5. Namen und Eigenschaften

Namen in der Beschreibung repräsentieren **Objekte** als Teil der Bedeutung des Textes

Auftreten von Namen sind an **Objekte** gebunden.

Die Bindung wird durch **Gültigkeitsregeln** der Beschreibungssprache geregelt. Z. B:

- Struktur-Namen werden in der ganzen Beschreibung identisch gebunden.
- Feld-Namen werden für jede Struktur separat gebunden.
- Typ-Namen sind angewandte Auftreten von Namen.
  Sie erfordern, dass es irgendwo ein definierendes Auftreten desselben Names gibt.

einige Auftreten von Namen: **einige Bindungen:** **einige Objekte:**

```
Customer ( addr:    Address;
           account: int;
)
Address  ( name:    String;
           zip:     int;
)
Article  ( name:    String;
           price:   int;
)
```

- eine Struktur (namens Address)
- ein Feld (Namens name)
- eine Struktur (namens Article)
- ein Feld (namens name)
- ...

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 501

Ziele:

Zusammenhang der Aufgaben

in der Vorlesung:

Am Beispiel Begriffe erläutern:

- Objekt der Beschreibung,
- Name von Objekten,
- Bindung eines Namen an ein Objekt,
- Gültigkeit von Bindungen

nachlesen:

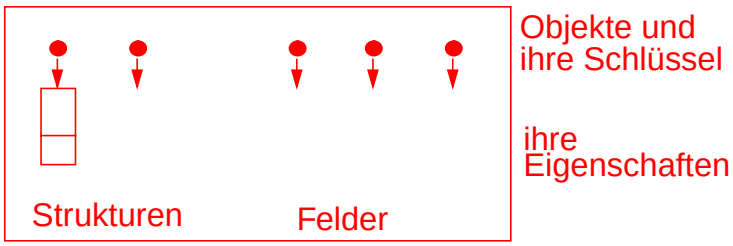
GdP-3.1 ff

Schlüssel und Eigenschaften

Eli Werkzeuge implementieren Eigenschaften von Objekten und Umgebungen

OIbjekte werden durch **Schlüssel** repräsentiert.
Eigenschaften werden ihnen zugeordnet.

Strukturen haben eine Eigenschaft „Umgebung“



Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 502

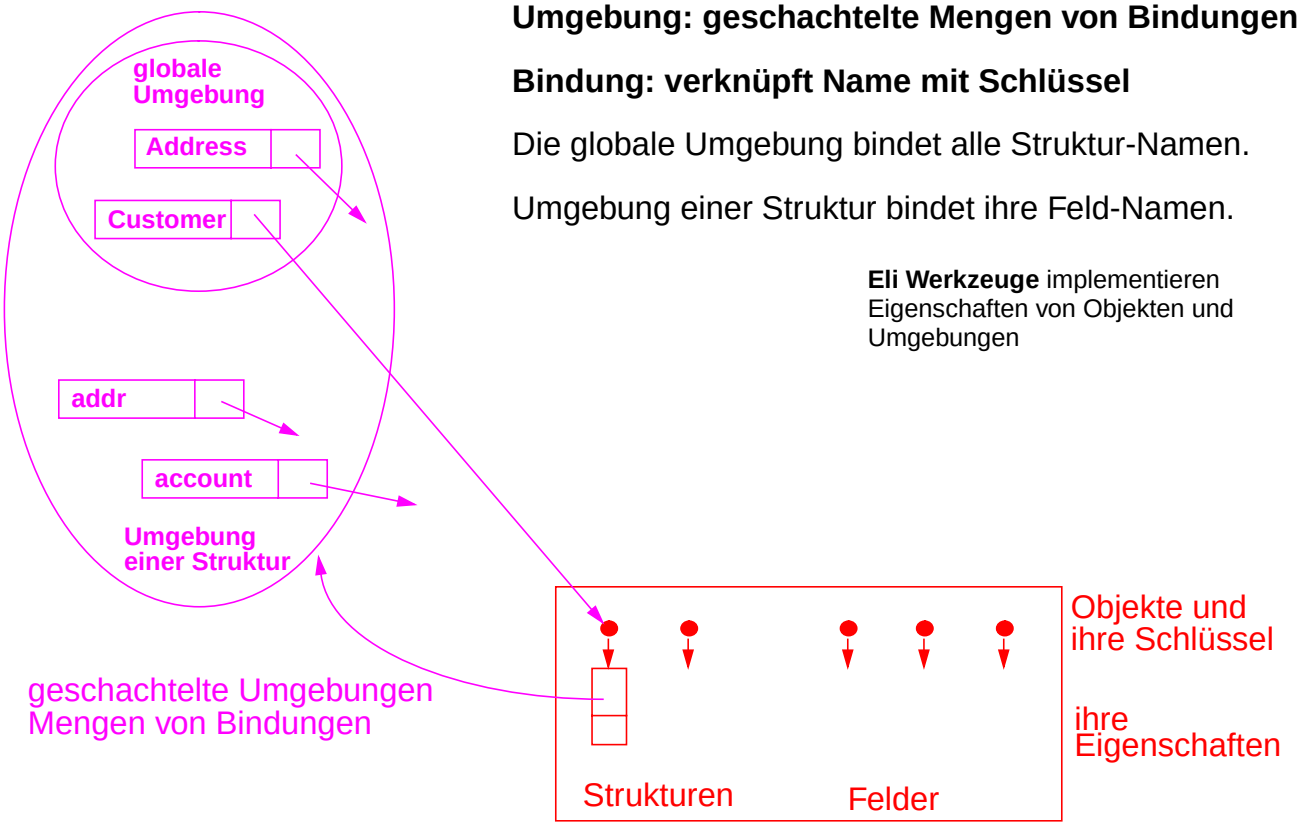
Ziele:

