I.2. Grundlagen von Programmiersprachen

- **1. Der Begriff „Informatik“**
- **2. Syntax und Semantik von Programmiersprachen**

1. Der Begriff „Informatik“

- **"Informatik" =** *Kunstwort* **aus** *Information* **und** *Mathematik*
  - Wissenschaft der Informationsverarbeitung mit großer Nähe zur Mathematik

- **Hauptaufgabe der Informatik**
  - ***Entwicklung formaler, maschinell ausführbarer Verfahren*** zur Lösung von Problemen der Informationsverarbeitung

- **Forderung der Durchführbarkeit mittels einer Maschine:**
  - Informationen müssen als ***maschinell verarbeitbare Daten*** dargestellt werden
  - Lösungsverfahren müssen bis ***ins Detail*** formal beschrieben werden.

Deterministischer Algorithmus

Berechnung von |x-y|

```
1.   Lies Eingaben x und y.

2.   Falls x ≤ y: Weiter mit Schritt 3.
     Falls x > y: Weiter mit Schritt 4.

3.   Berechne a = y - x.
     Weiter mit Schritt 5.

4.   Berechne a = x - y.

5.   Gib a aus.
```

Indeterministischer Algorithmus

Berechnung von |x-y|

```
1.   Lies Eingaben x und y.
     Weiter mit Schritt 2 oder Schritt 3.

2.   Berechne a = x - y.
     Weiter mit Schritt 4.

3.   Berechne a = y - x.

4.   Falls a ≥ 0: Gib a aus.
     Falls a < 0: Gib -a aus.
```

Fragen

- **Wie kann man aus einer Lösungsidee einen Algorithmus konstruieren?**
  - "schrittweise Programmentwicklung"

- **Wie zeigt man, dass ein Algorithmus tatsächlich das tut, was er tun soll?**
  - Verifikation: partielle Korrektheit
  - Terminierung

- **Wie "gut" ist ein Algorithmus?**
  - Speicherverbrauch, benötigte Zeit (***Effizienz***)
  - Aufwandsabschätzungen

Pro-
gra

Datenstruk-
turen und
Algorith-
men

I.2. Grundlagen von Programmiersprachen

- 1. Der Begriff „Informatik“
- **2. Syntax und Semantik von Programmiersprachen**

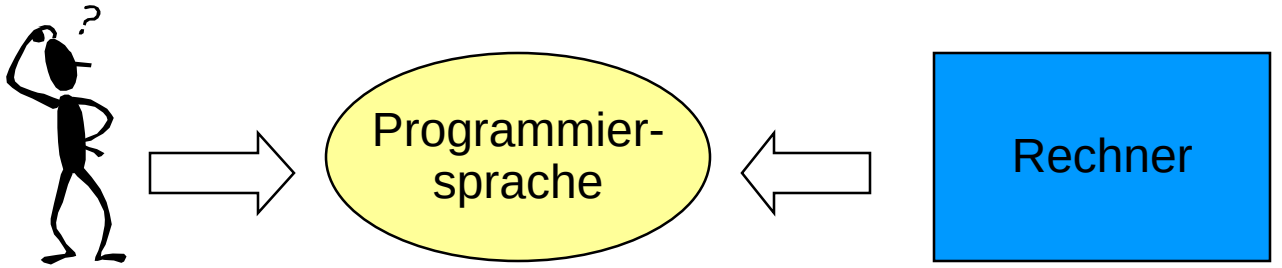
2. Syntax & Semantik von Programmiersprachen

Ein erstes Java-Programm:

```
public class Rechnung {

  public static void main (String [] arguments) {

     int x, y;
     x = 10;
     y = 23 * 33 + 3 * 7 * (5 + 6);
     System.out.print ("Das Resultat ist ");
     System.out.println (x + y);

  }

}
```

Programmiersprachen

- **Die Programmiersprache bildet die *Schnittstelle* zwischen Mensch und Rechner**



Beide haben unterschiedliche Anforderungen

- **Mensch**
  - Erlernbarkeit
  - Lesbarkeit
  - Ausdrucksstärke

Höhere Programmier-
sprachen

- **Rechner**
  - einfaches Übersetzen in Maschinensprache
  - Generierung von effizientem Code

Maschinen-
sprachen

Kenntnis verschiedener Sprachen

- **Eigene Ideen bei der Software-Entwicklung können besser ausgedrückt werden**
- **Nötig, um in konkreten Projekten geeignete Sprache auszuwählen**
- **Erleichtert das Erlernen weiterer Programmiersprachen**
- **Nötig für den Entwurf neuer Programmiersprachen**

