Software-Entwicklung 1

V06: UML und Syntaktische Strukturen

Prof. Maalej & Team @maalejw

Überblick

1 **UML**

2 **Syntaktische Strukturen**

Die UML als Notation und Technik

- Bei Analyse, Modellierung und Programmierung benutzen wir eine einheitliche Notation - die Unified Modeling Language (UML)
- UML ist
  - eine Sammlung von **Diagrammtypen** und Modellierungstechniken, die ursprünglich aus 3 objektorientierten Methoden zusammengestellt wurde
  - heute ein Quasi-Standard für die Darstellung von objektorientierten Modellen
- UML wurde ursprünglich von einer Firma (Rational) entwickelt, wird aber jetzt von einem weltweiten Konsortium (OMG) betreut.

http://www.omg.org/technology/documents/formal/uml.htm

OMG™ is an international, open membership, not-for-profit computer industry consortium. OMG Task Forces develop enterprise integration standards for a wide range of technologies, and an even wider range of industries. OMG's modeling standards enable powerful visual design, execution and maintenance of software and other processes.

Unified Modeling Language

- The UML is a language for
  - visualizing…
  - specifying…
  - constructing…
  - documenting…

…the artifacts of a software-intensive system

Die UML ist eine Sprache, um die Elemente eines software-intensiven Systems zu

- visualisieren
- spezifizieren
- konstruieren
- dokumentieren

RATIONAL SOFTWARE

Diagrammtypen der UML

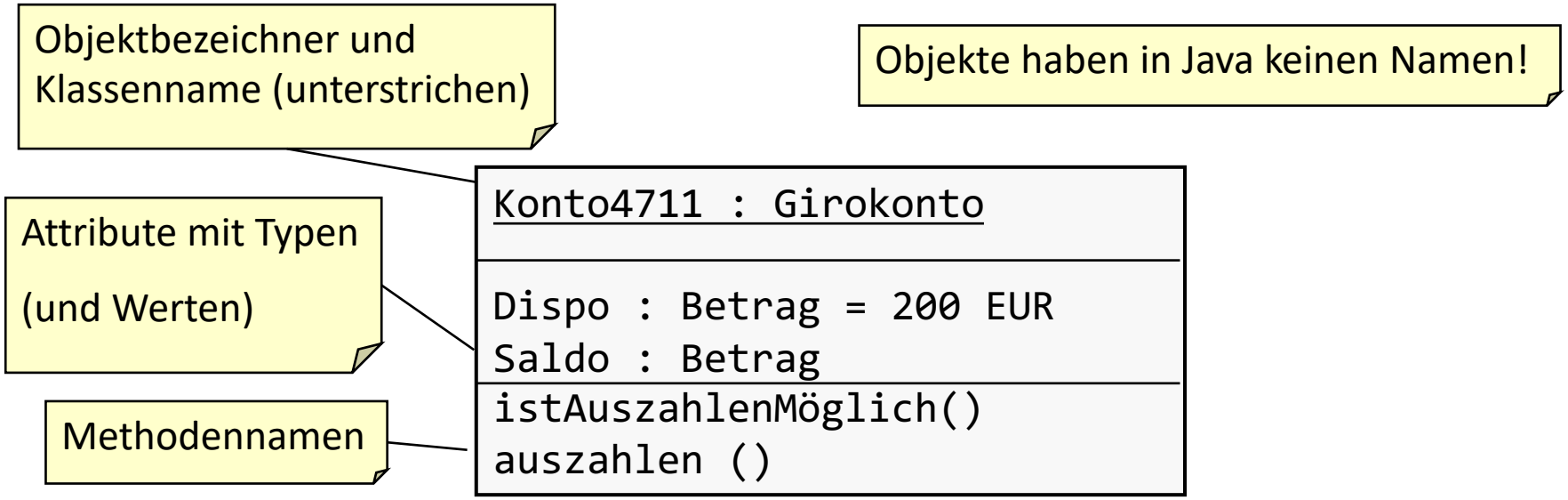
Unser Fokus in SE1

- Structure Diagrams:
  - Class Diagram
  - Object Diagram
  - Composite Structure Diagram (2.0)
  - Component Diagram
  - Deployment Diagram
  - Package Diagram

- Behavior Diagrams:
  - Activity Diagram
  - Use Case Diagram
  - State Machine Diagram
- Interaction Diagrams:
  - Sequence Diagram
  - Communication Diagram
  - Interaction Overview Diagram (2.0)
  - Timing Diagram (2.0)

