Modularisierung und Module

- **Modulkonzept**
  - Export-Schnittstelle
  - Import-Schnittstelle
- **Modulrumpf**
- **Austausch der Implementierung**
- **Diskussion modularer Programme**

Modul-konzept

Vorteile modularer Programme

- **Module können von *unterschiedlichen* Personen entwickelt und gepflegt werden.**
  - Software-Entwicklung ist Team-Arbeit!
- **Module können einzeln *getestet* werden.**
  - Test großer Programme ist extrem aufwendig!
- **Module können geordnet zum Gesamtsystem *integriert* werden.**
- **Eine Implementierung eines Moduls kann leicht durch eine neue Implementierung *ersetzt* werden.**
  - z.B. durch eine effizientere Implementierung
- **Module können in verschiedenen Programmen *wiederverwendet* werden (Modul-Bibliothek).**
  - Dies senkt die Kosten für die Entwicklung!

Modul-konzept

Module

- **Neuere imperative Sprachen sehen Module vor**
- **Entstanden aus der Notwendigkeit,**
  - große Programmtexte in für den ***Übersetzer faßliche Einheiten*** zu zerlegen,
  - Modulkonzept ist zum zentralen ***Organisationskonzept*** für Entwürfe und Programmtexte geworden.
- **Module werden hier als**
  - *Konstruktionshilfsmittel* der Sprache eingeführt.
- **Die Diskussion,**
  - wie das Modulkonzept genutzt werden sollte, folgt in einem eigenen Kapitel.

Modul-konzept

Definition: Modul

- **programmiersprachliche Definition:**
  - Ein Modul ist die ***Zusammenfassung von Konstanten, Datentypen, Variablen und Prozeduren*** zu einer Einheit. Soll ein Modul von einem anderen benutzt werden, so muß man angeben, welche Teile der Schnittstelle dieses Moduls von ***außen sichtbar*** sein sollen und welche nicht. Grundsätzlich bleibt aber die Implementierung eines Moduls, also die konkrete Realisierung der Datentypen und Prozedurrümpfe, vor allen anderen Modulen ***verborgen***.
- **methodische Definition:**
  - Unter einem Modul verstehen wir eine ***Sammlung von Objekten und Algorithmen*** mit der Eigenschaft, daß ihre Kommunikation mit der Außenwelt nur über eine klar ***definierte Schnittstelle*** erfolgt. Das ***Zusammensetzen*** mehrerer Module zu einer Gesamtlösung darf keine Kenntnis ihres ***inneren Aufbaus*** voraussetzen, und die Korrektheit eines Moduls muß ohne Kenntnis seiner Einbettung in die Gesamtlösung nachprüfbar sein.

Modul-konzept

Erinnerung: Lebensdauer

- **Prozedur:**
  - Alle Objekte im Namensraum einer Prozedur existieren nur solange die Prozedur aktiv ist.
  - Bei jedem neuen Aufruf einer Prozedur werden u.a. die lokalen Variablen neu angelegt.
- **Modul:**
  - Module sind ***statisch***, d.h. ihr Namensraum existiert solange das Programm oder die Anwendung ***insgesamt*** aktiv ist.
  - Variablen, die im Deklarationsteil eines Moduls eingeführt werden, haben die gleiche Lebensdauer wie das Modul; sie heißen ***global***.
- **In Modula-3**
  - können Module ***nicht*** ineinander geschachtelt werden!
  - Bei Prozeduren ist dies möglich!

Modul-konzept

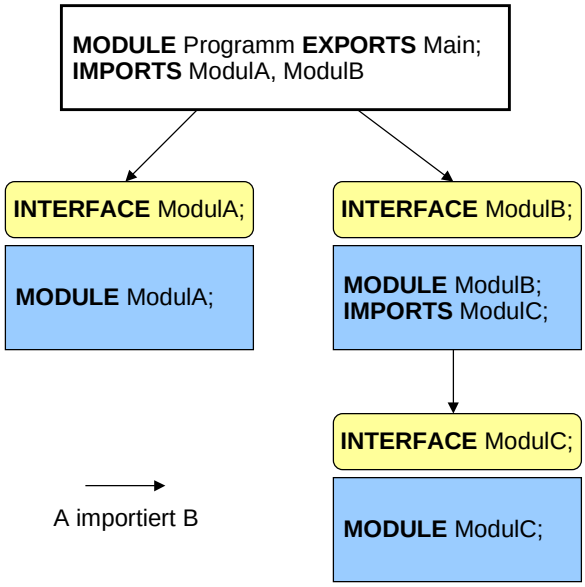
Aufbau von Modula-3 Programmen

- **Modula-3 Programm**
  - besteht wenigstens aus einem ***Modul***, dem ***Hauptmodul***
- **Modul-Aufbau**
  - ein Modul besteht (bis auf das Hauptmodul) aus
    - ***Schnittstelle*** (interface)
      - definiert, was ein Modul exportiert
      - Exportschnittstelle
    - ***Implementierung*** (body, Rumpf)
      - enthält die Implementierung der exportierten Elemente
      - versteckt die Implementierung
- **Bisher**
  - bestanden unsere Programme lediglich aus einem Modul, dem Hauptmodul und weiteren Prozeduren

