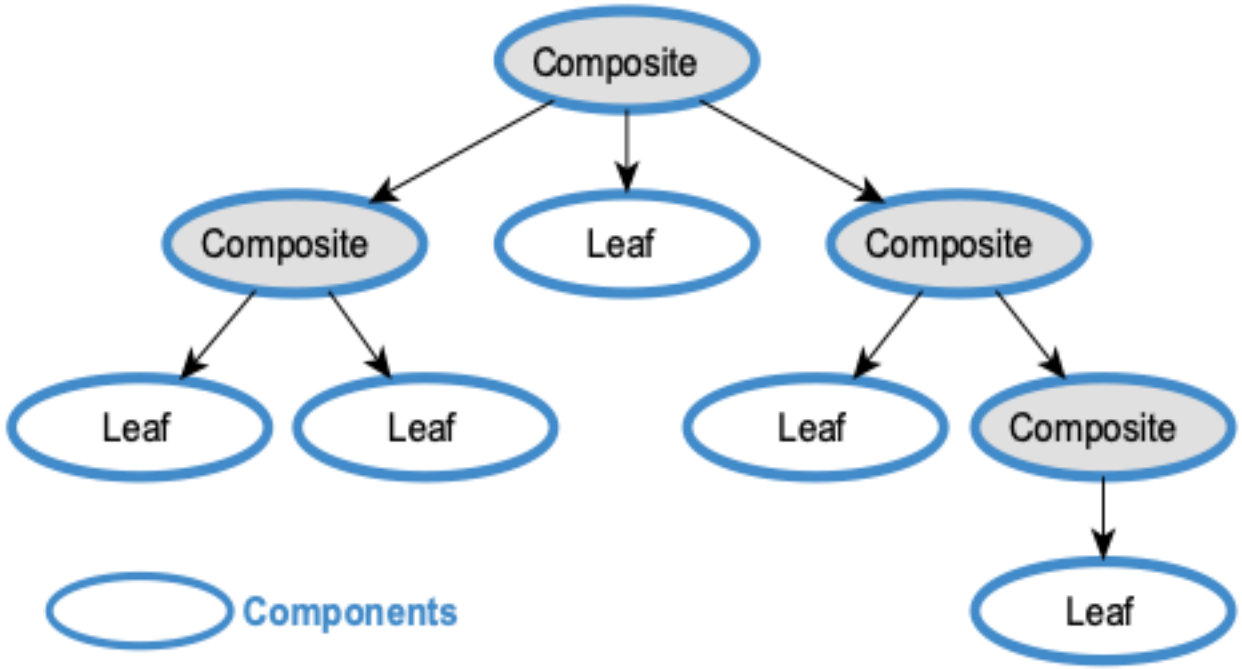


Composite Design Pattern Members

Klasse	Composite Member
Employee	Component
Normal Mitarbeiter	Leaf
HeadOfDepartment	Composite

General Structure - Link





AppRunner

```
/*
 * Build up a company hierarchy once
 */
//Vertrieb
HeadOfDepartment al1 = new HeadOfDepartment( name: "W. Fischer", department: "Vertrieb", telephone: 001);
al1.add(new NormalEmployee( name: "P. Meier", telephon: 123));
al1.add(new NormalEmployee( name: "I. Schulz", telephon: 112));
//Technologie
HeadOfDepartment al2 = new HeadOfDepartment( name: "T. Kunz", department: "Technologie", telephone: 002);
al2.add(new NormalEmployee( name: "M. Rehberg", telephon: 223));
al2.add(new NormalEmployee( name: "O. Riedel", telephon: 212));
//Entwicklung
HeadOfDepartment al3 = new HeadOfDepartment( name: "M. Hardbrot", department: "Entwicklung", telephone: 003);
al3.add(new NormalEmployee( name: "M. Rehberg", telephon: 323));
al3.add(al2);
//Vorstand
HeadOfDepartment XY = new HeadOfDepartment( name: "A. Müller", department: "Vorstand", telephone: 004);
XY.add(al1);
XY.add(new NormalEmployee( name: "U. Temann", telephon: 442));
XY.add(al3);

/*
 * Print company hierarchy
 */
XY.print("");
```

AppRunner

```
Head of Department A. Müller (Vorstand). Tel: 4
       Head of Department W. Fischer (Vertrieb). Tel: 1
              P. Meier. Tel: 123
              I. Schulz. Tel: 112
       U. Temann. Tel: 442
       Head of Department M. Hardbrot (Entwicklung). Tel: 3
              M. Rehberg. Tel: 323
              Head of Department T. Kunz (Technologie). Tel: 2
                     M. Rehberg. Tel: 223
                     O. Riedel. Tel: 212
```

Resource

- https://de.wikipedia.org/wiki/Kompositum_(Entwurfsmuster)
- https://www.javatpoint.com/composite-pattern
- https://www.philipphauer.de/study/se/design-pattern/composite.php
- TU Dresden - SWT SS19