
Proseminar

Ausarbeitung & Vortrag

Prof. Dr. Jürgen Giesl

Lehr- und Forschungsgebiet Informatik 2

Lernziele des Proseminars

■ **Wie komme ich zu gesicherten Ergebnissen?**

- Stimmen Angaben? Andere Quellen? Neuere Entwicklungen?

■ **Was ist der wesentliche Inhalt?**

- Inhalt verstehen
- danach: Abstraktion von den (technischen) Details
- Strukturierung der wesentlichen Punkte

■ **Wie vermittele ich meine Ergebnisse?**

- wesentlichen Inhalt (ohne Details) in eigenen Worten erklären
- muss für andere Studierenden ohne Vorkenntnisse verständlich sein
- alles erklären, was erwähnt wird
- innerhalb vorgegebener Dauer bzw. Seitenzahl

Organisation

- **15.04.2011 Kurzvortrag halten**
- **5 Wochen vor dem Vortrag**
 - Besprechung der Gliederung mit Betreuer
- **3 Wochen vor dem Vortrag**
 - Abgabe der Ausarbeitung beim Betreuer
- **nach Wunsch**
 - Besprechung der Vortragsfolien mit Betreuer
- **1 Woche vor dem Vortrag**
 - Ausarbeitung kopieren und zum Proseminar mitbringen

Ausarbeitung

Ausarbeitung

- **12 Seiten (inklusive Titelblatt)**
- auf **Verständlichkeit** achten
- **mehrmals durchlesen**
- **Rechtschreibung und Grammatik**
 - Abbruch der Bearbeitung bei mehr als 5 Rechtschreibfehlern pro Seite
- **LaTeX vorteilhaft bei Formeln**
- **Gliederung:**
 - Einleitung
 - Hauptteil (mehrere Kapitel). Ggf. Unterkapitel (dann mindestens 2).
 - Zusammenfassung
 - Literaturverzeichnis
- **Kein Inhalts- oder Abbildungsverzeichnis**

Einleitung und Zusammenfassung

Einleitung

- **kurze Beschreibung des Themas**
- **Motivation und Zielsetzung**
 - Warum interessant?
 - Was soll gezeigt werden?
- **Übersicht über den Aufbau der Arbeit**
 - Welchen Inhalt haben die folgenden Kapitel?

Zusammenfassung

- **Was wurde gezeigt?**
- **Ausblick**

Stil

■ keine 1. Person Singular

- nicht: „... wie ich gezeigt habe“
- sondern: „... wie gezeigt wurde“ oder „... wie wir gezeigt haben“

■ Schlüsselbegriffe *kursiv*

■ nicht zu viele Fußnoten

■ Beispiele

- jedes Konzept muss mit einem Beispiel erklärt werden
- möglichst immer dasselbe Beispiel
- möglichst kleines einfaches Beispiel

Literaturverzeichnis

■ Alphabetisch sortieren (Nachname des 1. Autors)

■ Vollständig zitieren

- *Zeitschriftenartikel*: Autoren, Titel, Name der Zeitschrift, Band- und Heftnummer, Seitenzahl, Jahr

[AW97] M. Arlitt, C. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631-645, 1997.

- *Konferenzbeitrag*: Autoren, Titel, Name der Konferenz, Seiten, Jahr

[TM02] Tan, H.K.; Moreau, L.: Certificates for Mobile Code Security. *Proceedings of the ACM Symposium on Applied Computing (SAC 2002)*, pages 76-81, 2002.

- *Fachbuch*: Autoren, Titel, Verlag, Jahr

[Ha98] Haverkort, B.R.: *Performance of Computer Communication Systems – A Model-Based Approach*. John Wiley & Sons, New York, 1998.

Vortrag

Vortrag

- **30 Minuten**
- **beide Studierenden sollten etwa gleich lang sprechen**
- **im Prinzip gleicher Inhalt wie Ausarbeitung**
- **Rechtschreibung und Grammatik**
- **LaTeX vorteilhaft bei Formeln (beamer style),
Powerpoint vorteilhaft bei Multimedia**
- **Gliederung**
 - Einleitung
 - Hauptteil (mehrere Kapitel)
 - Zusammenfassung
- **ggf. praktische Demo**

Stil

■ Hauptziel: **Verständlichkeit!!**

- wesentlichen Inhalt (ohne Details) in eigenen Worten erklären
- muss für andere Studierende ohne Vorkenntnisse verständlich sein
- alles erklären, was erwähnt wird oder auf Folien steht