Modul-konzept

Modul-Hierarchie

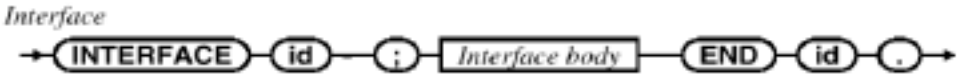
- ein Modul kann Elemente anderer Module ***benutzbar machen***
  - ein Modul ***importiert*** dazu andere Module
  - von einem importierten Modul ist nur die ***Schnittstelle*** sichtbar

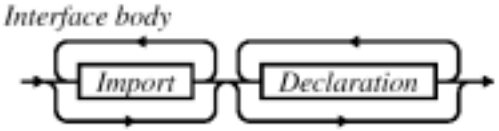


Modul-konzept

Die Export-Schnittstelle

- **Wie sieht eine Export-Schnittstelle eines Moduls aus?**





- **Alle in der Schnittstelle deklarierten Objekte werden vom Modul *exportiert***
  - können von importierenden Modulen ***verwendet*** werden.
- **Hinweis**
  - alles was in der Schnittstelle "***versprochen***" wird, muß auch von der Implementierung realisiert werden

Modul-konzept

Beispiel: Export-Schnittstelle

```
INTERFACE SIO;

IMPORT Fmt, Rd, Wr, Word;

...

PROCEDURE GetChar(rd: Reader := NIL): CHAR RAISES {Error};
(* Read next character from stream rd and return it. *)

PROCEDURE PutChar(ch: CHAR; wr: Writer := NIL);
(* Write ch to outputstream wr. *)

PROCEDURE GetText(rd: Reader := NIL; len: CARDINAL): TEXT;
(* Read a sequence of len characters from rd and return them. If there are not
enougth characters return what is there. *)

PROCEDURE PutText(t: TEXT; wr: Writer := NIL);
(* Write character sequence t to outputstream wr. *)

PROCEDURE GetLine(rd: Reader := NIL): TEXT RAISES {Error};
(* Read a full line of text terminated by the next RETURN from
   inputstream rd and return it (without RETURN!). *)
. . .
END SIO.
```


Modul-konzept

IMPORT-Schnittstelle - 1

- **Um Programmobjekte über Modulgrenzen zu verwenden,**
  - müssen sie ***gezielt angefordert*** werden.
  - Die IMPORT-Klausel dient dazu,
    - die Dienste in einem anderen Modul ***sichtbar*** zu machen.
- **Mögliche IMPORT-Varianten:**
  - importieren ***aller*** Dienste eines Moduls
  - importieren ***aller*** Dienste eines Moduls unter einem ***Alias-Namen***
  - importieren nur der Dienste eines Moduls, die ***tatsächlich*** vom importierenden Modul verwendet werden

Modul-konzept

IMPORT-Klausel - 2

- **Beispiele für Importe:**
  - wird nicht selektiv importiert, muß der Modulname als ***Qualifikator*** verwendet werden

```
IMPORT Text;                          (* import aller Deklarationen *)
IMPORT Rd AS Reader;                  (* import mit Alias-Namen *)
FROM SIO IMPORT                       (* selektives Importieren *)
  (*PROCS*) PutLine, PutText, GetChar;
…
VAR eingabestrom : Reader.T;
…
n := Text.Length(t1);
…
PutText("abcdefg");
```

Modul-konzept

Diskussion: Import-Varianten

- **Globales Importieren**

  An ***jeder Stelle*** im Programmtext ist ersichtlich, wo das jeweilige Objekt deklariert ist.

  ```
  Text.Length(t1);   Length(m3);   List.Length(l1);
  ```

  Identisch deklarierte Bezeichner verschiedener Module können verwendet werden.

  Zum Teil erheblich ***längere Schreibweise.***

  Erst mit einem Werkzeug kann einfach ermittelt werden, was ***tatsächlich*** alles von einem Modul verwendet wird.

- **Selektives Importieren**
  - ***kürzere*** Schreibweise
  - Es ist alles das, was verwendet wird, auch ***explizit angegeben***
  - Dies erhöht die Änderbarkeit

    Es ist nicht direkt ersichtlich, woher ein Dienst kommt.