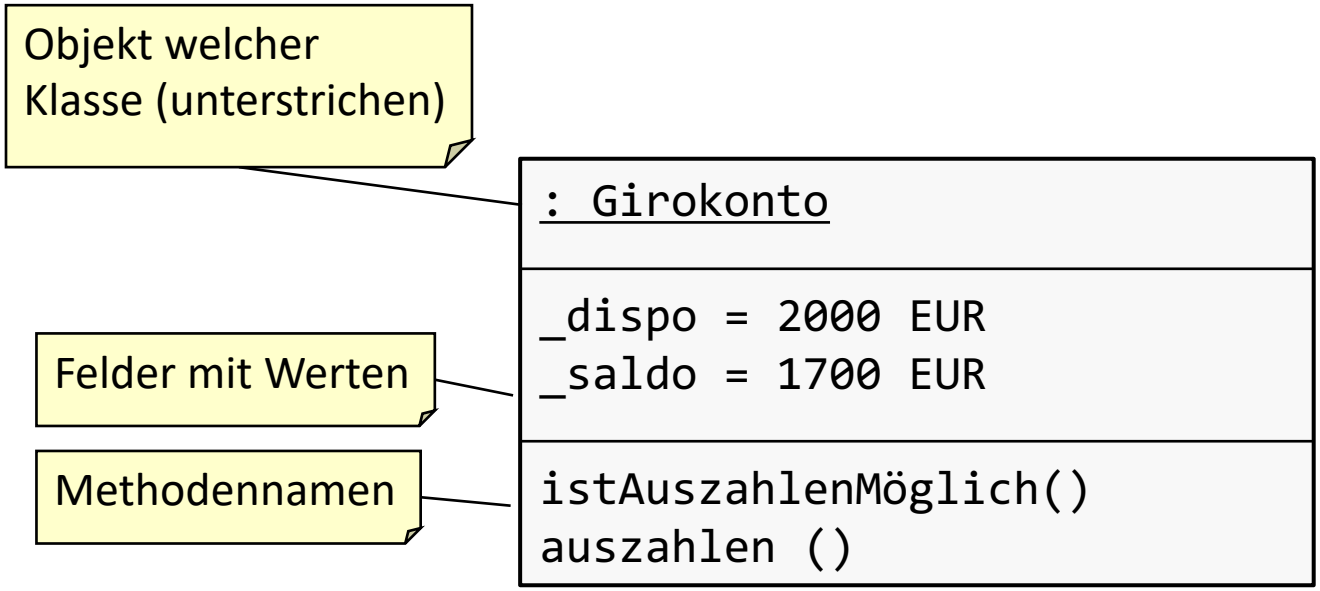
OMG-Unified Modeling Language, v2.0

Objektdiagramm, formal korrekt



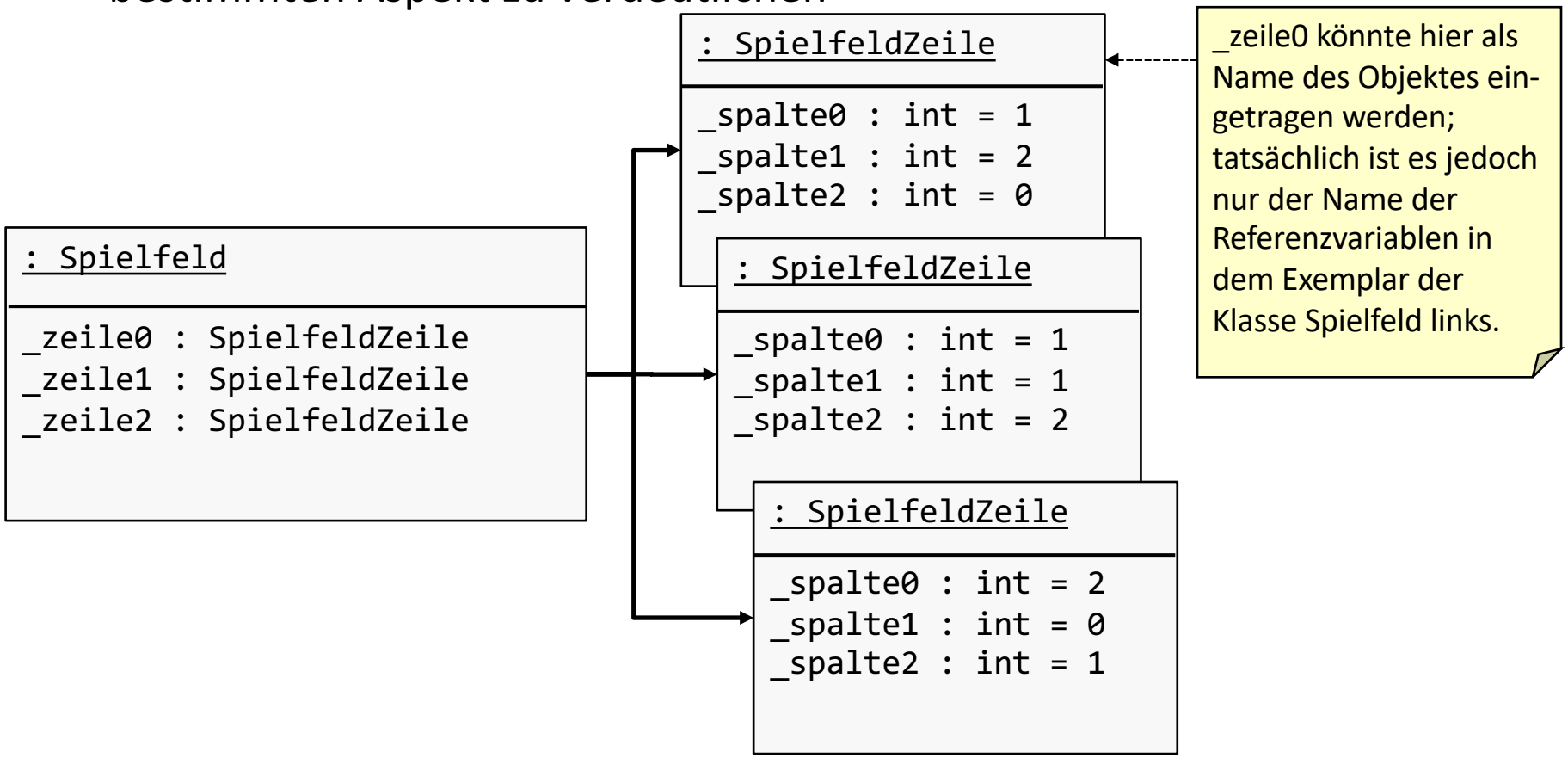
- Entweder der Objektbezeichner oder der Klassenname dürfen weggelassen werden; fehlt der Objektbezeichner, muss ein Doppelpunkt vor dem Klassennamen stehen
- Felder heißen in der UML Attribute; bei ihnen kann der Typ oder der konkrete Wert weggelassen werden
- Methodennamen können weggelassen werden