Übersicht

Imperative Sprachen

- Folge von nacheinander ausgeführten Anweisungen

Deklarative Sprachen

- Spezifikation dessen, *was* berechnet werden soll
- Festlegung, wie Berechnung verläuft durch Compiler

Prozedurale Sprachen

- Variablen, Zuweisungen, Kontrollstrukturen

Funktionale Sprachen

- keine Seiteneffekte
- Rekursion

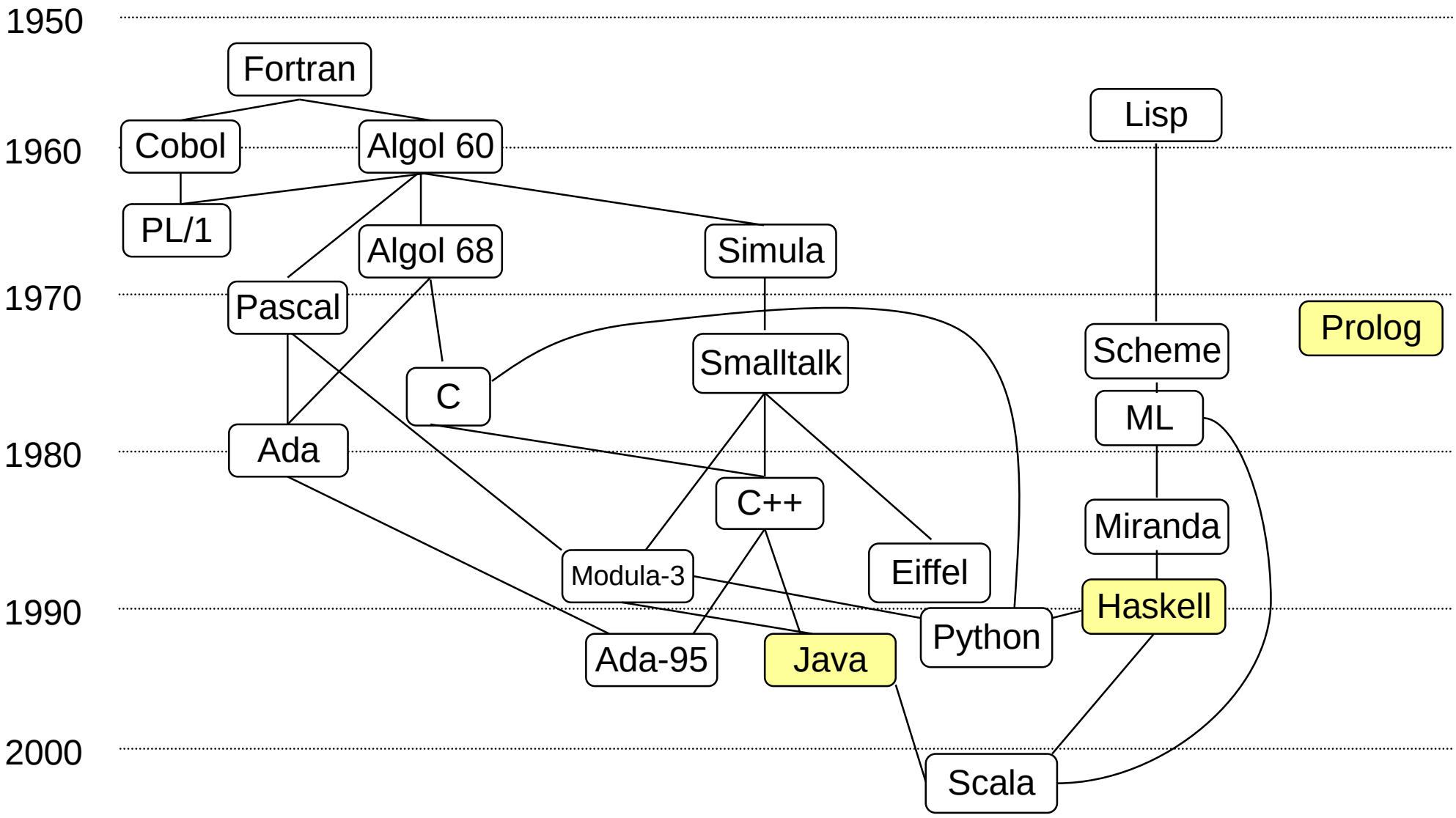
Objektorientierte Sprachen

- Objekte und Klassen
- ADT und Vererbung

Logische Sprachen

- Regeln zur Definition von Relationen

Wichtige Programmiersprachen



Programmiersprachen - Definition

- **Programmiersprachen sind Sprachen, deren Syntax und Semantik genau festgelegt ist.**