Übersicht zu Eigenschaften von Objekten

in der Vorlesung:

Themen der Folie erläutern:

Bindungen und Umgebungen



Umgebung: geschachtelte Mengen von Bindungen

Bindung: verknüpft Name mit Schlüssel

Die globale Umgebung bindet alle Struktur-Namen.

Umgebung einer Struktur bindet ihre Feld-Namen.

Eli Werkzeuge implementieren Eigenschaften von Objekten und Umgebungen

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 503

Ziele:

Übersicht zu Umgebungen und Bindungen

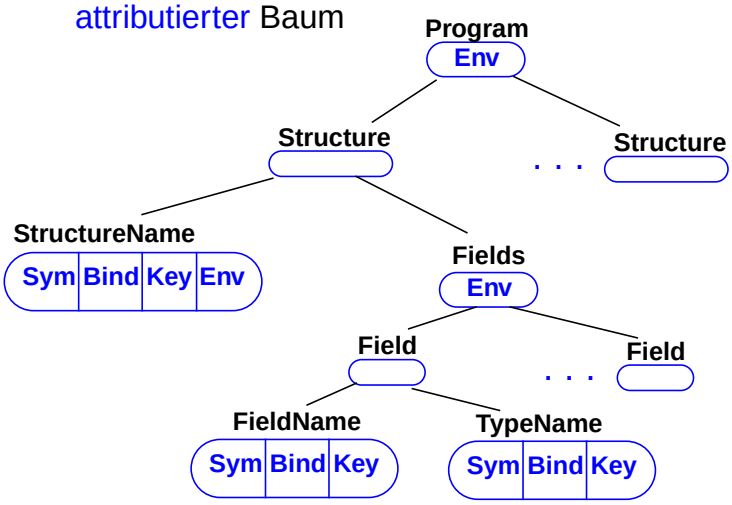
in der Vorlesung:

Themen der Folie erläutern:

Attributierter Baum für Namensanalyse

Attribute der Baumknoten
beschreiben Eigenschaften des repräsentierten Konstruktes

