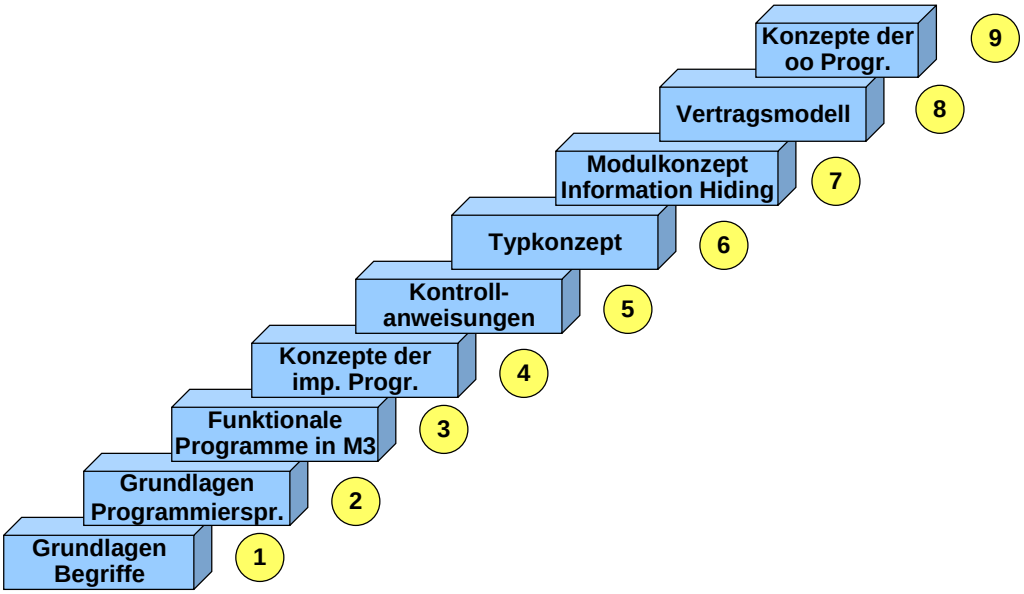
Zusammenfassung
Teile I - IV

Aufbau der Vorlesung - Teile I - IV



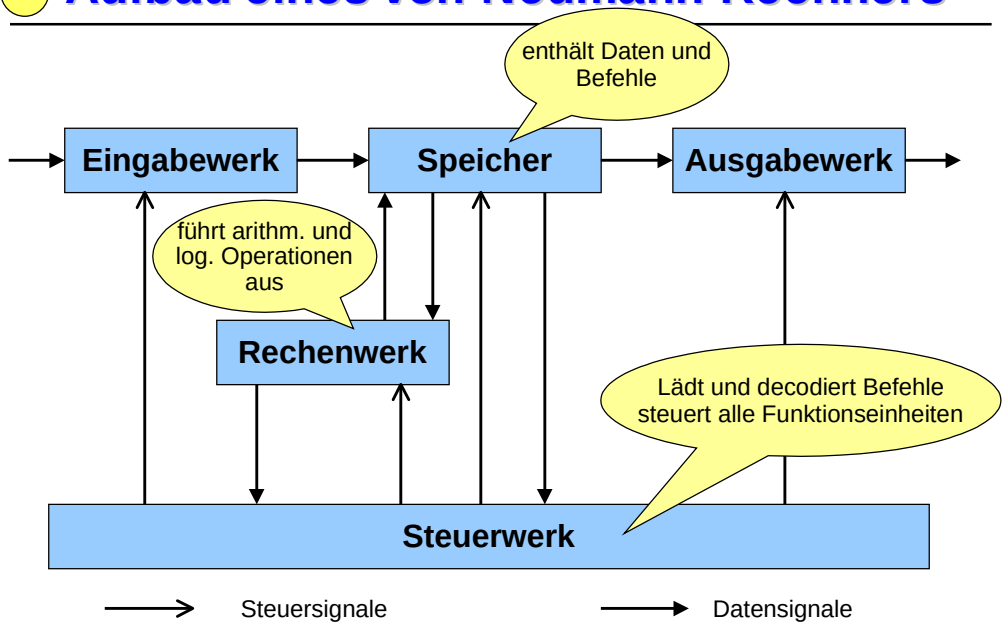
1

Begriffe

- **Ein Algorithmus ist ein Verfahren, welches**
  - in einem ***endlichen*** Text niedergelegt werden muß
  - ***effektiv*** ausführbar ist,
  - durch eine *Maschine* ***ausgeführt*** werden kann
  - Anzahl und Ausführungszeit der Elementaroperationen sind beschränkt
  - Ein Algorithmus wird entsprechend seiner Vorschrift schrittweise ausgeführt
- **Definition: Software**
  - "Computer ***programs***, ***procedures***, and possibly associated ***documentation*** and ***data*** pertaining to the operation of a computer system."
- **Definition: Programm**
  - "A combination of ***computer instructions*** and ***data definitions*** that enable computer hardware to ***perform*** computational or control functions".
- **Unter dem Begriff Programmieren versteht man**
  - das **Lösen von Problemen** unter Zuhilfenahme eines **Rechners**

1

Aufbau eines von-Neumann-Rechners



2

Alphabet, Wort, Formale Sprache

Alphabet

- Ein Alphabet ist eine ***nichtleere endliche*** Menge von ***unterscheidbaren*** Zeichen ("Buchstaben")
  A = {a_1, a_2, a_3, …} mit einer ***Ordnungsrelation*** ≤ ($a_1 \leq a_2 \leq a_3$ ….)

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 ***jede beliebige Teilmenge von A****.

2

Grammatik

Definition:

- Eine Grammatik G für eine Sprache L ist definiert durch
- ein ***Viertupel (N, T, P, S)***
- N: Menge der ***Nichtterminalsymbole***
- T: Menge der ***Terminalsymbole***
- P: Menge von ***Produktionsregeln***
- S: das ***Startsymbol***

Typ-0-Grammatik