- **Syntax:**
  - Definition aller ***zulässigen Wörter / Programme***, die in einer Sprache formuliert werden können

- **Semantik:**
  - ***Bedeutung*** der zulässigen Wörter / Programme
  - Syntaktisch ***falsche*** Wörter / Programme haben ***keine*** Semantik

Alphabet, formale Sprache

Alphabet

- ***nichtleere endliche*** Menge von Zeichen („Buchstaben“, Symbole)

Wort über einem Alphabet

- ***endliche Folge*** von Buchstaben, die auch ***leer*** sein kann (ε leere Wort)
- A* bezeichnet die ***Menge aller Wörter*** über dem Alphabet A (inkl. dem leeren Wort)

Formale Sprache

- Sei A ein Alphabet. Eine (formale) Sprache (über A) ist ***eine beliebige Teilmenge von A****.

Endliche Beschreibungsvorschrift für unendliche Sprachen

- ***Grammatik***, die Sprache erzeugt, oder ***Automat***, der Sprache erkennt

Grammatik - informell

- **Definiert *Regeln*, die festlegen, welche Wörter über einem *Alphabet* zur Sprache gehören und welche nicht.**

- **Beispiel: Grammatik für "Hund-Katze-Sätze"**

1	Satz	→	Subjekt Prädikat Objekt
2	Subjekt	→	Artikel Attribut Substantiv
3	Artikel	→	ε
4	Artikel	→	der
5	Artikel	→	die
6	Artikel	→	das
7	Attribut	→	ε
8	Attribut	→	Adjektiv
9	Attribut	→	Adjektiv Attribut
10	Adjektiv	→	kleine
11	Adjektiv	→	bissige
12	Adjektiv	→	große
13	Substantiv	→	Hund
14	Substantiv	→	Katze
15	Prädikat	→	jagt
16	Objekt	→	Artikel Attribut Substantiv

Kontextfreie Grammatik / Sprache

- **Produktionsregeln:** $A \rightarrow y$ $A \in N, y \in V^*$

  d.h.: Links steht genau ein Nichtterminalsymbol

- **Wichtigste Klasse zur formalen Beschreibung der Syntax von Programmiersprachen.**

- **Es ist möglich, Automaten zu bauen, die Wörter einer kontextfreien Sprache erkennen (*Wortproblem*) und ihre syntaktische Struktur analysieren (Compilerbau)**

- **Notationen zur Darstellung kontextfreier Grammatiken**
  - ***Syntaxdiagramme***
  - ***Extended Backus-Naur-Form* (EBNF)**

Grammatik - Beispiel

- **Sei G = (N, T, P, S) mit**
  - N = {A, B}
  - T = {a, b, c, d}
  - P = { $A \rightarrow aBbc$,
    $B \rightarrow aBb$,
    $aBb \rightarrow d$ }
  - S = A

$$L(G) = \{ a^n d\, b^n c \mid n \geq 0 \}$$

  - G ist keine kontextfreie Grammatik, da die dritte Produktionsregel auf der linken Seite mehr als nur das Nichtterminalsymbol B enthält.

- **Ersetzt man in G die Produktionen P durch P', dann ist G' kontextfrei. Es gilt L(G) = L(G').**
  - P' = { $A \rightarrow Bc$,
    $B \rightarrow aBb$,
    $B \rightarrow d$ }

EBNF

EBNF (Extended Backus-Naur-Form)

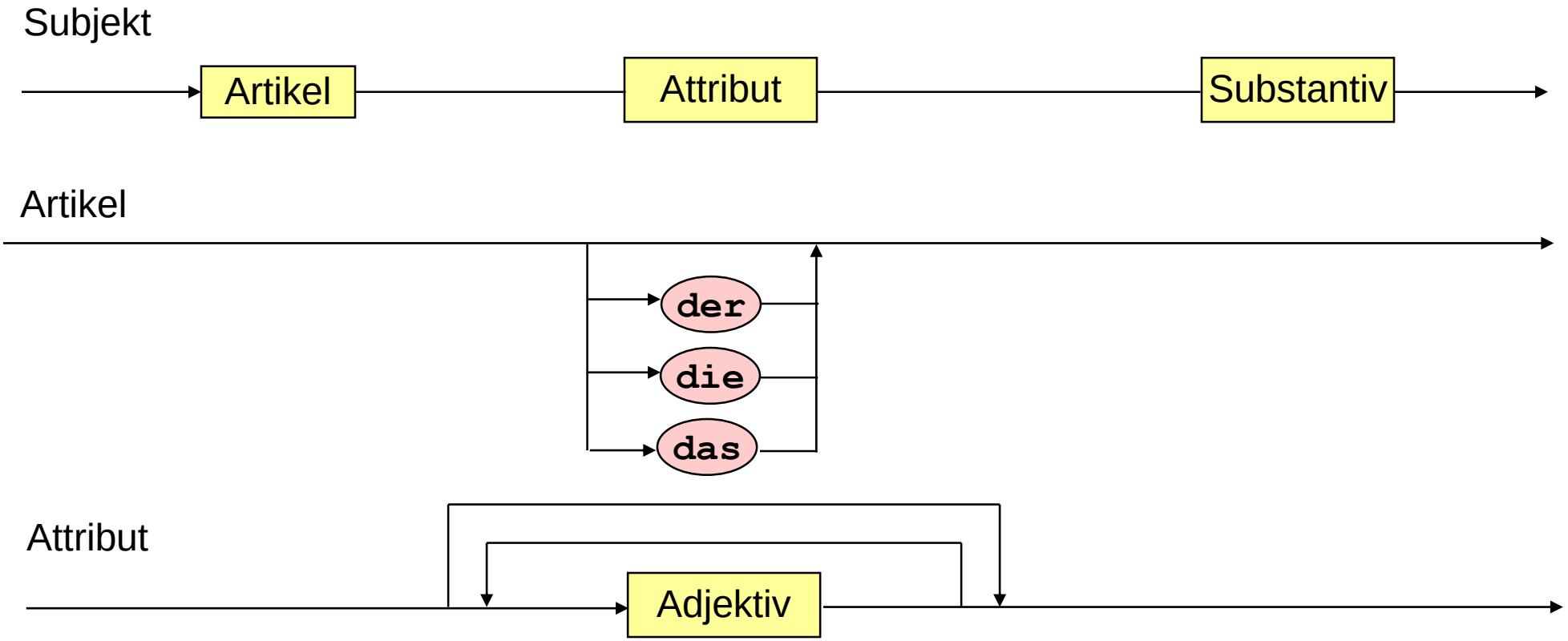
- kompaktere Repräsentation kontextfreier Grammatiken
- BNF erstmals benutzt zur Definition der Sprache ***Algol-60***
- **EBNF-Notation**
  - = „definiert als“
  - (...|...) genau eine Alternative aus der Klammer muss kommen
  - [...] Inhalt der Klammer kann kommen oder nicht
  - { ... } Inhalt der Klammer kann n-fach kommen, $n \geq 0$
  - Terminalsymbole werden in " " eingeschlossen

Beispiel - Grammatik

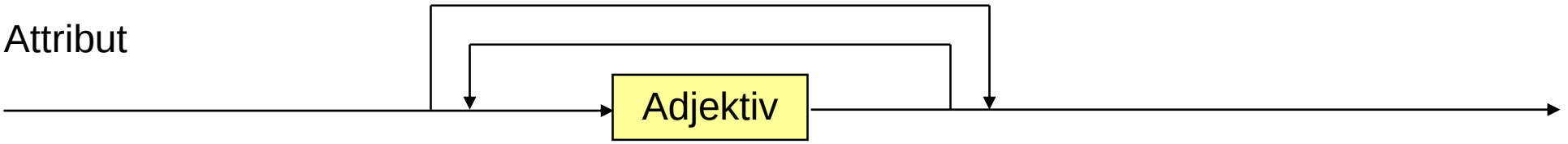
1	Satz	→	Subjekt Prädikat Objekt
2	Subjekt	→	Artikel Attribut Substantiv
3	Artikel	→	ε
4	Artikel	→	der
5	Artikel	→	die
6	Artikel	→	das
7	Attribut	→	ε
8	Attribut	→	Adjektiv
9	Attribut	→	Adjektiv Attribut
10	Adjektiv	→	kleine
11	Adjektiv	→	bissige
12	Adjektiv	→	große
13	Substantiv	→	Hund
14	Substantiv	→	Katze
15	Prädikat	→	jagt
16	Objekt	→	Artikel Attribut Substantiv

Syntaxdiagramme - 1

- **Syntaxdiagramme (beschreiben Produktionen *grafisch*)**
  - Nichtterminalsymbole sind Rechtecke
  - Terminalsymbole sind rund / oval



Syntaxdiagramme - 2



Alternative: Rekursives Syntaxdiagramm