attributierter Baum



Program hat die globale Umgebung

StructureName und Fields haben die Umgebung der Struktur

Jeder Namensknoten hat Attribute für

- die Codierung des Symbols,
- die Bindung seines Namens, und
- sein Schlüssel

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 504

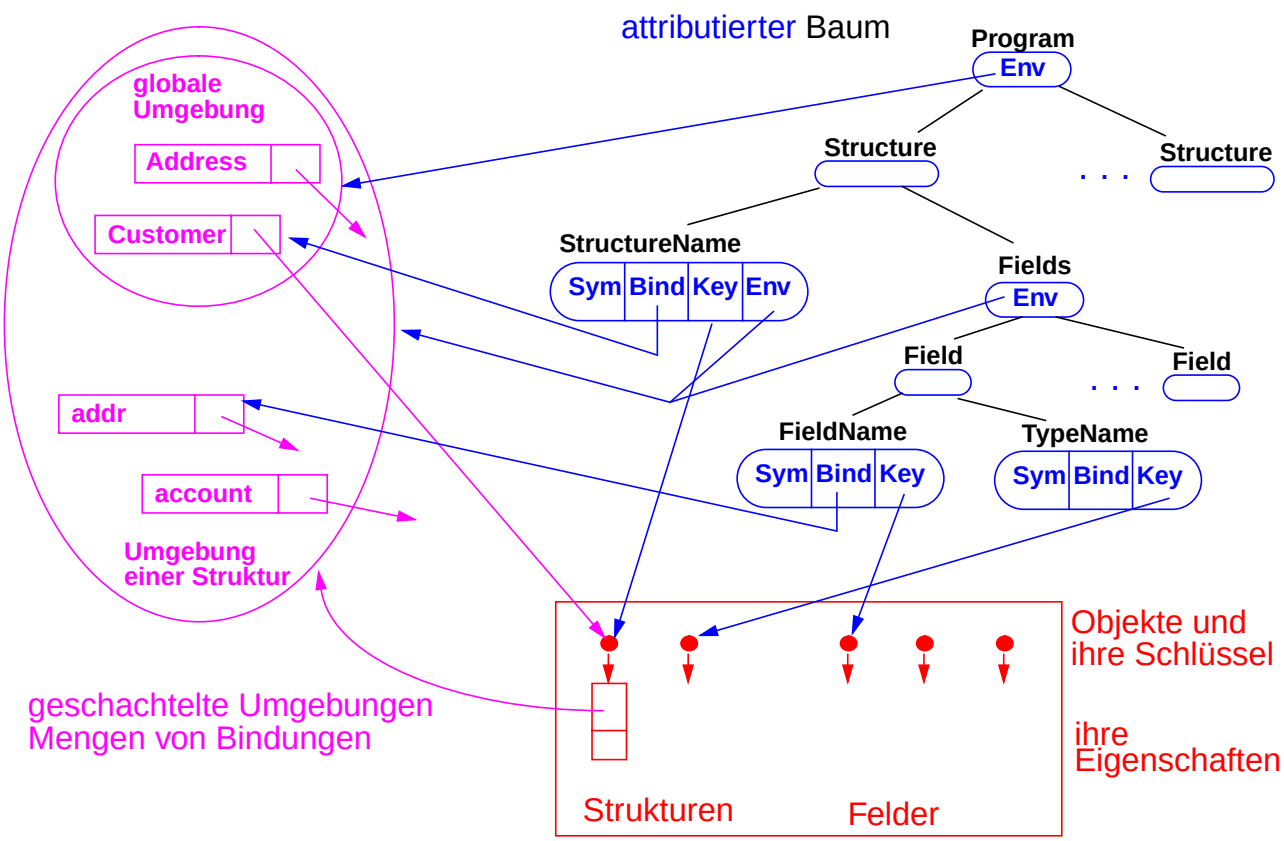
Ziele:

Übersicht zu Namen und Bindungen im Baum

in der Vorlesung:

Themen der Folie erläutern:

Attribute, Umgebungen und Schlüssel



Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 505

Ziele:

Zusammenhang Baum, Bindungen Eigenschaften

in der Vorlesung:

Themen der Folie erläutern:

Umgebungsmodul

Implementiert den abstrakten Datentyp **Environment** (Umgebung):
hierarchisch geschachtelte Mengen von **Bindungen (Name, Umgebung, Schlüssel)**

Funktionen:

NewEnv ()	bildet eine neue Umgebung e, als Wurzel einer Hierarchie; eingesetzt im **Wurzelkontext**
NewScope (e_1)	bildet eine neue Umgebung e_2 das in e_1 geschachtelt ist. Jede Bindung von e_1 ist auch eine Bindung von e_2, falls sie dort nicht verdeckt ist; eingesetzt im **Range-Kontext**
BindIdn (e, id)	bildet eine neue Bindung (id, e, k) falls e noch keine Bindung für id hat; dann ist k ein neuer Schlüssel für ein neues Objekt; das Ergebnis ist immer das Bindungstripel (id, e, k); eingesetzt für **definierende Auftreten**.
BindingInEnv (e, id)	liefert ein Bindungstripel (id, e_1, k) aus e oder einer umfassenden Umgebung von e; liefert NoBinding, falls es keine solche Bindung gibt; eingesetzt für **angewandte Auftreten**
BindingInScope (e, id)	liefert ein Bindungstripel (id, e, k) aus e, falls e es direkt enthält, sonst NoBinding; eingesetzt z. B. für **qualifizierte Namen**

