Einführung in die Informatik

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Sommersemester 2014

Teil VI

Objektorientierte Programmierung

Programmierparadigmen

Im Laufe des letzten Jahrhunderts wurden verschiedenen Programmierparadigmen vorgeschlagen:

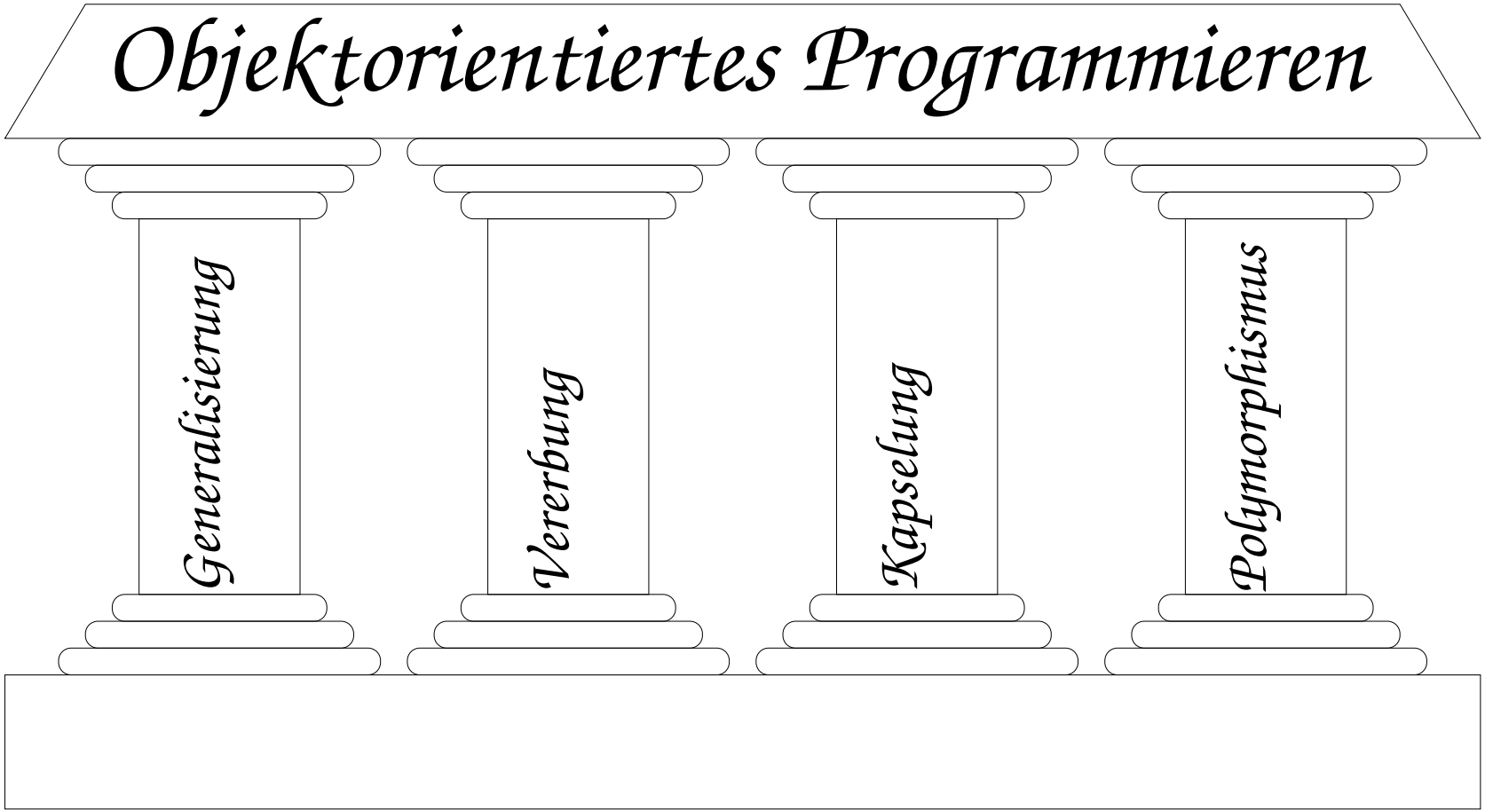
- *Imperative Programmierung*: Hintereinander Ausführung von Befehlen, Schleifen. Zur Strukturierung werden Teile in eigene Methoden ausgelagert.
- *Logische Programmierung*: Das Problem wird durch logische Formeln beschrieben. Die Programmiersprache enthält Automatismen zum Lösen der Formeln mit Lösungssuche.
- *Funktionale Programmierung*: Das Programm wird in Funktionen geschrieben. Diese sollten nach Möglichkeit wenig oder keine Seiteneffekte haben. Es gibt keine Variablen.
- *Objektorientierte Programmierung*: Das Programm wird in Klassen unterteilt. Die Objekte dienen nicht nur als Datenspeicher, sondern haben eigene „Intelligenz“.

Philosophie der Objektorientierung

Everything is an object — *Alan Kay, Erfinder von Smalltalk*

- Objekte speichern die Daten (in Form von Komponenten).
- Objekte enthalten Programmcode (in Form von Methoden).
- Objekte kommunizieren miteinander (in Form von Methodenaufrufen).
- Objekte sind Instanzen von Klassen.

Die Grundpfeiler der Objektorientierten Programmierung



Generalisierung / Abstraktion

Objekte werden in Klassen kategorisiert.

Beispiel: Dateien

Auf einem Rechner liegen viele Dateien. Auch wenn sie alle verschieden sind, haben sie Gemeinsamkeiten.
Dateien enthalten eine Folge von Zeichen. Sie können gelesen oder geschrieben werden.
Die Dateien sind Objekte, die zur selben Klasse (Datei) gehören.

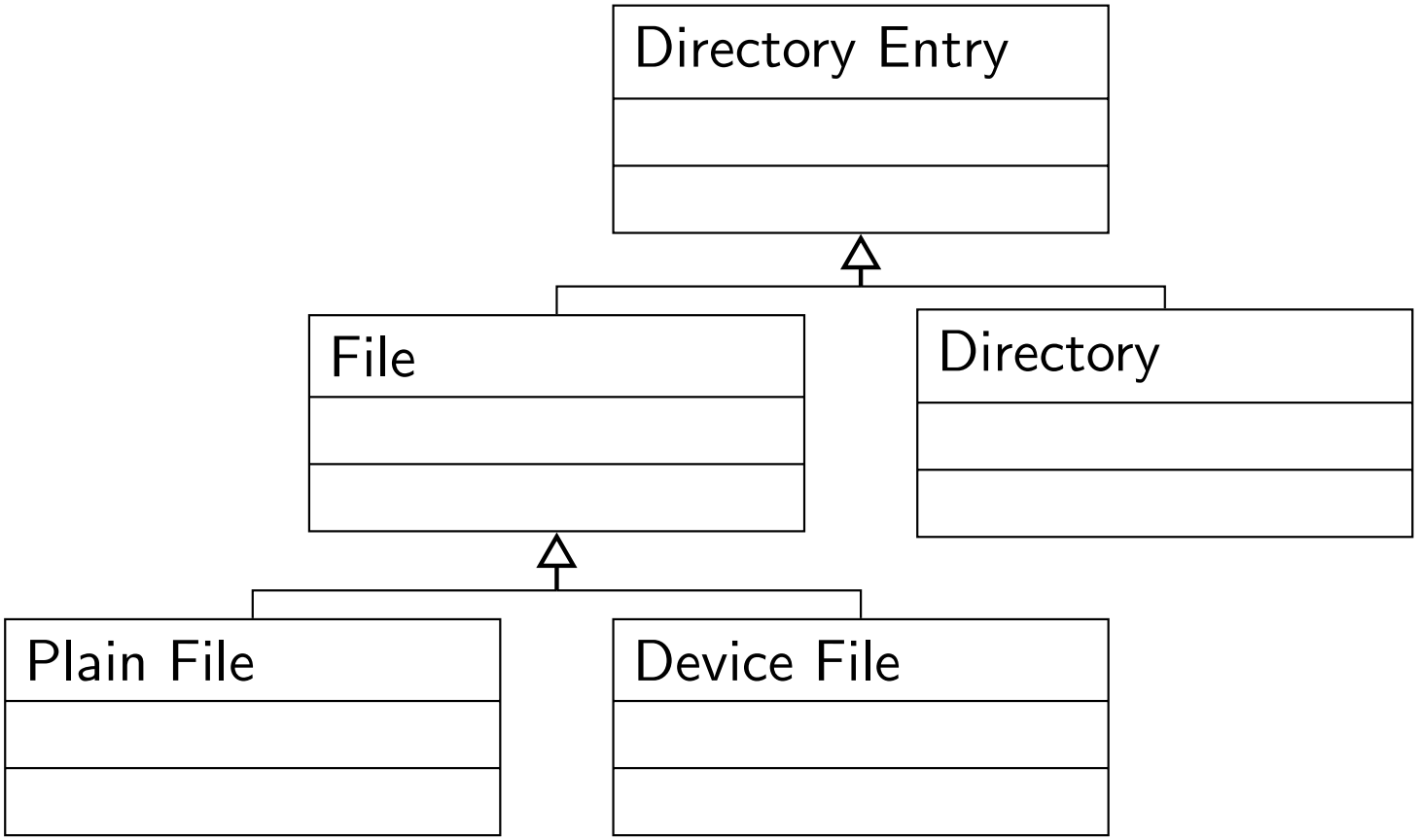
Kapselung

Kapselung verbietet anderen Programmteilen Zugriff auf das Innenleben

- Wohldefinierte Schnittstelle erleichtert die Zusammenarbeit in großen Programmierprojekten.
- Innenleben kann später geändert werden ohne die anderen Programmteile ändern zu müssen.
- Mehrere Objekte können die gleiche Schnittstelle aber verschiedenes Innenleben haben.

Vererbung

Klassen können hierarchisch geordenet werden. Subklassen erben Attribute von Superklassen.