■ nur **Stichworte auf Folien, keine ganzen Sätze**

■ keine **technischen Details**

■ Folien nicht zu voll

■ große **Schrift, ggf. Farben sinnvoll einsetzen**

■ **Beispiele**

- jedes Konzept muss mit einem Beispiel erklärt werden
- möglichst immer dasselbe Beispiel
- möglichst kleines einfaches Beispiel

Stil

■ **Vortrag mehrfach üben!!**

- vorher überlegen, was man zu jeder Folie sagen will
- üben, wie der Vortrag in 30 Minuten gehalten werden kann
- Notfallplan, wenn man zu langsam oder zu schnell ist

■ **frei reden**

■ **ins Publikum schauen**

■ **alle benötigten Informationen müssen auf der Folie sein**

- Ggf. 2. Beamer benutzen, um parallel Konzept und Bsp zu zeigen

■ **keine Nachfragen**

- „Hat das jeder verstanden?“

■ **keine Schlussfolie**

- „Noch Fragen?“, „Vielen Dank für die Aufmerksamkeit“

Kurzvortrag

15. 4. 2011

Kurzvortrag

- **6 Minuten**
- **beide Studierenden sollten etwa gleich lang sprechen**
- **Inhalt: Überblick über das Thema**
- **ähnliche Hinweise wie beim Vortrag**
- **Verständlichkeit**
 - muss für andere Studierende ohne Vorkenntnisse verständlich sein
- **Test, um erstes Feedback zum Vortragsstil zu bekommen**
- **Motivation, sich schon jetzt um das Thema zu kümmern**
- **Donnerstag, 14.4.: Folien an Betreuer schicken**
 - pdf, Powerpoint, Open Office

Beispielvortrag

Programmierkonzepte in Java, Haskell und Prolog

(30 Minuten)