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 506

Ziele:

Schnittstelle des Umgebungsmodul kennenlernen

in der Vorlesung:

Erläuterung und Anwendung der Funktionen

PDL: Generator für Definitionsmodule

zentrale Datenstruktur ordnet **Objekten Eigenschaften** zu,
z. B. T*yp einer Variablen, Elementtyp eines Array-Typs.*

Objekte werden durch **Schlüssel** *(key)* identifiziert.

Operationen:

`NewKey ( )`	liefert neuen Schlüssel
`ResetP (k, v)`	setzt zum Schlüssel k die Eigenschaft P mit Wert v
`SetP (k, v, d)`	setzt (ersetzt) zum Schlüssel k die Eigenschaft P mit Wert v (bzw. d)
`GetP (k, d)`	liefert Wert der Eigenschaft P des Schlüssels k; liefert d, falls P zu k nicht gesetzt ist

Aufrufe: in abhängigen Operationen im Baum

Implementierung: z. B. Liste von Eigenschaften zu jedem Schlüssel

aus **Spezifikationen**

Eigenschaftsname: Eigenschaftstyp;

werden Funktionen **`ResetP, SetP, GetP`** für die Eigenschaft **`P`** generiert.

Ziele:

Benutzung des Eigenschafts-Generators kennenlernen

in der Vorlesung:

Erläuterung und Anwendung der Funktionen

Beispiel: Namen und Eigenschaften für Struktur-Generator

Abstrakte Syntax

```
RULE: Descriptions  LISTOF Import | Structure          END;
RULE: Import ::= 'import' ImportNames 'from' FileName  END;
RULE: ImportNames   LISTOF ImportName                  END;
RULE: Structure ::= StructureName '(' Fields ')'       END;
RULE: Fields        LISTOF Field                       END;
RULE: Field ::=     FieldName ':' TypeName ';'         END;
RULE: StructureName ::= Ident                          END;
RULE: ImportName ::=    Ident                          END;
RULE: FieldName ::=     Ident                          END;
RULE: TypeName ::=      Ident                          END;
```

Verschiedene Nichtterminale für Bezeichner in unterschiedlichen Rollen, wenn unterschiedliche Berechnungen erwartet werden, z. B. für definierende und angewandte Auftreten.

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 508

Ziele:

Beispiel weiterverwenden

in der Vorlesung:

- Auf Beispiel GSS-1.11, GSS-5.1 verweisen,
- abstrakte Syntax vorstellen,
- Bezeichnerrollen erläutern

Umgebungsattribute berechnen

2 Symbole spielen die **Rolle eine Range**: Unterbaum begrenzt Gültigkeit darin enthaltener Definitionen

```
CLASS SYMBOL Range: Env: Environment;
SYMBOL Descriptions INHERITS Range END;
SYMBOL Fields  INHERITS Range END;
```

Wurzel der Umgebungshierarchie

```
SYMBOL Descriptions COMPUTE
   SYNT.Env = NewEnv ();
END;
```

Jedes Structure-Objekt hat eine **Umgebung als Eigenschaft**.

Darin werden die Feldnamen gebunden.

Sie wird in die globale Umgebung eingebettet

```
RULE: Structure ::= StructureName '(' Fields ')'
COMPUTE
   Fields.Env = StructureName.Env;
END;

SYMBOL StructureName COMPUTE
   SYNT.GotEnvir =
      IF (EQ (GetEnvir (THIS.Key, NoEnv), NoEnv),
         ResetEnvir
            (THIS.Key,
             NewScope (INCLUDING Range.Env)));

   SYNT.Env =
      GetEnvir (THIS.Key, NoEnv) <- SYNT.GotEnvir;
END;
```


Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 509

Ziele:

Systematik der Env-Attribute

in der Vorlesung:

Themen der Folie erläutern:

- Range-Rolle,
- geschachtelte Umgebung mit NewEnv(),
- Envir-Eigenschaft später.