Objektdiagramm, pragmatisch



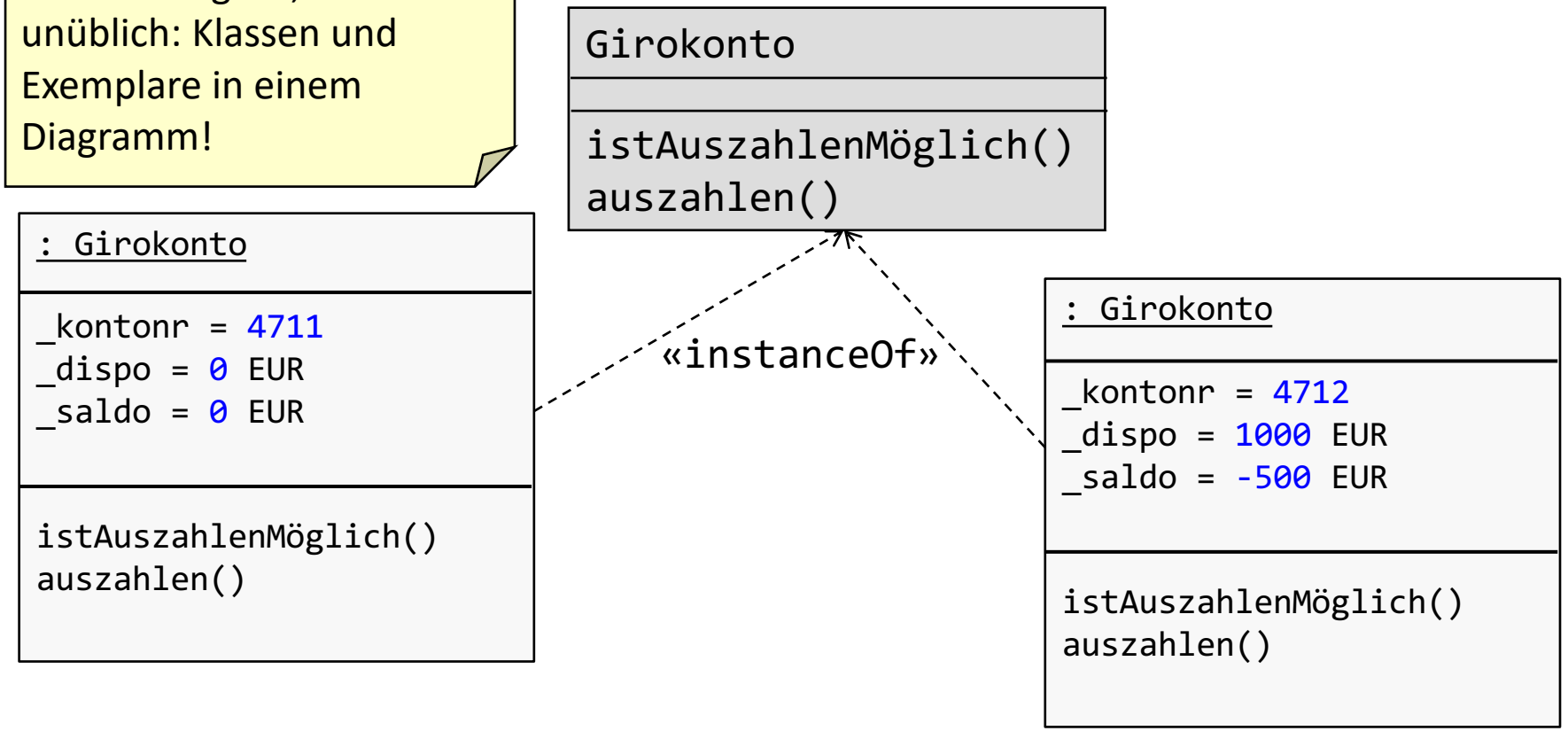
Objektdiagramme liefern Schnappschüsse

- Ein Objektdiagramm ist ein Schnappschuss eines laufenden Programms
- Es zeigt nur einen Ausschnitt des Objektgeflechts zur Laufzeit, um einen bestimmten Aspekt zu verdeutlichen