Modul-konzept

Regeln

- **1. Schnittstelle und Implementierung eines Moduls sind in *unterschiedlichen* Dateien (Programmtextdatei) enthalten.**
  - Jede Programmtextdatei bildet eine ***Übersetzungseinheit*** und kann vom Übersetzer getrennt behandelt werden.
- **2. Import**
  - alle Dienste: Qualifikation zeigt Herkunft des Dienstes
  - selektiver Import: Erhöht die Änderungsfreundlichkeit
- **3. Zyklische Importe sind verboten!**
  - A importiert B, B importiert A
  - zyklischer Import ist ein Hinweis auf eine ***schlechte Modularisierung***

Modul-konzept

Import für Schnittstelle oder für Rumpf

- **Import zur Schnittstellendefinition**
  - z.B. Typ für Deklaration von Formalparameter oder Ergebnis
  - Konstante für Vorberechnung von Parametern
- **Import für Rumpfrealisierung**
  - z.B. Typ für die Realisierung einer modulrumpflokalen Variablen
  - Prozedur/Funktion als Hilfe für die Implementierung einer Schnittstellenoperation oder des Anweisungsteils

Modul-rumpf

Implementierung eines Moduls

- **Regel:**
  - Die Implementierung einer Schnittstelle realisiert ***alle*** in der Schnittstelle ***deklarierten*** Prozeduren (Funktionen).
- **EXPORTS-Klausel**
  - gibt an, welche Schnittstelle ein Modul realisiert

```
MODULE Geometrie EXPORTS Geometrie;
```

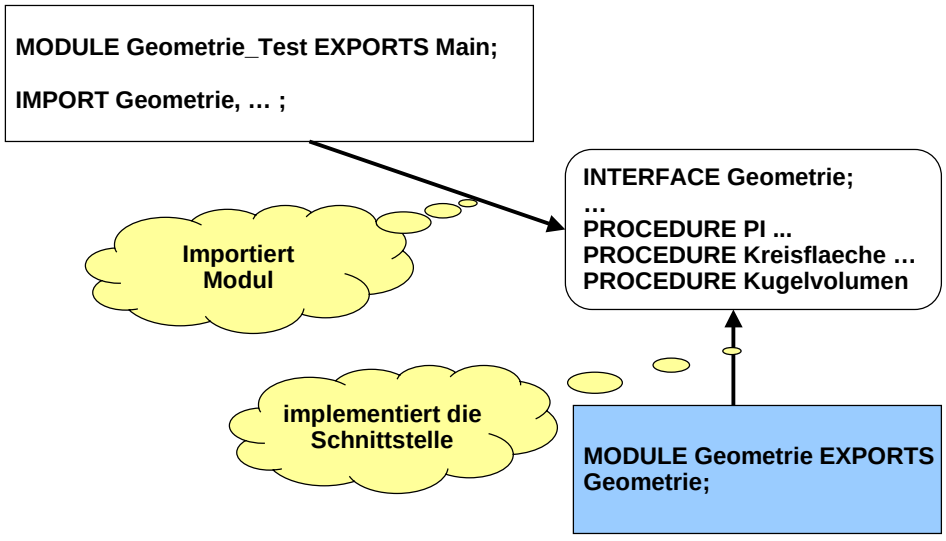
  - Fehlt die EXPORTS-Klausel, dann realisiert das Modul eine Schnittstelle ***gleichen Namens.***

```
MODULE Geometrie;
```

- **Modul-Initialisierung**
  - Im ***Block*** eines Moduls wird das Modul initialisiert.
  - Regel: Ein importiertes Modul wird vor dem importierenden Modul initialisiert.

Modul-rumpf

Beispiel



Modul-rumpf

Beispiel: Schnittstelle

```
MODULE Geometrie_Test EXPORTS Main;
IMPORT Geometrie, SIO;

VAR radius : REAL;
BEGIN
  SIO.PutText ("Geben Sie bitte einen Radius ein: ");
  radius := SIO.GetReal();
  SIO.PutText ("Kreisflaeche: ");
  SIO.PutReal (Geometrie.Kreisflaeche(radius));
  SIO.Nl();
  SIO.PutText ("Kugelvolumen: ");
  SIO.PutReal (Geometrie.Kugelvolumen(radius));
END Geometrie_Test.
```

```
INTERFACE Geometrie;

PROCEDURE PI () : REAL;
PROCEDURE Kreisflaeche(r :REAL): REAL;
PROCEDURE Kugelvolumen(r : REAL): REAL;

END Geometrie.
```


Modul-rumpf

Beispiel: Implementierung

```
MODULE Geometrie EXPORTS Geometrie;

VAR pi : REAL;

PROCEDURE PI () : REAL =
BEGIN
  RETURN pi;
END PI;

PROCEDURE Kreisflaeche(radius :REAL): REAL =
BEGIN
  RETURN (pi * radius * radius);
END Kreisflaeche;

PROCEDURE Kugelvolumen(radius : REAL): REAL =
BEGIN
  RETURN ( (4.0/3.0) * pi * radius * radius * radius );
END Kugelvolumen;

BEGIN
  pi := 3.14147;
END Geometrie.
```

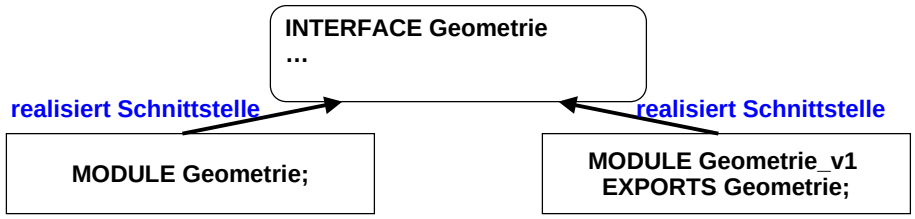
Interne Variable

Realisierung der Prozeduren / Funktionen der Schnittstelle

Initialisierung der Moduls

Austausch der Implementierung

Zwei Rümpfe für die gleiche Schnittstelle




```
% Makefile ...
…

% module(Geometrie)

interface(Geometrie)
implementation (Geometrie_v1)

implementation (Geometrie_Test)
program (Geometrie_Test)
```

```
MODULE Geometrie_v1 EXPORTS Geometrie;

PROCEDURE PI () : REAL =
BEGIN
  RETURN 3.14147;
END PI;

PROCEDURE Kreisflaeche(r :REAL): REAL = ...

PROCEDURE Kugelvolumen(r : REAL): REAL = ...
BEGIN
END Geometrie_v1.
```


Diskussion modularer Programme

Vorteile modularer Programme

- **Vorteile modularer Programme**
  - Module können von ***unterschiedlichen*** Personen entwickelt und gepflegt werden.
    - Software-Entwicklung ist Team-Arbeit!
  - Module können einzeln ***getestet*** werden.
    - Test großer Programme ist extrem aufwendig!
  - Module können geordnet zum Gesamtsystem ***integriert*** werden.
  - Eine Implementierung eines Moduls kann leicht durch eine neue Implementierung ***ersetzt*** werden.
    - z.B. durch eine effizientere Implementierung
  - Module können in verschiedenen Programmen ***wiederverwendet*** werden (Modul-Bibliothek).
    - Dies senkt die Kosten für die Entwicklung!

Diskussion modularer Programme

Modulkonzept

- **Module sind Sammlungen von Programmobjekten und Algorithmen:**
  - Sie sind keine ***direkt aufrufbaren*** Programmeinheiten (wie Prozeduren).
  - Sie sind eine Einheit für die ***Übersetzung***.
- **Als Sammlung sollen sie *keine* beliebige Anordnung sein**
  - Kriterien für die Zusammenstellung eines Moduls müssen geklärt werden.
  - Module verbergen ***Implementierungen*** d.h. ihren inneren Aufbau und zeigen nur ihre Schnittstelle:
  - Es gibt unterschiedlich ***starke Möglichkeiten*** des Verbergens.
  - Der Aufbau einer Schnittstelle unterliegt bestimmten Kriterien.
- **Module benutzen andere Module:**
  - Das Zusammenspiel verschiedener Module bezeichnet man auch als ***Architektur***.
  - Die ***Import-Beziehungen*** koppeln Module miteinander.

Was haben wir gelernt?

- **Module als Sprachkonstrukt moderner imperativer Programmiersprachen**
- **Modul Schnittstelle - Rumpf, verschiedenen Rümpfe zu einer Schnittstelle**
- **Vorteile der Modularisierung bzgl. Qualität und Effizienz der Softwareerstellung bzw. bzgl. Qualität des resultierenden Programmsystems**
- **Modulhierarchie über Importbeziehungen**
- **Implementierung eines Moduls: Realisierung der Dienste und Initialisierung**

Glossar

- **Exportschnittstelle, Importschnittstelle eines Moduls, Rumpf eines Moduls**
- **getrennte Übersetzung von Schnittstelle und Rumpf**
- **programmiersprachliche und methodische Definition eines Moduls**
- **statische Lebensdauer von Modulvariablen (immer im Rumpf)**
- **Import für Schnittstelle oder für Rumpf**
- **Schnittstellenimplementierung, Implementierung der Initialisierung**
- **schlechter Sprachgebrauch Interface (Module), (Implementation) Module**