Programmierkonzepte in Java, Haskell und Prolog

Prof. Dr. Jürgen Giesl

Lehr- und Forschungsgebiet Informatik 2

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

■ Prozedurale Sprachen

- Variablen, Zuweisungen, Kontrollstrukturen

■ Objektorientierte Sprachen

- Objekte und Klassen
- ADT und Vererbung

Deklarative Sprachen

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

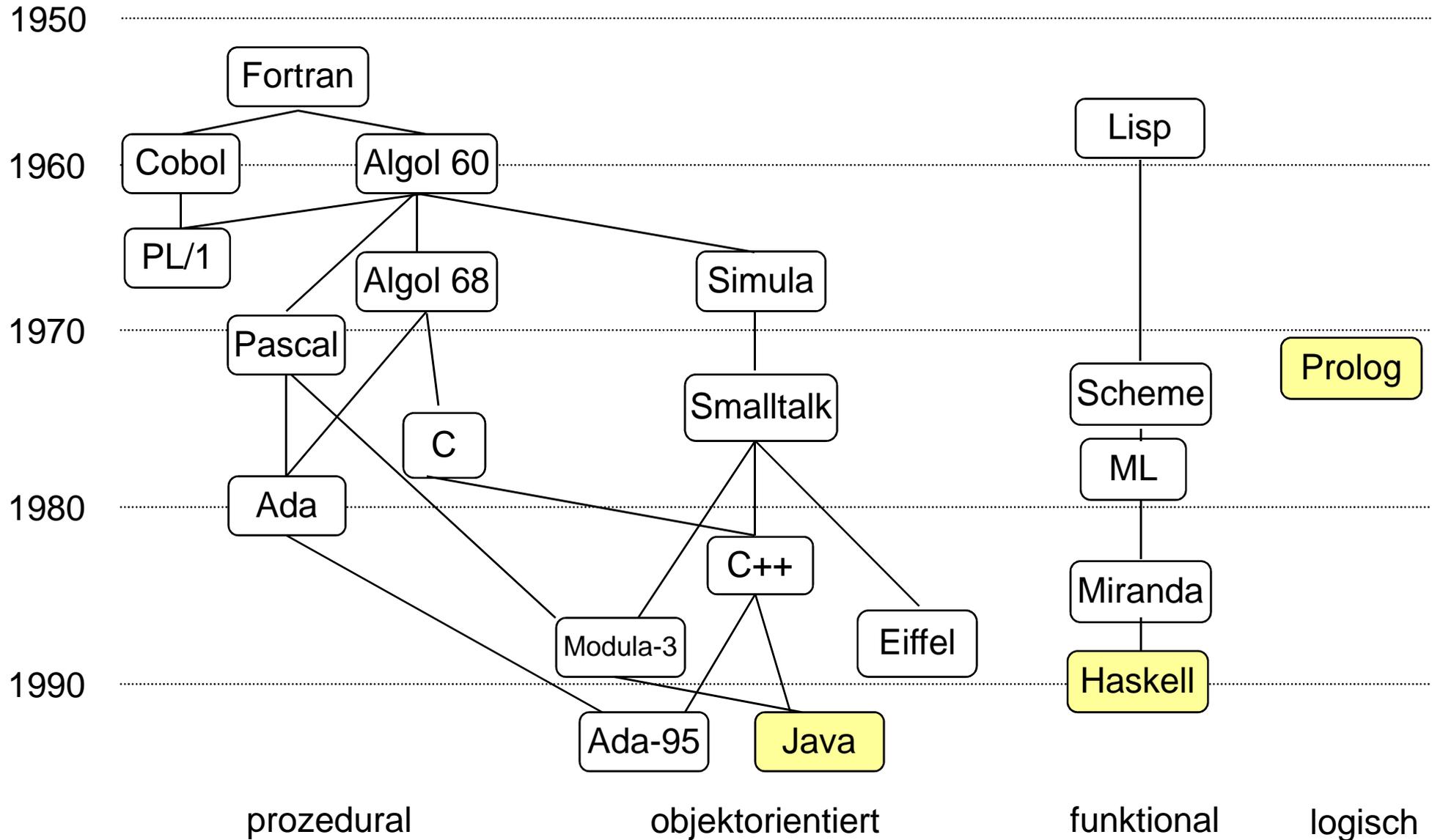
■ Funktionale Sprachen

- keine Seiteneffekte
- Rekursion

■ Logische Sprachen

- Regeln zur Definition von Relationen

Wichtige Programmiersprachen



1. Imperative prozedurale Programmierung

Eingabe: Eine Liste z

Ausgabe: Länge der Liste

Algorithmus:

1. Setze $x = z$.
2. Setze $n = 0$.
3. Falls x nicht leer:
 - 3a. Lösche das erste Element von x .
 - 3b. Erhöhe n um 1.
 - 3c. Zurück zu Schritt 3.
4. Gib Wert von n zurück.

Imperatives Programm (*Java*)

Eingabe: Eine Liste z

Ausgabe: Länge der Liste

Algorithmus:

1. Setze $x = z$.
2. Setze $n = 0$.
3. Falls x nicht leer:
 - 3a. Lösche das erste Element von x .
 - 3b. Erhöhe n um 1.
 - 3c. Zurück zu Schritt 3.
4. Gib Wert von n zurück.

```
static int len (List x) {  
    int n = 0;  
  
    while (x != null) {  
        x = x.rest;  
        n = n + 1;  
    }  
  
    return n;  
}
```

Kennzeichen imperativer Programmierung

- Programm besteht aus einzelnen Anweisungen, die nacheinander abgearbeitet werden.
- Verschiedene Kontrollstrukturen, um Programmablauf zu steuern.
- Abarbeitung einer Anweisung ändert Werte der Variablen.
- Seiteneffekte

2. Objektorientierte Programmierung (*Java*)

```
class List {
    int kopf;
    List rest;
    ...
    int len () {
        List x = this;
        int n = 0;

        while (x != null) {
            x = x.rest;
            n = n + 1;
        }
        return n;
    }
}
```

```
static int len (List x) {
    int n = 0;

    while (x != null) {
        x = x.rest;
        n = n + 1;
    }

    return n;
}
```

Vererbung

```
class PersonenListe extends List {  
    String name;  
    ...  
}
```

- `PersonenListe` ist Unterklasse von `List`
- Objekte von `PersonenListe` *erben* Attribute und Methoden von `List`
- `y = [(Müller,15), (Meier,70), (Schmidt,36)]`
Dann: `y.len() = 3`

Kennzeichen objektorientierter Programme

- **Teilmenge der imperativen Sprachen**
- **Klassen und Objekte**
- **Unterschiedliche Zugriffsrechte für verschiedene Attribute und Methoden (*Geheimnisprinzip, Datenkapselung, abstrakte Datentypen*)**
- **Modularität, Änderungsfreundlichkeit**
- **Vererbung**

3. Funktionale Programmierung

- A. Falls die Liste z leer ist, so ist $\text{len}(z) = 0$.
- B. Falls die Liste z nicht leer ist und "rest" die Liste z ohne ihr erstes Element ist, so ist $\text{len}(z) = 1 + \text{len}(\text{rest})$.

Programm in *Haskell*:

```
len [] = 0
len (kopf : rest) = 1 + len rest
```

Ausführung funktionaler Programme

```
len [] = 0
```

```
len (kopf : rest) = 1 + len rest
```

```
len [15, 70, 36]
= 1 + len [70, 36]
= 1 + 1 + len [36]
= 1 + 1 + 1 + len []
= 1 + 1 + 1 + 0
= 3
```

Kennzeichen funktionaler Programme

- **Rekursion statt Schleifen**
- **Keine Seiteneffekte (*referentielle Transparenz*)**
- **Funktionen als gleichberechtigte Datenobjekte (*Funktionen höherer Ordnung*)**
- **Verwendung von Funktionen für verschiedene Typen (*Polymorphismus*)**
- **Programme sind kürzer, klarer, besser zu warten, zuverlässiger, schneller zu erstellen**

4. Logische Programmierung

- A. Die Länge der leeren Liste ist 0.
- B. Die Länge der Liste [Kopf | Rest] ist N, falls M die Länge der Liste "Rest" ist und falls N den Wert M + 1 hat.

Programm in *Prolog*:

```
len([], 0).  
  
len([Kopf | Rest], N) :- len(Rest, M),  
                        N is M + 1.
```

Ausführung logischer Programme

```
len([], 0).
```

```
len([Kopf | Rest], N) :- len(Rest, M),  
                          N is M + 1.
```

```
?- len([15, 70, 36], 3).
```

```
Yes
```

Ausführung logischer Programme

```
len([], 0).
```

```
len([Kopf | Rest], N) :- len(Rest, M),  
                          N is M + 1.
```

```
?- len([15, 70, 36], 2).
```

No

Ausführung logischer Programme

```
len([], 0).
```

```
len([Kopf | Rest], N) :- len(Rest, M),  
                          N is M + 1.
```

```
?- len([15, 70, 36], Q).
```

```
Q = 3
```

```
?- len(L, 2).
```

```
L = [X, Y]
```

```
?- len(L, Q).
```

```
L = []           Q = 0 ;
```

```
L = [X]         Q = 1 ;
```

```
L = [X, Y]      Q = 2
```

Kennzeichen logischer Programme

- **Programme = Fakten und Regeln**
- **Keine Kontrollstrukturen**
- **Ein- und Ausgabevariablen liegen nicht fest**
- **Besonders gut geeignet für Künstliche Intelligenz (z.B. *Expertensysteme*)**

Zusammenfassung

■ Grundlegende Paradigmen der

- imperativen prozeduralen Programmierung (*Java*)
- objektorientierten Programmierung (*Java*)
- funktionalen Programmierung (*Haskell*)
- logischen Programmierung (*Prolog*)

■ Charakteristische Konzepte der Sprachfamilien

■ Einfaches Programm in typischen Sprachen

Analyse

Analyse des Beispielvortrags

- LaTeX vorteilhaft bei Formeln (beamer style),
Powerpoint vorteilhaft bei Multimedia
- Hauptziel: **Verständlichkeit!!**
 - wesentlichen Inhalt (ohne Details) in eigenen Worten erklären
 - muss für Studierende ohne Vorkenntnisse verständlich sein
- nur **Stichworte auf Folien, keine ganzen Sätze**
- **keine technischen Details**
- **Folien nicht zu voll**

Analyse des Beispielvortrags

- **große Schrift, ggf. Farben sinnvoll einsetzen**

- **Beispiele**
 - jedes Konzept muss mit einem Beispiel erklärt werden
 - möglichst immer dasselbe Beispiel
 - möglichst kleines einfaches Beispiel

- **Alle benötigten Informationen müssen auf der Folie sein**

Proseminar

Datum	Vortragende	Thema	Betreuer
08.04.2011	J. Giesl	Allgemeine Hinweise zu Vorträgen	
15.04.2011	Alle Proseminaristen	Kurzvorträge	alle
29.04.2011	J. L. Nimpa, M. Pabst	Java Collections, Iterations und Generics	C. Otto
06.05.2011	H.-P. Kühn, M. Rosenstengel	Java Refactoring	M. Brockschmidt
13.05.2011	A. Bayer, D. Studer	Multithreading und Synchronisation in Java	C. Fuhs
20.05.2011	O. Kautz, B. Sommer	Haskell Typklassen und unendliche Datenobjekte	M. Brockschmidt
27.05.2011	M. Behrendt, A. Prause	Effiziente Datenstrukturen mit Haskell	F. Emmes
10.06.2011	D. Neuen, P. Keldenich	Prolog mit vordefinierten Prädikaten	T. Ströder