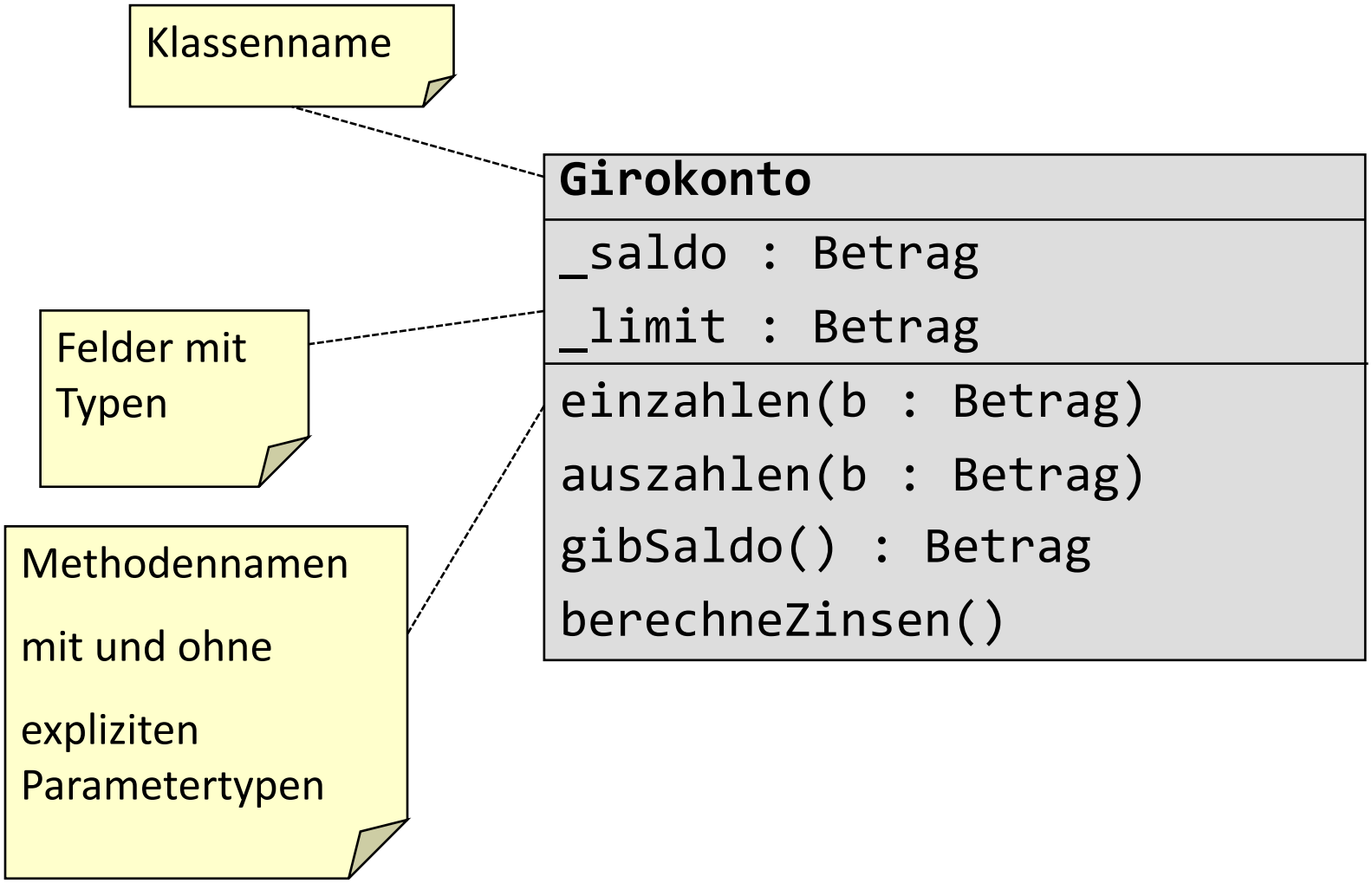
Objekte sind Exemplare von Klassen

In UML möglich, aber unüblich: Klassen und Exemplare in einem Diagramm!

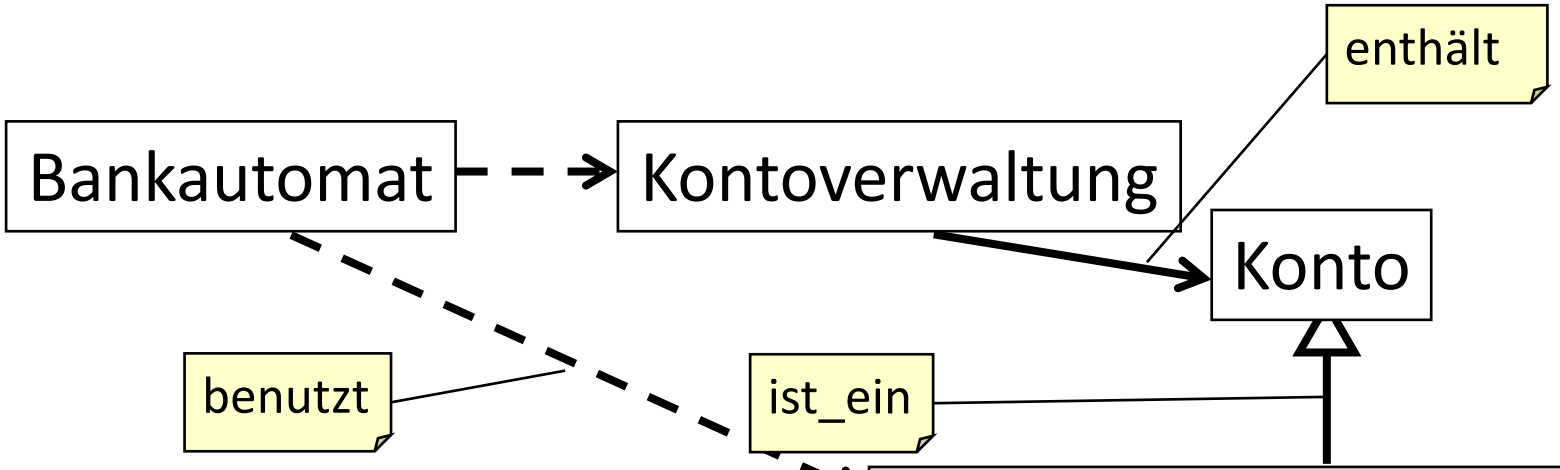


Die Klasse legt die Initialisierung, das Verhalten und die Struktur jedes Exemplars fest. Aber jedes Exemplar kann einen eigenen Zustand haben.

Klassendiagramme (1)



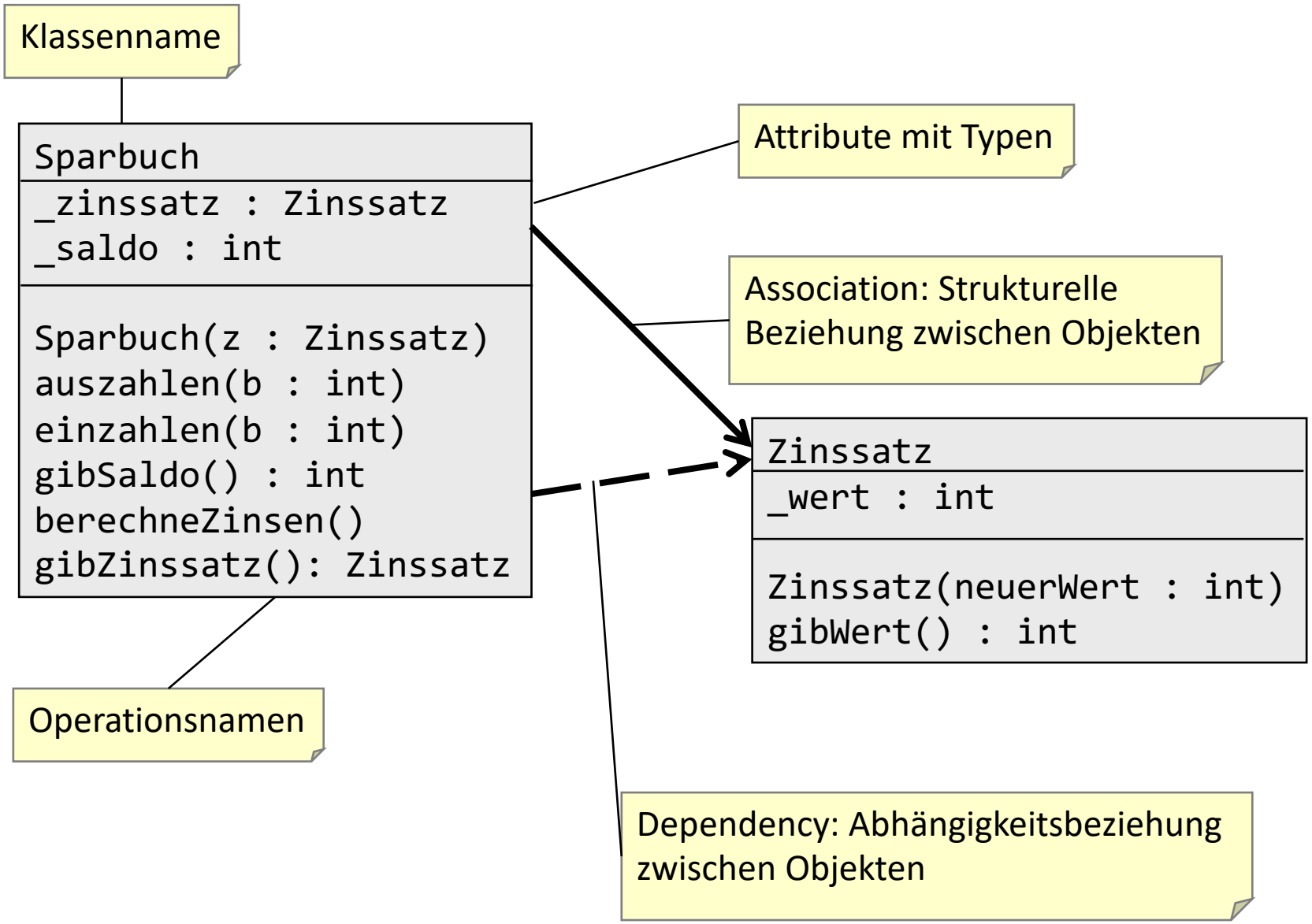
Klassendiagramme (2)




```
Girokonto
_saldo : Betrag
_limit : Betrag
einzahlen(b : Betrag)
auszahlen(b : Betrag)
gibSaldo() : Betrag
berechneZinsen()
```

- **Klassendiagramme** enthalten allgemein Klassen und ihre Beziehungen.
- Die wesentlichen Beziehungen sind
  - **benutzt** (d.h. ruft Dienstleistung auf), dependency
  - **enthält / verwaltet** (d.h. hält Verweis in Feld), association
  - **ist_ein** (d.h. erbt von), generalization

Noch einmal: Ein UML-Klassendiagramm



Zusammenfassung

1. **UML** ist eine grafische Sprache für die Beschreibung von Software-Systemen.

2. UML bildet einen Quasi-Standard für objektorientierte Systeme und ist sehr umfangreich.

3. Die wichtigsten Diagrammtypen der UML die **Klassendiagramme** und die **Objektdiagramme**.

4. Für den Einstieg in die UML ist das Buch „**UML Distilled**“ von Martin Fowler zu empfehlen (im Deutschen „UML konzentriert“).

Überblick

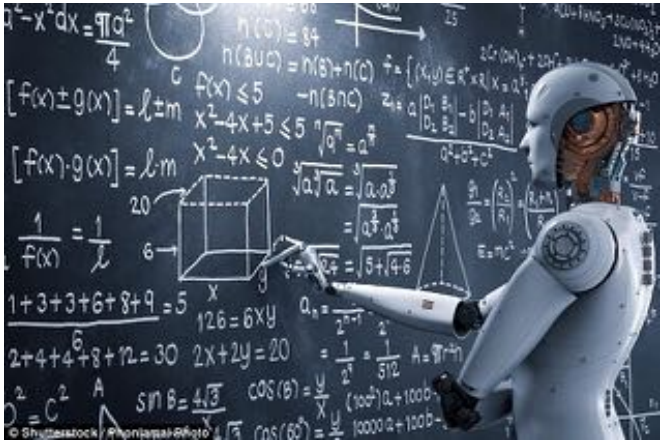
1 UML

2 **Syntaktische Strukturen**

Syntax von Klassendefinitionen

- Die Struktur von Klassendefinitionen richtet sich nach der **Syntax**
- Die Syntax einer Programmiersprachen wird **formal beschrieben** damit sie:

Lesbar für **Menschen** um syntaktisch korrekte Programme zu schreiben



Lesbar für **Computer** um korrekte Programme zu verarbeiten

Syntax, Semantik, Pragmatik

Syntax

- Regeln um Zeichen aneinander zu reihen formen eine Sprache
- Die Syntaxregeln einer Programmiersprache definieren den formalen Aufbau der Sätze und Wörter

Semantik

- Lehre von der inhaltlichen Bedeutung einer Sprache
- Die Semantik eines Programms ist das, was das Programm beim Ablauf im Rechner (und darüber hinaus) bewirkt
- **Semantikregeln** sorgen beispielsweise dafür, dass nur deklarierte Variablen verwendet werden dürfen

Pragmatik

- Lehre vom Gebrauch einer Sprache in einem bestimmten Zusammenhang
- Die Pragmatik eines Programms wird durch den Zweck, die Aufgabenstellung und die jeweilige Verwendung bestimmt

Beispiel in der deutschen Sprache

Die **Syntax** einer Sprache wird üblicherweise in Grammatikregeln beschrieben

<*Deutscher Satz*> → <Subjekt> <Prädikat> <Objekt> .

„Der Mann beißt den Hund.“

Menschen verstehen die **Semantik**, wenn der Satz Sinn ergibt

„Das Haus streichelt den Mann.“

(korrekte Syntax, sinnlose Semantik)

Benutzerhandbücher oder Quelltextkonventionen beinhalten oft Hinweise zur **Pragmatik**

Beispiel: Semantik vs Pragmatik

„Da ist die Tür.“

Semantik

- Beschreibung, wo sich die Tür befindet

Pragmatik (situationsabhängig)

- Antwort auf die Frage einer orientierungslosen Person
- Aufforderung den Raum zu verlassen

Syntax, Semantik, Pragmatik in Java

```
class Girokonto
{
   private int _saldo;

   public void einzahlen(int betrag)
   {
      _saldo = _saldo + betrag;
   }
}
```

```
class <Klassenname>
{
   <optional: Felder>
   <optional: Konstruktoren>
   <optional: Methoden>
}
```

- Vereinfachte **Syntax** für Klassen

- Die **Semantikregeln** von Java definieren beispielsweise Standardkonstruktor

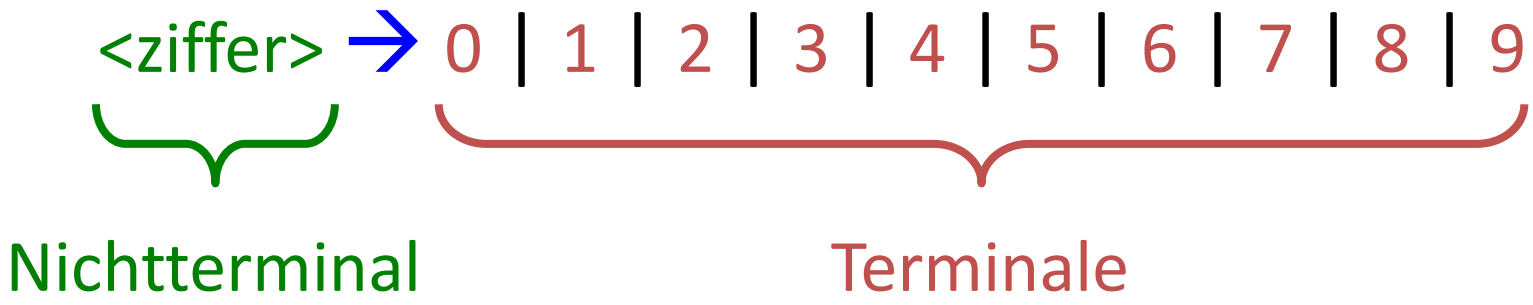
- Die Reihenfolge ist ein Beispiel für die **Pragmatik** von Java
- Die Syntax lässt es anders zu und die Semantik ist nicht beeinflusst

Syntaxbeschreibungen von Programmiersprachen

- Für die Programmierung müssen wir zunächst die **Syntax** der Programmiersprache verstehen
- Den theoretischen Hintergrund dazu (die **kontextfreien Grammatiken** aus der Chomsky-Hierarchie) wird in den FGI-Modulen behandelt
- Wir behandeln für die pragmatische Darstellung:
  - **Backus-Naur-Form**
  - **Syntaxdiagramme**

Backus-Naur-Form (BNF)

- Definiert syntaktische Strukturen mit Hilfe zweier Elementarten
  - **Nichtterminale** oder auch Variablen genannt
  - **Terminale** oder auch Basiselemente genannt



Nichtterminale

- Werden in spitzen Klammern notiert
- Definition eines Nichtterminals heißt **Regel** oder **Produktion**

<Zuweisung> → <Bezeichner> '=' <Ausdruck>

- Durch eine **Regel** können wir das Nichtterminal auf der linken Seite durch die Verkettung der Elemente auf der rechten Seite ersetzen
- Links vom **Ableitungssymbols** (→) ist genau ein Nichtterminal
- Rechts vom → können beliebig viele Elemente stehen
- Jedes Nichtterminal muss in mindestens einer Regel auf der linken Seite stehen

Terminale

- In einer Programmiersprachgrammatik üblicherweise:
  - reservierten Wörter, (z.B. `if`, `else`, `return` etc.)
  - Bezeichner (z.B. Namen von Variablen),
  - Literale (z.B. Zahlen und Zeichenketten)
  - Sonderzeichen (z.B. Operatoren, Satzzeichen, Klammern)
- Werden oft in einfachen Anführungsstrichen notiert

<Zuweisung> → <Bezeichner> '=' <Ausdruck>

- Terminale sind aus Sicht der Grammatik **unteilbare Elemente**
- Für sie existieren keine Regeln innerhalb der Grammatik
- Können auch nach komplizierten Regeln aufgebaut sein
  z.B. die Gleitkommazahlen

Rekursive Definition

- Nichtterminale können rekursiv definiert sein:

```
<Anweisungsfolge> → <Anweisung>
                  | <Anweisung> ';' <Anweisungsfolge>
```

Ableiten von Wörtern einer Sprache

Gegeben ist die folgende **Syntaxbeschreibung**:

```
<Zuweisung>  → <Bezeichner> ':=' <Ausdruck> .
<Bezeichner> → 'A' | 'B' | 'C'.
<Ausdruck>   → <Bezeichner> '+' <Ausdruck>
             | <Bezeichner> '*' <Ausdruck>
             | '(' <Ausdruck> ')'
             | <Bezeichner> .
```

Durch **schrittweises Ersetzen der Nichtterminale** generieren wir ein Wort der Sprache

Beispiel für die **Ableitung** eines Wortes:

```
<Zuweisung> ⇒ <Bezeichner> ':=' <Ausdruck>
            ⇒ 'A :=' <Ausdruck>
            ⇒ 'A :=' <Bezeichner> '*' <Ausdruck>
            ⇒ 'A := B *' <Ausdruck>
            ⇒ 'A := B * (' <Ausdruck> ')'
            ⇒ 'A := B * (' <Bezeichner> '+' <Ausdruck> ')'
            ⇒ 'A := B * ( A +' <Ausdruck> ')'
            ⇒ 'A := B * ( A +' <Bezeichner> ')'
            ⇒ 'A := B * ( A + C )'
```

Erweiterte BNF (BNF)

- Zur praktischen Darstellung erweitern wir die BNF um **optionale** und **wiederholte Elemente**
- **Wiederholbare Elemente** (auch kein Mal) schreibt man in **geschweiften** Klammern.
- Dadurch wird aus:

```
Anweisungsfolge → Anweisung
                | Anweisung ';' Anweisungsfolge
```

mit Hilfe der geschweiften Klammern:

```
Anweisungsfolge → Anweisung { ';' Anweisung }
```

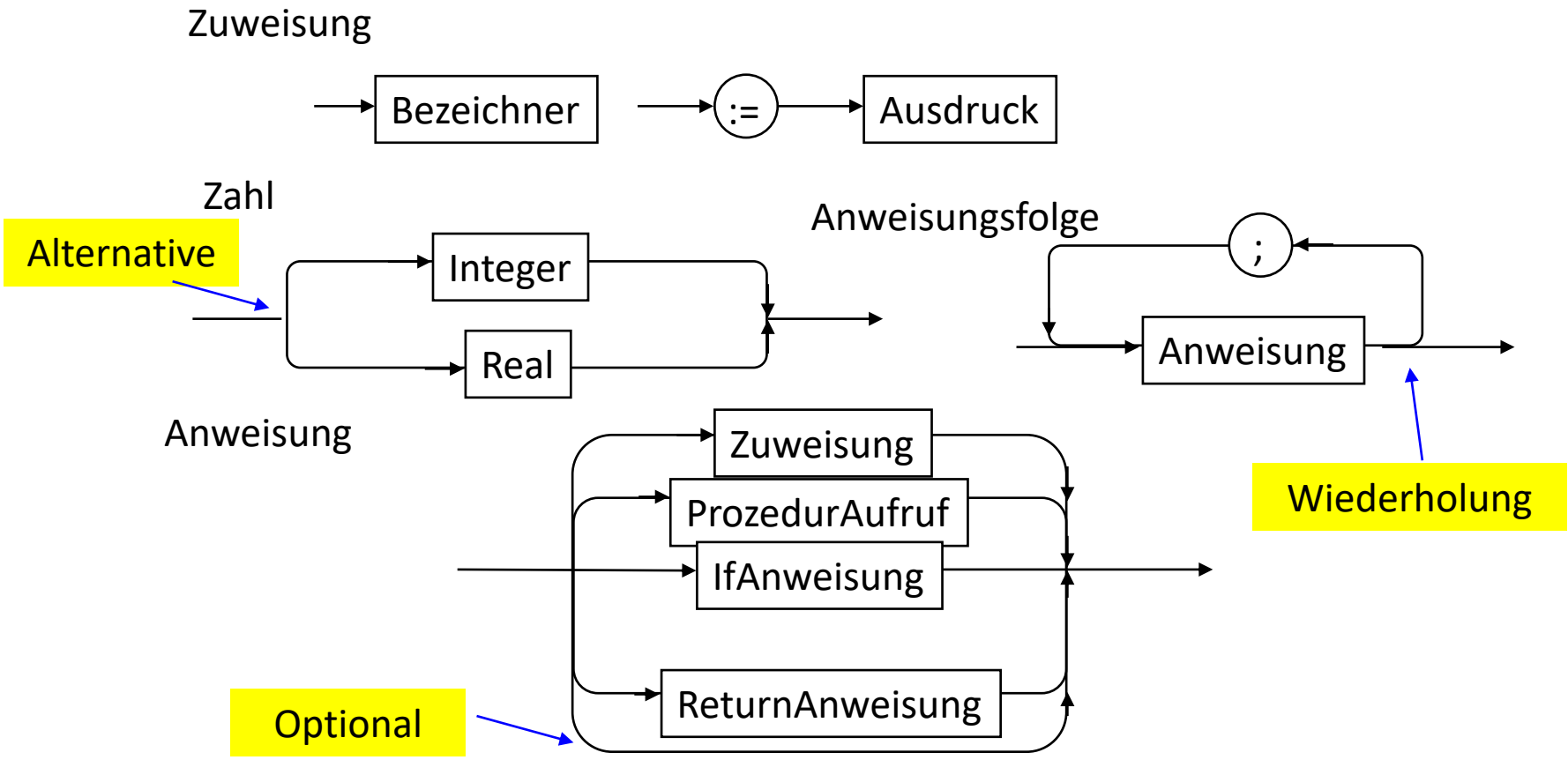
- **Optionale Elemente** schreibt man in **eckigen** Klammern:

```
Anweisung → [ Zuweisung | ProzedurAufruf
            | IfAnweisung | CaseAnweisung
            | WhileAnweisung | RepeatAnweisung
            | LoopAnweisung | WithAnweisung
            | ExitAnweisung | ReturnAnweisung ]
```

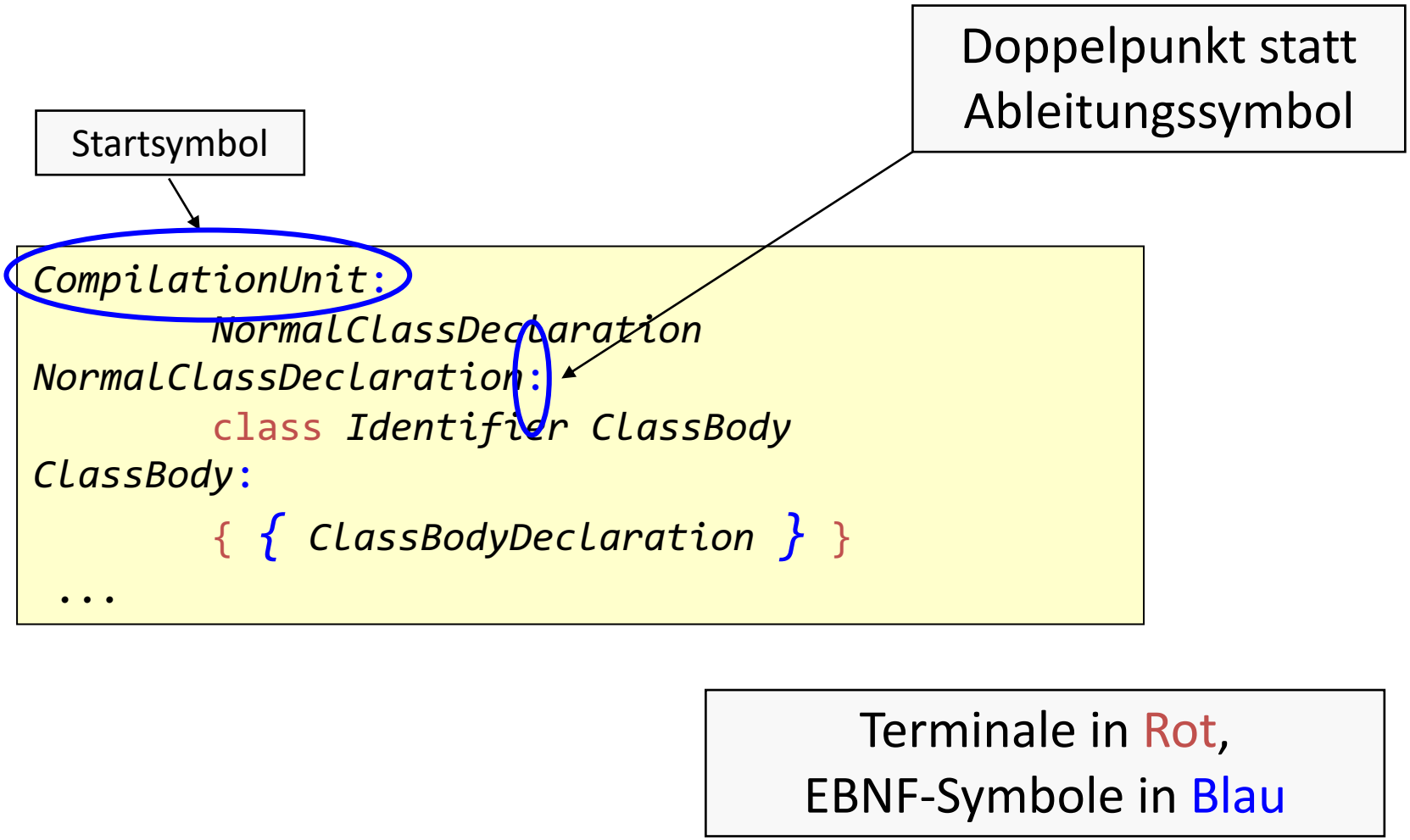
Spitze Klammern ausgelassen

Syntaxdiagramme

- Zur Darstellung von EBNF-Grammatiken können wir **Syntaxdiagramme** verwenden
- **Terminale** werden als **Kreise** dargestellt
- **Nichtterminale** in **Rechtecken** dargestellt.



Vereinfachte Java Syntax Regeln



Vereinfachte Java Syntax Regeln

Nichtterminal für eine **Anweisung** (engl.: statement) als linke Seite einer Regel

Rechte Seite: **6 Alternativen**, jede in einer eigenen Zeile

```
...
Statement:
    if ParExpression Statement [ else Statement ]
    return [ Expression ] ;
    ;
    Assignment ;
    MethodInvocation ;
    Block

Assignment:
    Identifier = Expression

ParExpression:
    ( Expression )
```

Offizielle Beschreibung der Java Syntax

- https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html

Zusammenfassung

1. Der formale Aufbau einer Programmiersprache wird durch **Syntaxregeln** definiert

2. Die **Semantik** eines Programms ist das, was das Programm beim Ablauf im Rechner (und darüber hinaus) bewirkt

2. Die **Pragmatik** eines Programms wird durch den Zweck, die Aufgabenstellung und die jeweilige Verwendung bestimmt

4. Die syntaktische Struktur von Programmiersprachen wird häufig in der **Erweiterten Backus-Naur-Form** (EBNF) beschrieben

5. Die **Syntax** von Java ist in einer **Abwandlung der EBNF** notiert, die wir auch für die **Syntaxdefinition** von Java verwenden.