Polymorphismus

Polymorphie (wörtlich: Vielgestalt) erlaubt es verschiedenen Objekten, die gleiche Schnittstelle aber verschiedenes Verhalten zu haben.

- eine gemeinsame Superklasse definiert die Schnittstelle in Form einer Methode. Sie kann auch ein Standardverhalten vorgeben.
- Die Subklassen können die Methode überschreiben (Englisch: override), um ihr eigenes Verhalten zu definieren.

Klassen

Die Objekte werden in Klassen kategorisiert.

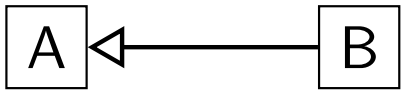
- Eine Klasse definiert die Daten (Komponenten), aus denen die Objekte zusammengesetzt sind.
- Eine Klasse definiert auch Methoden, um diese Daten zu manipulieren (die Schnittstelle der Objekte).
- Alle Objekte einer Klasse haben die gleichen Komponenten und die gleichen Methoden, aber verschiedene Datenwerte.
- Die Daten und Methoden können öffentlich (public) oder privat (private) sein. Die interne Struktur soll privat sein, um eine Kapselung zu erreichen.

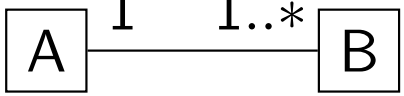
Klassendiagramme

Um Objekte in Klassen zu kategorisieren und Abhängigkeiten zu visualisieren bieten sich Klassendiagramme aus der UML an.

- Eine Klasse wird dargestellt durch eine Box

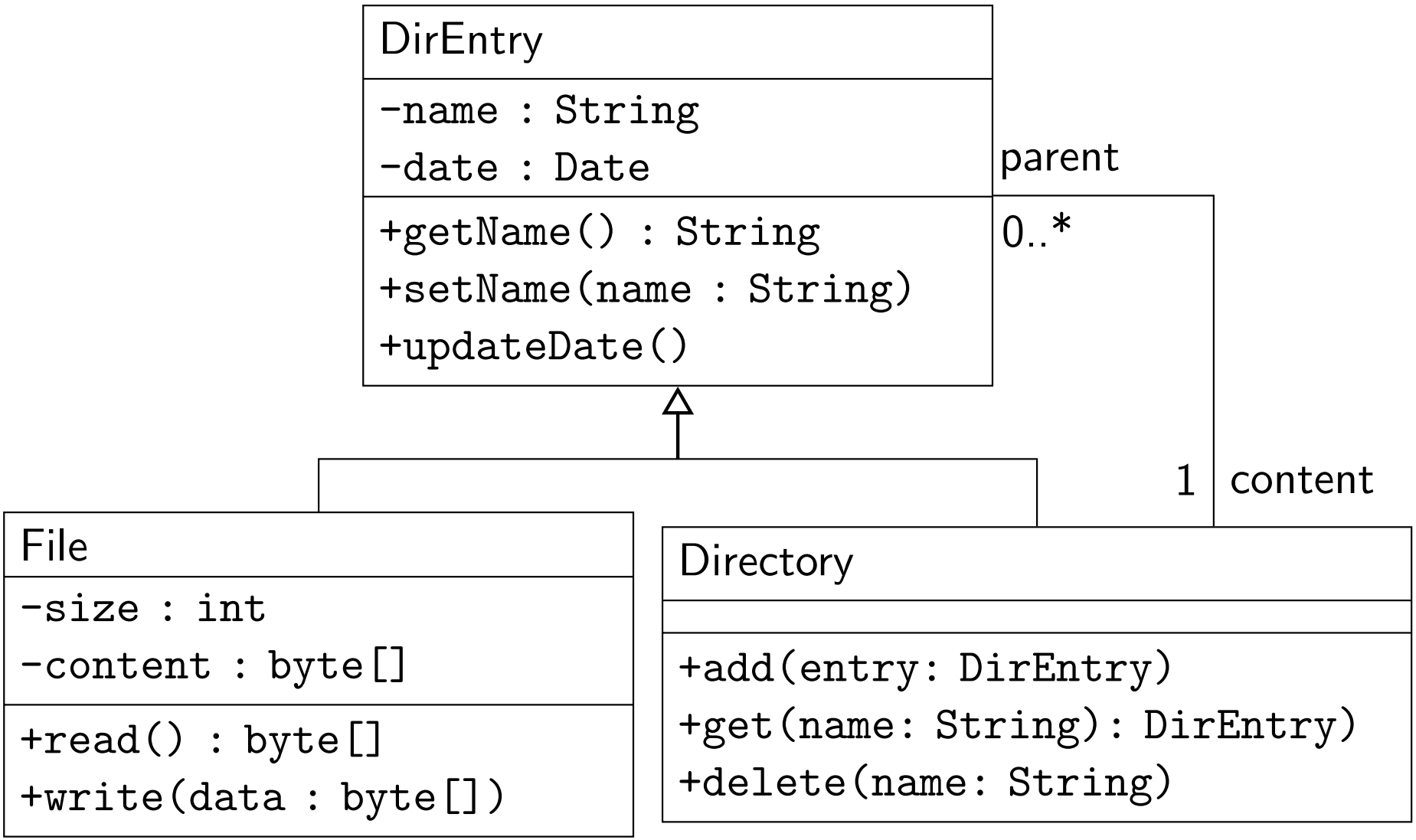
KlassenName
`-komponente: int`
`-interneMethode() : int` `+methode(param : int): int`

- Mit dem Pfeil 

wird eine Vererbung angezeigt. Klasse B erbt von Klasse A.
- Mit einer Kante 

wird eine Relation zwischen Klassen angezeigt. Einem A-Objekt ist mindestens ein B-Objekte zugeordnet; einem B-Objekt ist genau ein A-Objekt zugeordenet.

Beispiel: Klassendiagramm



Objektdiagramme

Zu einer Klasse kann es viele Objekte geben.

Beispiel

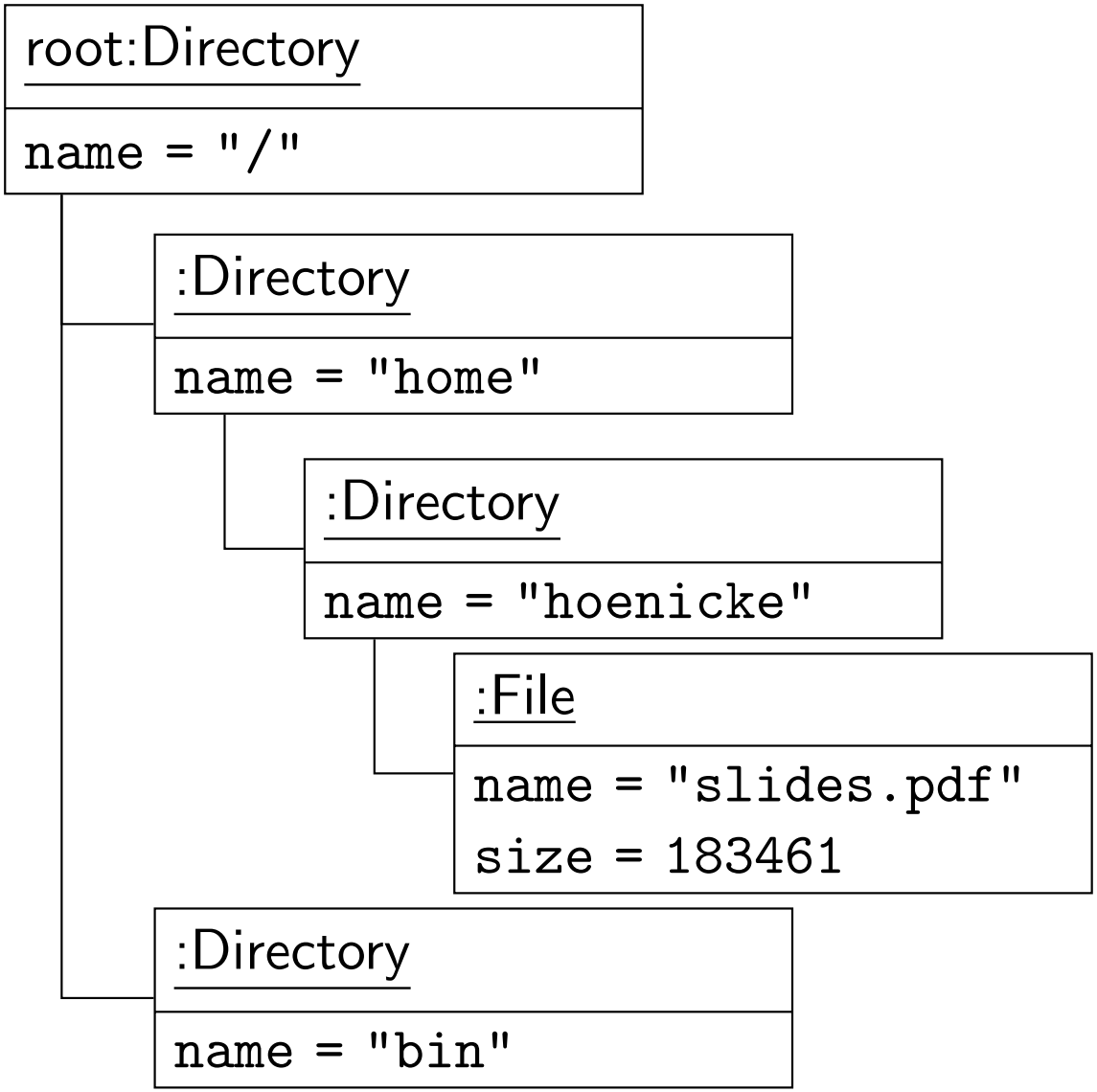
Die Klasse `File` hat viele Objekte, nämlich eins für jede Datei.