Definierende Auftreten von Bezeichnern

Alle **IdentOcc**-Knoten haben diese 3 Attribute

Diese Brechnungen sind gleich für alle **IdentOcc**-Knoten

```
CLASS SYMBOL IdentOcc:
   Sym: int,
   Bind: Binding,
   Key: DefTableKey;

CLASS SYMBOL IdentOcc COMPUTE
   SYNT.Sym = TERM;
   SYNT.Key =  KeyOf (SYNT.Bind);
END;
```

Alle definierenden Auftreten **binden** ihren Namen im Env-Attribut des **nächst-umfassenden Range**

```
CLASS SYMBOL DefineAnIdent INHERITS IdentOcc
COMPUTE
   SYNT.Bind =
      BindIdn (INCLUDING Range.Env, THIS.Sym);
END;

SYMBOL StructureName INHERITS DefineAnIdent END;
SYMBOL ImportName    INHERITS DefineAnIdent END;
SYMBOL FieldName     INHERITS DefineAnIdent END;
```


Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 510

Ziele:

Berechnungen klassifizieren

in der Vorlesung:

Themen der Folie erläutern:

- CLASS-Konstrukt zur Systematisierung nutzen,
- Bindung in Umgebung herstellen (BindIdn),
- Nutzen der Rolle Range.

Angewandte Auftreten von Bezeichnern

Vorbedingung für das Suchen einer Bindung:

Alle Bindungen in den umfassenden Ranges sind hergestellt.

```
SYMBOL Descriptions COMPUTE
   SYNT.GotBindings =
      CONSTITUENTS DefineAnIdent.Bind SHIELD Range;
END;

SYMBOL Fields COMPUTE
   SYNT.GotBindings =
      CONSTITUENTS DefineAnIdent.Bind
      <- INCLUDING Range.GotBindings;
END;
```

Für den Bezeichner eine Bindung in den umfassenden Umgebungen suchen

Misserfolg melden

Nur ein angewandtes Auftreten

```
CLASS SYMBOL UseAnIdent INHERITS IdentOcc COMPUTE
   SYNT.Bind =
      BindingInEnv (INCLUDING Range.Env, THIS.Sym)
         <- INCLUDING Range.GotBindings;

   IF (EQ (SYNT.Bind, NoBinding),
      message (ERROR,
         CatStrInd ("Name is not defined: ",
                    THIS.Sym),
          0, COORDREF));
END;

SYMBOL TypeName   INHERITS UseAnIdent END;
```


Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 511

Ziele:

Abhängigkeiten verstehen

in der Vorlesung:

Themen der Folie erläutern:

- Abhängigkeiten,
- Formulierung mit CONSTITUENTS und INCLUDING,
- SHIELD-Konstrukt,
- Meldung

Eigenschaften definieren, setzen und prüfen

Beispiel: Ein Feldname darf nicht an der Stelle eines Typnamens auftreten:

```
Customer (addr: Address; account: addr;)
```

Boolsche Eigenschaft isField definiert für das Werkzeug PDL:

```
isField: int;
```

Eigenschaft auf wahr setzen für FieldName:

```
SYMBOL FieldName COMPUTE
   SYNT.GotisField =
      ResetisField (THIS.Key, 1);
END;
```

Eigenschaft prüfen im TypeName Kontext:

```
SYMBOL TypeName COMPUTE
  IF (GetisField (THIS.Key, 0),
    message (ERROR,
      CatStrInd
       ("Field name not allowed",
        THIS.Sym),0, COORDREF))
   <- INCLUDING Program.GotisField;
END;
```

Vorbedingung garantiert, dass alle ResetisField aufgerufen sind, bevor ein GetisField aufgerufen wird:

```
SYMBOL Program COMPUTE
  SYNT.GotisField =
    CONSTITUENTS FieldName.GotisField;
END;
```

Eigenschaften von Objekten
- beschreiben die **Bedeutung**
- gegen **Einschränkungen** prüfen
- zur **Transformation** verwenden

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 512

Ziele:

Eigenschaften benutzen

in der Vorlesung:

Zu Eigenschaften erläutern:

- definieren,
- setzen und lesen,
- Abhängigkeiten.