- Gestalt der Produktionen ist nicht eingeschränkt

Typ-1 oder kontextsensitive Grammatik

- Produktionen sind beschränkt oder kontextsensitiv

Typ-2 oder *kontextfreie* Grammatik

- die linke Seite einer Produktion ist immer ein Nichtterminal

Typ-3 oder reguläre Grammatik

- Produktionen sind terminierend, links- , rechtslinear

2

EBNF u. Syntaxdiagramme

- **EBNF**
  - **E**xtended **B**ackus-**N**aur-**F**orm
  - ***Meta-Sprache*** zur Beschreibung der Syntax formaler Sprachen
  - ***Metasymbole*** von EBNF sind
    - = „definiert als“
    - (...|...) genau eine Alternative aus der Klammer muß stehen
    - [...] Inhalt der Klammer kann stehen oder nicht
    - { ... } Inhalt der Klammer kann n-fach stehen, n >= 0
    - . Ende der Produktion
    - Terminalsymbole werden in " " eingeschlossen
- **Syntaxdiagramme**
  - beschreiben Produktionen ***grafisch***
  - Nichterminalsymbole sind Rechtecke
  - Terminalsymbole sind Langrunde

3

Funktionale Programmierung

- **Konzept**
  - Formulierung von ***Funktionsdefinitionen***
  - Ausführung eines funktionalen Programms besteht in der ***Berechnung*** eines ***Ausdrucks*** mit Hilfe dieser Funktionen
  - Berechnung liefert ein ***Ergebnis*** zurück
- **Was benötigt man dazu**
  - Daten / ***Datentypen*** und ***elementare*** Funktionen
  - Möglichkeit, Funktionen zu ***definieren***
  - Ausdrucksmittel zur ***Vernetzung*** von Funktionen
- **Eine Funktion**
  - hat einen ***Namen***
  - hat keinen oder mehrere ***Eingabeparameter*** (Argumentbereich)
  - hat einen ***Ergebnistyp*** (Ergebnisbereich)
  - hat eine ***Berechnungsvorschrift*** , ist frei von ***Seiteneffekten***

3

Formale & aktuelle Parameter

- **Parameter**
  - erlauben es, Funktionen mit ***Eingabewerten*** zu versorgen
  - haben eine ***Typ***
  - dadurch werden Funktionen ***flexible einsetzbar***
- **Formale Parameter**
  - werden in der ***Definition*** einer Funktion angegeben
  - dienen als ***Stellvertreter*** im ***Rumpf*** der Funktion für die zur Laufzeit des Programms übergebenen aktuellen Parameter
- **Aktuelle Parameter**
  - beim ***Aufruf*** einer Funktion müssen ihre formalen Parameter gemäß ihrer Definition an aktuelle Parameter ***gebunden*** werden
  - diese werden dann im Rumpf ***verwendet***.

3

Deklaration, Anweisung, Ausdruck

- **Deklarationen:**
  - Idee: Namen (***Bezeichner***) werden vereinbart, damit diese später benutzt werden können
  - Die in einem Block deklarierten Bezeichner sind nur innerhalb des Blockes ***gültig***.
- **Anweisungen:**
  - Sind in Blöcken enthalten.
  - Die Folge der Anweisungen eines Blocks wird bei Ausführung in der ***Reihenfolge der Aufschreibung*** abgearbeitet.
- **Ausdrücke**
  - werden ***ausgewertet***
  - liefern einen Wert (Ergebnis)
  - Viele Anweisungen erlauben, daß Ausdrücke verwendet werden können

3

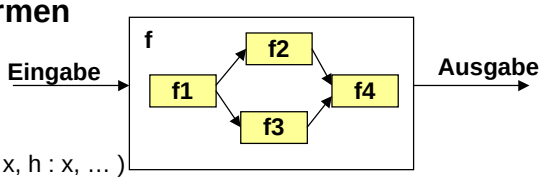
Datentypen

- **Typbegriff**
  - im Zusammenhang mit Programmiersprachen hat der Begriff ***Typ*** oder auch ***Datentyp*** eine zentrale Bedeutung
- **Man unterscheidet grob:**
  - ***einfache*** Datentypen
  - ***zusammengesetzte*** Datentypen
- **Einfachen Datentypen in Modula-3**
  - **Ganze Zahlen**
  - **Zeichen**
  - **Texte**
  - **Wahrheitswerte**
  - **Gleitkommazahlen**

3

Vernetzung von Funktionen

- **Um komplexe Ausdrücke zu berechnen,**
  - werden Funktionen vernetzt.
- **Funktionalformen (oder Funktionale)**
  - beschreiben "Vernetzungsmuster"
- **Beispiele für Funktionalformen**
  - **Komposition**
    - f o g : x = g : (f : x)
  - **Konstruktion**
    - [f, g, h, …] : x = (f : x, g : x, h : x, …)
  - **Bedingung**
    - if t then f else g : x =



f : x , falls t : x = true
g : x , falls t : x = false
? , sonst

3

Rekursion

- **Idee:**
  - allgemein bezeichnet man mit ***Rekursion*** die Definition eines Problems, einer Funktion oder ganz allgemein eines Verfahrens "***durch sich selbst***"
- **Rekursive Funktion**
  - darunter verstehen wir Funktionen, die sich ***selbst wieder aufrufen***
- **Indirekte Rekursion:**
  - ***Indirekte*** Rekursion kann in einem System von Funktionen definiert werden, die Seite an Seite vereinbart werden und sich ***gegenseitig stützen***.
- **Anmerkung:**
  - Es darf keine ***unkontrollierte*** (unendliche) Rekursion entstehen
  - Jeder rekursive Funktionsaufruf gehört in eine ***bedingte Anweisung***
  - Rekursionsabbruch

4

Variable und Wertzuweisung

- **Variable**
  - **logischer** Speicherplatz mit seinem Wert
  - besitzt einen **Namen**, unter dem man die Variable ansprechen kann
  - Datentyp legt fest, welche **Werte** eine Variable annehmen kann und welche **Operationen** darauf ausgeführt werden können
  - Variable kann als Behälter betrachtet werden
    - hat einen Wert und einen Typ
- **Wertzuweisung**
  - dient dazu, den Wert einer Variablen zu verändern
  - Der Ausdruck wird zuerst ausgewertet, das Ergebnis wird anschließend der Variable zugewiesen
    - In diesem Zusammenhang sprechen wir oft von der rechten und der linken Seite einer Zuweisung:
    - (right-hand side - RHS, left-hand side - LHS).
  - LHS und RHS müssen Typkompatibel sein

4

Abstraktion durch Prozeduren

- **Fachlich ist eine Prozedur**
  - die programmiersprachliche ***Realisierung*** eines Algorithmus.
- **Softwaretechnisch**
  - kann eine Prozedur zunächst als ***benannte Anweisungsfolge*** verstanden werden.
- **Die Grundidee ist,**
  - den Namen der Prozedur "***stellvertretend***" für diese Anweisungsfolge zu verwenden.
- **Die Prozedur ist eine wesentliche Umsetzung des Konzepts der *algorithmische Abstraktion* (auch *Prozeßabstraktion* genannt):**
  - Statt einer expliziten Anweisungsfolge (der genauen Verarbeitungsvorschrift) wird ein davon ***abstrahierender*** Name verwendet.

4

Parameterübergabearten

- **Allgemein gibt es folgende Parameterarten für Prozeduren**
  - ***Eingabeparameter***
  - ***Ausgabeparameter***
  - ***Ein- / Ausgabeparameter***
- **Der formale Parameter beim *Call by Value* ist ein**
  - ***Wertparameter***
  - realisieren Eingangsparameter
  - Veränderungen des formalen Parameters in der Prozedur haben nur ***lokale Auswirkung***. Der aktuelle Parameter bleibt unverändert.
- **Der formale Parameter beim *Call by Reference* ist ein**
  - ***Variablenparameter*** (oder Referenzparamter)
  - realisieren Ausgangs und Ein- / Ausgangsparameter
  - Jede Änderung des formalen Parameters ist ***direkt*** im aktuellen Parameter wirksam.

4 Gültigkeitsbereich und Lebensdauer

- **Gültigkeitsbereich (scope) eines Bezeichners**
  - der ***statische Teil*** des Programms, in dem der Bezeichner mit exakt ***gleicher Bedeutung*** verwendet werden darf
  - wird auch ***Sichtbarkeitsbereich*** oder ***Namensraum*** genannt
  - Bezeichner ist in seinem Sichtbarkeitbereich gültig
- **Lebensdauer eines Objekts (Variable, Prozedur)**
  - bezieht sich auf den zur ***Programmlaufzeit*** belegten Speicherplatz
- **Ein Objekt heißt lokal in Block B,**
  - wenn es im Block B deklariert ist.
- **Ein Objekt heißt global,**
  - wenn es auf Modulebene deklariert ist.
- **Ein Objekt heißt global relativ zu B,**
  - wenn es in B gültig ist, aber nicht lokal in B ist

5 Arten von Kontrollanweisungen

- **Sequenz von Aktionen:**
  - Eine Aktion wird ***nach der anderen*** abgearbeitet.
- **Auswahlanweisungen kommen vor als:**
  - Einwegauswahl (one-way selection) IF-THEN
  - Zweiwegauswahl (two-way-selection) IF-THEN-ELSE
  - Mehrfachselektion (multiple selection) CASE
- **Wiederholungsanweisungen werden benötigt,**
  - um ***iterative Algorithmen*** zu formulieren.
  - Schleife mit vorheriger Prüfung WHILE und FOR
  - Schleige mit nach nachfolgender Prüfung REPEAT-UNTIL
  - Schleife ohne Prüfung LOOP (EXIT)

6

Einteilung von Typen

- **Einfache und zusammengesetzte Datentypen:**
  - ***Einfache Datentypen*** erlauben ***keinen*** Zugriff auf ihre innere Struktur. Ihre Werte können unmittelbar notiert werden.
  - **Zusammengesetzte Datentypen** sind aus anderen Datentypen aufgebaut. Auf ihre einzelnen Elemente kann zugegriffen werden.
- **Vorgegebene und benutzerdefinierte Datentypen:**
  - **Vorgegebene Datentypen** haben einen vordeklarierten Namen und können unmittelbar zur Deklaration von Variablen verwendet werden.
  - **Benutzerdefinierte** Datentypen haben einen selbst definierten Namen und müssen deklariert werden.
- **Statische und dynamische Datentypen**
  - **Statisch**: Größe der Typobjekte ist von vornherein ***bekannt***
  - **Dynamisch**: Größe ist während der Laufzeit ***veränderbar***

6

Benutzerdefinierte Typen

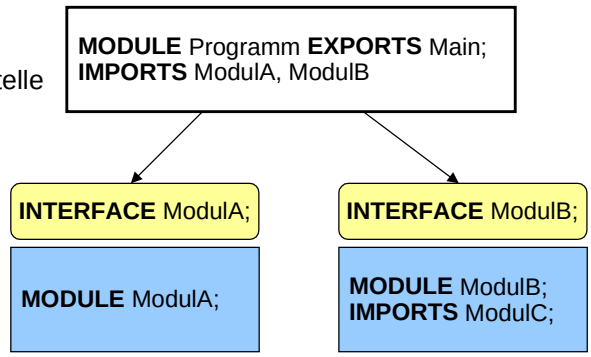
- **Benutzdefinierte einfache Typen,**
  - `TYPE Zeit = REAL;`
- **Unterbereichstyp (subrange type)**
  - `TYPE Index = [1..10];`
- **Aufzählungstyp (emumeration type)**
  - `TYPE Ampelfarbe = {rot, gelb, gruen};`
- **Array-Typ:**
  - Aneinanderreihung von ***gleichartigen Elementen***, wobei auf die Komponenten mit Hilfe eines ***Index*** zugegriffen wird.
- **Record-Typ**
  - Darstellung ***inhomogener***, aber ***zusammengehöriger*** Informationen.
- **Mengen-Typ**
  - Mengen sind ***ungeordnete*** Sammlungen von Elementen

6 Dynamische Datentypen/ Prozedurtyp

- **Eigenschaften von Objekten dynamischer Datentypen**
  - ***Lebensdauer*** ist nicht an Prozedur oder Moduls gebunden
  - werden ***explizit erzeugt*** und eventuell auch wieder beseitigt
  - sie werden in einem speziell dafür vorgesehenen Speicherbereich angelegt (***Halde oder Heap***)
  - können in prinzipiell ***beliebiger*** Menge geschaffen werden
  - haben im Gegensatz zu den bisherigen Objekten ***keinen festen Bezeichner***
  - sie werden stattdessen über einen ***Zeiger (Pointer)*** identifiziert
- **Prozedurtyp**
  - Ein Prozedurtyp definiert eine ***Signatur.***
  - Die Werte eines Prozedurtyps sind ***Prozeduren***, die der vorgegebenen Signatur entsprechen.
  - Entsprechend können Variablen als ***Prozedurvariablen*** deklariert werden.

7 Modulkonzept

- **Entstanden aus der Notwendigkeit,**
  - große Programmtexte in für den ***Übersetzer faßliche Einheiten*** zu zerlegen,
  - Modulkonzept ist zum zentralen ***Organisationskonzept*** für Entwürfe und Programmtexte geworden.
- **Konzept:**
  - Trennung von Schnittstelle und Implementierung
  - IMPORT-Beziehung zwischen Modulen



7

Information Hiding

- **Prinzip:**
  - Es werden nur die Informationen zur Verfügung gestellt, die ***absolut notwendig*** sind!
  - Alle anderen, insbesondere die ***wichtigen Informationen*** werden ***versteckt***!
  - Der Zugriff auf diese Informationen geschieht über "***Vermittler***".
- **Information Hiding wird realisiert durch**
  - Datenkaspelung in Objektmodulen
  - Abstrakte Datentypen

7

Objektmodul - Datenkapsel

- **Die zentrale Idee:**
  - ***Trenne*** die konkrete Realisierung (i.e. Implementation) einer Datenstruktur von ihren sichtbaren Eigenschaften.
- **Merkmale:**
  - Eine Datenstruktur wird in einem Modul ***eingekapselt***.
  - An der Schnittstelle des Moduls sind nur ***Operationen sichtbar***, die den allgemeinen Umgang mit der Datenstruktur beschreiben.
  - Die Datenstruktur selbst ist ***verborgen***.
- **Jedes Objektmodul**
  - beschreibt und realisiert nur eine ***einzige sog. abstrakte Datenstruktur***.

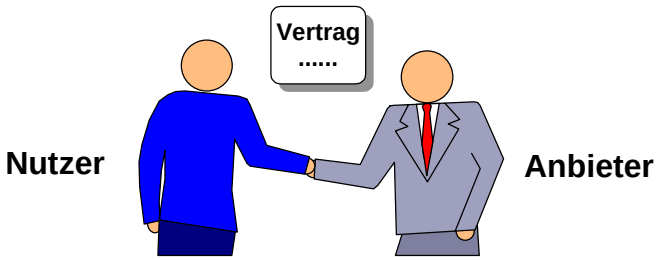
7

Konzept Abstrakter Datentyp

- **Kann als *Generalisierung* des Objektmoduls (Datenkapsel) betrachtet werden.**
  - Anstatt eines Objektes wird ein Typ (für diese Objekte) definiert.
- **Betrachten wir den ADT als (formale) Spezifikation eines Typs,**
  - dann entwerfen wir ihn durch Angabe von
  - ***Typnamen***
  - ***Signaturen*** (Operationen)
    - zum Erzeugen von Objekten, zum Verändern etc.
  - ***Axiome***
    - formulieren den semantischen Zusammenhang der Operationen.
  - ***Vorbedingungen***
    - geben an, in welchem Zustand welche Operationen gültig sind.

8

Idee des Vertragsmodells



- **Vertrag**
  - zwischen Nutzer und Anbieter einer Operation regelt, wer der beiden Partner welche ***Verpflichtungen*** einhalten muß (und welchen ***Nutzen*** er dadurch hat)
- **Operation**
  - arbeitet korrekt, wenn sie vertragsgemäß ***benutzt*** bzw. ***realisiert*** wird.

8

Zusicherungen

- **Zusicherungen**
  - sind eine Technik, um eine bestimmte Art von Verträgen zwischen Anbieter und Nutzer zu formulieren.
- **Zusicherungen werden formuliert als**
  - ***Vorbedingungen*** für Operationen
  - ***Nachbedingungen*** von Operationen
  - ***Invarianten*** von abstrakten Datentypen
- **Zusicherungen**
  - erhöhen die ***Benutzbarkeit***, indem sie diese formaler definieren
  - verbessern die ***Testbarkeit***
  - verbessern die ***Fehlersuche*** (debugging)
  - verlangen vom Entwickler ein ***abstraktes*** Denken

8

Realisierung von Zusicherungen in M3

- **Mit Hilfe des Pragmas ASSERT**

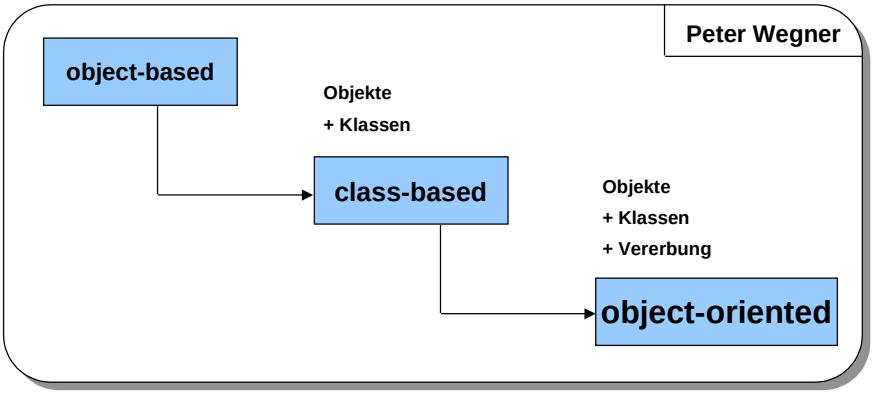
```
PROCEDURE EntnehmeText (VAR o: Ordner): TEXT =
VAR t : TEXT;
BEGIN
 <* ASSERT NOT IstLeer(o) *>
 . . .
 <* ASSERT NOT IstVoll(o)  *>
 <* ASSERT Invariante(o)   *>
END EntnehmeText;
```

- **Mit Hilfe von Ausnahmen**

```
PROCEDURE EntnehmeText (VAR o: Ordner) : TEXT
                        RAISES {As. Violated} =
VAR t : TEXT;
BEGIN
  As.Require (NOT IstLeer(o), "OrdnerADT.EntnehmeText");
  . . .
  As.Ensure (NOT IstVoll(o), "OrdnerADT.EntnehmeText");
  As.Ensure (Invariante(o), "OrdnerADT.EntnehmeText");
END EntnehmeText;
```

9

Merkmale objektorientierter Sprachen



Ada
Modula-2

CLU
(Ada)
(Modula-2)

Simula
Smalltalk-80
Eiffel
C++
Java
Modula-3

9

Objekt, Nachricht, Klasse, Vererbung

- **Ein Objekt ist eine Datenkapsel, die aus zwei Teilen besteht**
  - Der Wert der Daten repräsentiert den **Zustand** des Objekts
  - Daten können nur mithilfe von **Operationen** verändert werden
- **Objekte kommunizieren miteinander,**
  - dadurch daß sie **Nachrichten** versenden und Nachrichten empfangen.
- **Gleichartige Objekte werden in einer Klasse beschrieben**
  - **Speicherstruktur**
  - **Operationen** (Methoden, Routinen)
  - konkrete Klassen, abstrakte Klassen
- **Gemeinsame Eigenschaften verschiedener Klassen**
  - werden in einer **eigenen Klasse** zusammengefaßt und definiert, und anschließend an diese vererbt.
  - Einfachvererbung - Mehrfachvererbung

9

Das dynamische Binden

```
g : Geo_Object;
r : Rectangle;
c : Circle;

r.Create; c.Create

g := r;
g.contains (p);

g := c;
g.contains (p);
```

```
class Geo_Object

feature
  contains (p : Point) : Boolean is
    deferred
    end;
end -- class Geo_Object
```

```
class Rectangle

feature
   contains (p : Point) :Boolean is
     ...
end -- class Rectangle
```

```
class Circle

feature
  contains (p : Point) :Boolean is  ...
end -- class Circle
```

Dynamisches Binden heißt:

die **richtige** Implementierung zur **Laufzeit** finden

9

Klassen in Modula-3 - Objekttypen

```
INTERFACE Geo_Object;
IMPORT Point;

TYPE T =  ROOT OBJECT
  METHODS
    moveTo (p : Point.T);
    contains (p : Point.T) : BOOLEAN;
  END;
END Geo_Object.
```

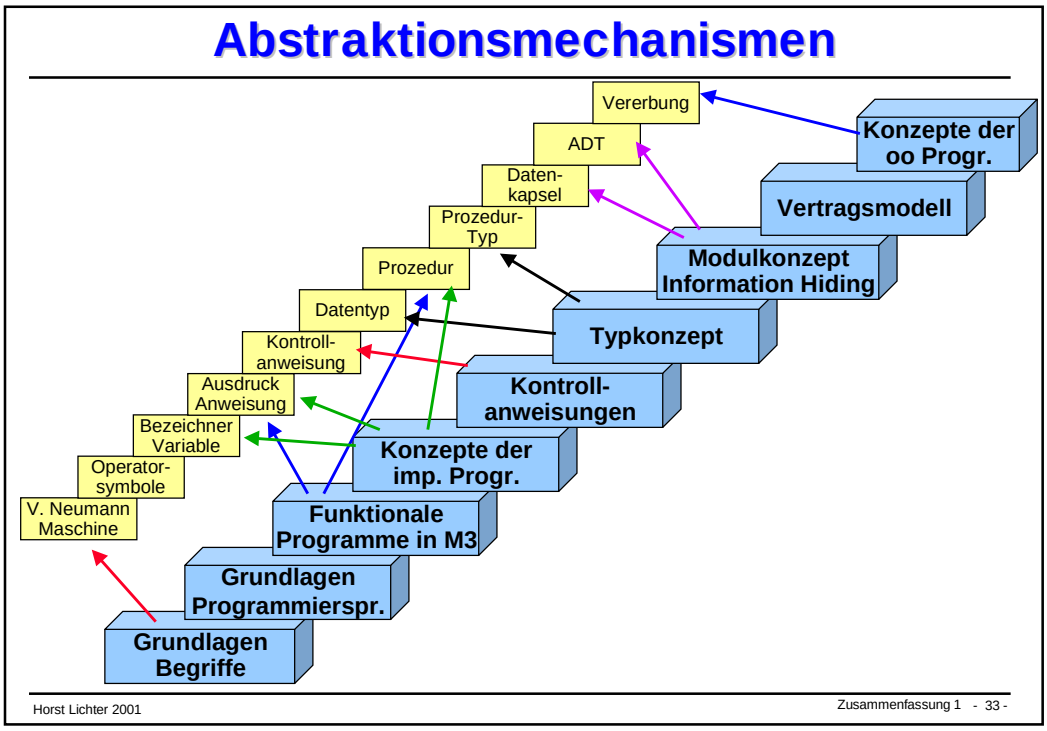
Abstrakte Klasse

```
INTERFACE Rectangle;
IMPORT Geo_Object, Point;

TYPE T <: Public;
     Public = Geo_Object.T OBJECT
              METHODS
                area() : INTEGER;
                height () : INTEGER;
                setOrigin (p : Point.T);
                getOrigin(): Point.T;
                setCorner(p : Point.T);
                getCorner(): Point.T;
                asText () : TEXT;
              END;
END Rectangle.
```

```
INTERFACE DisplayableRectangle;
IMPORT Rectangle;

TYPE T <: Public;
     Public = Rectangle.T OBJECT
              METHODS
                setColor(c : TEXT);
                getColor() : TEXT;
                draw(): TEXT;
              END;
END DisplayableRectangle.
```


Abstraktionsmechanismen
Vererbung
ADT
Daten-kapsel
Prozedur-Typ
Prozedur
Datentyp
Kontroll-anweisung
Ausdruck Anweisung
Bezeichner Variable
Operator-symbole
V. Neumann Maschine
Konzepte der oo Progr.
Vertragsmodell
Modulkonzept Information Hiding
Typkonzept
Kontroll-anweisungen
Konzepte der imp. Progr.
Funktionale Programme in M3
Grundlagen Programmierspr.
Grundlagen Begriffe
Horst Lichter 2001
Zusammenfassung 1 - 33 -