Objekte werden durch einen Doppelpunkt und einem Unterstrich gekennzeichnet:

objektName: KlassenName
`komponente = 5`

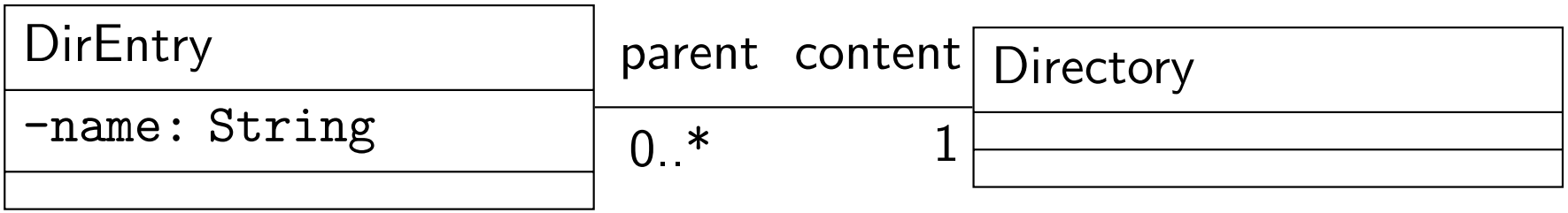
Im Gegensatz zu Klassen müssen Objekte nicht immer einen Namen haben.

Beispiel: Objektdiagramm



Relationen

Relationen lassen sich in Java nicht direkt ausdrücken.



Man kann Sie durch Attribute (Komponenten) modellieren, zum Beispiel:

Allerdings müsste man noch dokumentieren, dass für ein Directory `dir` gilt: `dir.content[i].parent == dir`.
Manchmal braucht man auch die eine Richtung der Relation nicht und kann sie weglassen.

Klassen in Java

Java-Klassen

```
public class ClassName {
    private String component1;
    private int component2;

    private String internalMethod() {
        ...
    }
    public int publicMethod() {
        ...
    }
}
```

- Achtung: kein `static` benutzen.
- Mit den Schlüsselwörtern `private` oder `public` wird angezeigt, ob die Komponente bzw. Methode privat oder öffentlich ist.

Instanzmethoden

Eine Instanzmethode ist eine Methode, die nicht mit `static` gekennzeichnet ist.

```
public class DirEntry {
    private String name;

    public void setName(String n) {
        this.name = n;
        // alternativ:  name = n;
    }
}
```

- Eine Instanzmethode ist in einer Klasse definiert,
- gehört aber zu einem Objekt.
- Mit `this` kann die Methode auf dieses Objekt zugreifen.
- Wenn es eindeutig ist, kann `this.` auch weggelassen werden.

Aufruf von Instanzmethoden

Auf die Instanzmethode eines Objektes wird ebenfalls mit der Punktnotation zugegriffen.

```
DirEntry entry = someEntry;
entry.setName("home");
```

Hier ist `entry.setName` die Instanzmethode `setName` des Objektes `entry`.

Wenn man aus eine Instanzmethode von einer anderen des gleichen Objekts aufrufen will, kann man `this` benutzen:

```
this.setName("abc");
```

Man kann `this.` auch weglassen.

Beispiele für Methodenaufrufe

Wir haben diese Notation bereits kennengelernt, ohne darauf näher einzugehen:

- `System.out.println("Hello")` ruft die Methode `println` des Objektes `System.out` auf. Letzteres Objekt entspricht der Standardausgabe (normalerweise Konsolenausgabe).
- `"abc".toUpperCase()` ruft die Methode `toUpperCase` einer Zeichenkette auf. In der Tat ist auch `String` eine Klasse und alle Zeichenketten sind Objekte.

Validierung durch Instanzmethoden

Instanzmethoden können benutzt werden, um Felder zu setzen und dabei auf gültige Werte zu validieren:

```
/**
 * Set the file name.
 * @param n the new filen name.
 * @returns true if successful,
 *          false otherwise.
 */
public boolean setName(n : String) {
    // Do not allow '/' in a file name.
    if (n.indexOf('/') != -1)
        return false;
    this.name = n;
    return true;
}
```

Rekursive Instanzmethoden

Auch Instanzmethoden können rekursiv sein.

```
public class DirEntry {
    String name;
    DirEntry parent;
    public String getAbsolutePath() {
        if (parent == null)
            return "/";
        return parent.getAbsolutePath()
                + "/" + name;
    }
}
```

Zwei Sichten auf Instanzmethoden

Es gibt zwei Sichtweisen:

1. Jedes Objekt einer Klasse hat seine eigene Instanzmethode. Diese unterscheiden sich dadurch, dass `this` ein anderes Objekt bezeichnet.
2. Es gibt nur eine Methode, nämlich die der Klasse. Das Objekt `this` ist nur ein versteckter Parameter der Methode. Beim Methodenaufruf schreibt man diesen Parameter vor den Methodennamen statt in die Klammern.

Ohne Polymorphie (später) gibt es keine Möglichkeit, diese Sichtweisen zu unterscheiden. Man kann sich aussuchen, welche man intuitiver findet.

Statische Komponenten und Methoden

statische Komponenten und Methoden

Was bedeutet das Schlüsselwort `static`?

```
public class Test {
    public static int wert;

    public static void main(String[] param) {
        wert = 3 + 4;
        System.out.println(wert);
    }
}
```

- Eine statische Komponente gehört nicht zum Objekt, sondern zur Klasse.
- Wenn es mehrere Objekte gibt, hat die Komponente nur einen Wert.
- Eine statische Methode kennt kein `this`.

Lesen und Schreiben von statischen Feldern

Folgender Programmcode

```
Test objekt = new Test();
objekt.wert = 5;
```

hat den gleichen Effekt wie:

```
Test.wert = 5; // besser!
```

- Die statische Komponente `wert` ist eine Komponente von der Klasse `Test`, nicht von dem Objekt.
- Den Ausdruck `Test.wert`, kann man überall benutzen um diese Komponente zu lesen oder zu schreiben.
- In der Klasse `Test` kann man auch kurz `wert` schreiben.

Aufrufen von statischen Methoden

Eine statische Methode kann ebenfalls von überall aufgerufen werden:

```
Test.main(new String[4]);
```

- Diese Methode ist keinem Objekt zugeordnet, daher gibt es kein `this`.
- Man kann auf statische Komponenten und Methoden normal zugreifen.
- Man kann keine Instanzmethoden aufrufen (außer man hat ein Objekt).
- Aus einer Instanzmethode kann man eine statische Methode ohne Klassennamen aufrufen.

Beispiele für statische Methoden und Komponenten

- Alle Methoden im ersten Teil der Vorlesung waren statisch!
- Die Bibliotheksklasse `Math` definiert viele nützliche statische Funktionen. Zum Beispiel, `Math.pow(double x, double y)`, um Potenzen zu berechnen.
- `Math.PI` ist ein Beispiel für eine statische Komponente, die (eine Approximation von) π enthält. Die Komponente ist auch als `final` markiert, d.h., sie darf nicht überschrieben werden.
- `System.out` ist eine statische Komponente (ebenfalls konstant).
- `IOTools.readInteger()` ist eine statische Methode.

Wann benutzt man statische Methoden/Komponenten?

In folgenden Fällen benutzt man statische Methoden/Komponenten:

- Die `main`-Methode muss statisch sein, da beim Programmstart noch kein Objekt existiert.
- Für Hilfsfunktionen, die kein Objekt benötigen (z.B. die Methoden in `Math`.
- Um Konstanten wie `Math.PI` zu definieren.

Man sollte nach Möglichkeit keine nicht konstanten statischen Felder benutzen.