Richtlinien zu: Eigenschaften von Objekten (B)

Vorbereitung:

- Objekte werden meist durch Bezeichner in der Beschreibung benannt; durch **`DefTableKeys`** repräsentiert
- Modul der **Namensanalyse** bindet Auftreten von Bezeichnern.
- Symbolknoten der Bezeichner haben ein **`Key`**-Attribut; es identifiziert das Objekt

Entwurfsschritte für Berechnungen von Eigenschaften:

1. Bestimme **Namen und Typ der Eigenschaft** und spezifiziere sie für PDL.
2. Bestimme die **Kontexte und die Operation für das Setzen** der Eigenschaft.
3. Bestimme die **Kontexte für das Lesen** der Eigenschaft.
4. Bestimme die **Abhängigkeiten zwischen (2) und (3)**.
   In einfachen Fällen ist dies: "alle Setzen vor jedem Lesen".
5. Spezifiziere (2), (3) und das Abhängigkeitsmuster (4).

Eigenschaften des Objektes **setzen oder lesen**, möglichst im **Kontext des Symbols für den Bezeichner** platzieren
nicht die die **`Key`**-Werte im Baum weiterreichen.

Wenn möglich, **SYMBOL-Berechnungen** verwenden (siehe A).

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 513

Ziele:

PDL-Operationen im Baum systematisch anwenden

in der Vorlesung:

Erläuterung des Entwurfsmusters an Beispielen. Entwurfsschritte für folgende Aufgaben skizzieren:

- Fehler melden bei mehr als einer Definition.
- An Anwendungen Zeilennummer der Deklaration ausgeben.
- An Anwendungen Zeilennummer der vorangehenden Anwendung ausgeben.
- Fehler melden, wenn Anwendung vor Deklaration steht.
- Warnung ausgeben wenn lesender vor schreibendem Zugriff.

nachlesen:

PDL-Dokumentation

Themen für Projekte entwickeln

- spezielles, enges Anwendungsgebiet
  (aus Programmentwicklung, Web-Entwurf, Informatik-fremdes Gebiet, Hobby)
- gleichartige, strukturierte Texte in verschiedenen Varianten benötigt
- systematisch erzeugbar aus einfachen Beschreibungen

Beispiele:

- Datenstruktur-Generator
- Roboter-Steuerung
- Molekül-Darstellung
- Spielplan-Gestaltung

Vorgehen:

1. Thema formulieren
2. Beispiele für Ausgabe entwickeln
3. Beispiele für Beschreibungen (Eingabe) entwickeln
4. Regeln für Erzeugung von Ausgabe aus Beschreibungen angeben

Prüfen: **Thema geeignet?**

Vorlesung Generierung von Software aus Spezifikationen WS 2002 / Folie 514

Ziele:
Projektthemen finden

in der Vorlesung:
Randbedingungen und Vorgehen erläutern

Übungsaufgaben:
Themen suchen, beschreiben und vorstellen

Technik: Do it once

Aufgabe:

- Viele Auftreten eines Namens an dasselbe Objekt gebunden
- An genau einem Auftreten eine Berechnung ausgeführen (z. B. Zieltext für das Objekt)

Lösung:

Boolsches Attribut berechnen:
An genau einem Auftreten wahr sonst falsch.
Eigenschaft benutzen; Entwurfsschritte:

1. Eigenschaft definieren: **`Done: int;`**
2. Setzen im Namenskontext, wenn noch nicht gesetzt
3. Lesen im Namenskontext
4. keine Abhängigkeiten
5. siehe rechts

```
CLASS SYMBOL DoItOnce:
                DoIt: int;

CLASS SYMBOL DoItOnce
      INHERITS IdentOcc COMPUTE
   SYNT.DoIt =
      IF (GetDone (THIS.Key, 0),
         0,
         ORDER
          (ResetDone (THIS.Key, 1),
          1));
END;
```

Anwendung:

```
SYMBOL StructName INHERITS DoITOnce
COMPUTE
   SYNT.Text =
      IF (THIS.DoIt,
         PTGTransform (...),
         PTGNULL);
END;
```


Ziele:

Technik kennenlernen

in der Vorlesung:

Technik erläutern: