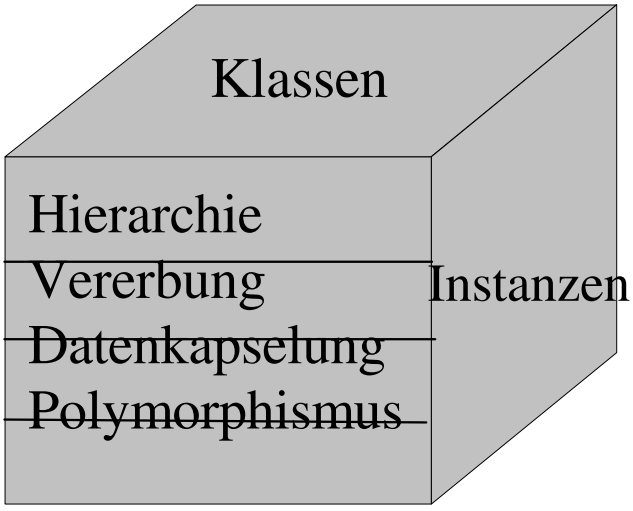
Objektorientiertes PL/I - Uebersicht

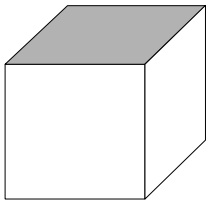
- Refresher: OO Konzepte
- OO PL/I Programmkonstrukte
- Pause
- Konzepte in OO PL/I
- OO PL/I Precompiler
- Fragen
- Pause oder Ende
- Technische Implementation

Refresher: OO Konzepte



OO PL/I Programmkonstrukte

Klassen - Deklaration



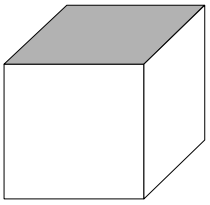
- abstrakte Einheit
- enthält Definitionen für Daten und Methoden

Methode

```
DCL 1 Car CLASS,
      2 accelerateTo PROC(BIN FIXED(31)),
      2 slowDownTo PROC(BIN FIXED(31)),
      2 break PROC,
      2 speed PROC RETURNS(BIN FIXED(31)),
      2 currentSpeed BIN FIXED(31) PRIVATE;
```

Daten

Klassen - Implementation

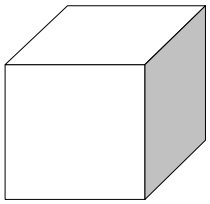


- jede Methode hat einen Implementationsteil
- Datenmember implizit/explizit ansprechbar

Klassenname Methodenname

```
Car.accelerateTo: PROC(speedKmh);
  DCL speedKmh BIN FIXED(31);


  currentSpeed = speedKmh;
END;
```

Instanzen



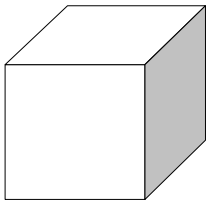
- repräsentiert ein Objekt einer Klasse
- alloziert Speicher für die Daten
- benutzt Methoden aus Klasse
- wird über Variable angesprochen

currentSpeed
currentSpeed

```
DCL aMercedes Car;
DCL aBMW      Car;

CALL aMercedes.accelerateTo(50);
CALL aBMW.break;
```

```
Car.accelerateTo: PROC;
...
END;

Car.break: PROC;
...
END;
```

Instanzen II



- statisch

```
PROG: PROC;

DCL aMercedes Car;

END;
```

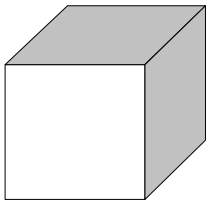
- dynamisch

```
PROG: PROC;

DCL aBMW Car DYNAMIC;

ALLOCATE Car SET(aBMW);

FREE aBMW;
```

currentSpeed

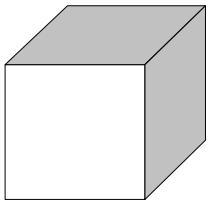
```
END;
```

Methoden / Parameterübergabe



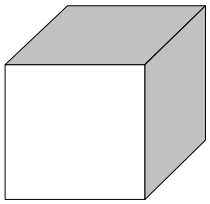
- Unterstützt sind: CHAR, BIN FIXED, DECIMAL, PTR
- Instanzvariablen sind als Parameter erlaubt
- RETURNS Attribut unterstützt, gleiche Typen + PIC
- Deklaration wie in herkömmlichen PL/I Prozeduren
- Parametertyp in CLASS und in Methodenimplementation muss übereinstimmen

Methoden / Parameterübergabe II



```
 DCL 1 Car CLASS,
       2 accelerateTo PROC(BIN FIXED(31)),
       2 speed PROC RETURNS(BIN FIXED(31)),
       ...;
Car.accelerateTo: PROC(speedKmh);
  DCL speedKmh BIN FIXED(31);
  ...
END;


Car.speed: PROC RETURNS(BIN FIXED(31));
 RETURN(currentSpeed);
END;
```

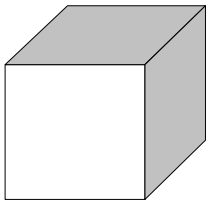
Konstruktoren / Destruktoren



- Konstrukturen sind spezielle Methoden, die bei Allozierung einer Instanz automatisch aufgerufen werden,
- Desktruktoren entsprechend bei De-allozierung
- Allozierung erfolgt je nach Instanztyp (dynamisch/statisch)
- Konstruktoren sind parametrisierbar

```
DCL 1 Car CLASS,
        2 init PROC(CHAR(30)) CONSTRUCTOR,
        2 term PROC DESTRUCTOR,
        ....;
```

Konstruktoren / Destruktoren II



➢statisch

```
XY: PROC;
 DCL myCar Car;
 ...
END;
```

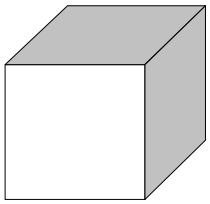
```
Car.init:
PROC;
 ...
END;
```

```
Car.term:
PROC;


END;
```

➢dynamisch

```
XY: PROC;
 DCL myCar Car DYN;
 ALLOCATE Car ...
 ...
 FREE myCar;
END;
```

Konstruktoren / Destruktoren III



- Parameter für Konstruktoren sind wie folgt zu übergeben:

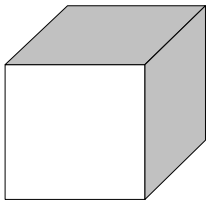
- statisch

```
DCL aMercedes Car('ZH 123456');
```

- dynamisch

```
DCL aBMW Car DYN;
ALLOCTAE Car('LU 678123') SET(aBMW);
```

Datenelemente in Klassen und Methoden



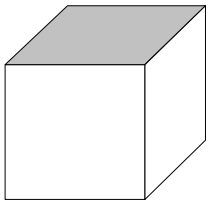
- Datenelemente sind explizit oder implizit ansprechbar
- die Variable *THIS* erlaubt expliziten Zugriff
- PRIVATE Datenelemente sind nur innerhalb Methoden ansprechbar
- es besteht keine Einschräkung bezüglich Typen

```
Car.accelerateTo: PROC(speedKmh);
  DCL speedKmh BIN FIXED(31);

  currentSpeed = speedKmh;
  THIS.currentSpeed = speedKmh;
END;
```

implizit

explizit

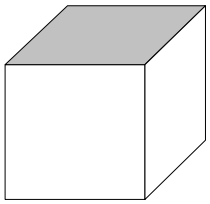
interne und externe Klassen



- intern = im "Benutzermodul" deklariert + implementiert
- extern = in einem unabhängigen Modul implementiert; Deklaration muss vorliegen

```
DCL 1 Car:CCAR CLASS,                    Entryname
    2 accelerateTo PROC(BIN FIXED(31)),
    2 slowDownTo PROC(BIN FIXED(31)),
    2 break PROC,
    2 speed PROC RETURNS(BIN FIXED(31)),
    2 currentSpeed BIN FIXED(31) PRIVATE;
```

interne und externe Klassen II



INCLUDE(CAR)

```
DCL 1 Car:CCAR CLASS,
  2 ...;
```

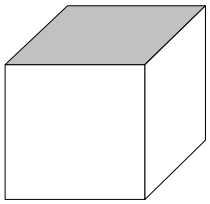
OPL(CCAR)

```
CCAR: PROC;
%INCLUDE
    INCLUDE(CAR);
Car.break: PROC;
...
END;
END;
```

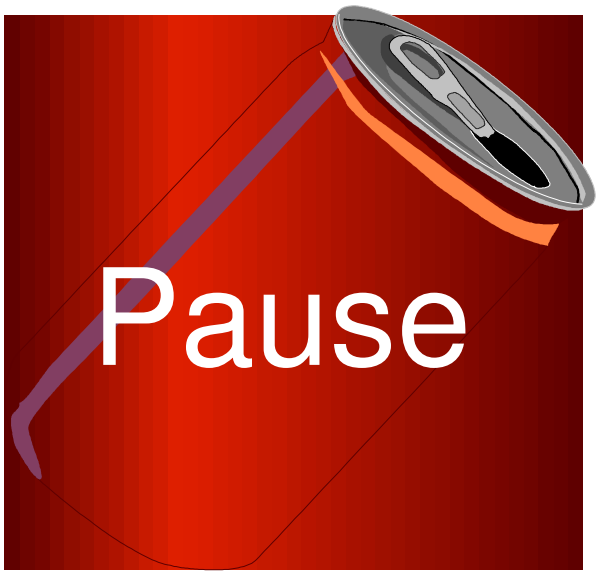
OPL(MAIN)

```
MAIN: PROC OPTIONS(MAIN);

%INCLUDE INCLUDE(CAR);

DCL aBMW Car;

CALL aBMW.break;

END;
```

interne und externe Klassen III

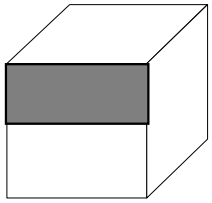


- Superklassen und abgeleitete Klassen *müssen nicht* im gleichen Module implementiert sein
- Include-Files können beliebig verschachtelt sein; der Precompiler liest jedes File nur einmal ein
- Include-Filenames müssen *qualifiziert* sein; Support für DD-Statements ist nicht implementiert


Pause

Konzepte der Objektorientierten Programmeriung in OO PL/I

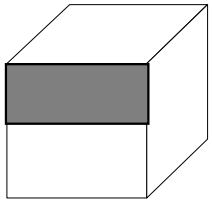
Hierarchie & Vererbung



- neue Klasse "erbt" von bestehender alle Datenelemente und Methoden (ausser Konstrukturen/Destruktoren!)
- nur abweichende Elemente werden hinzugfügt

- Methoden der Superklasse können mittels der Spezial-variable *PARENT* aufgerufen werden
- Superklasse wird bei Instanzbildung der neuen Klasse automatisch alloziert.
- nur eine Superklasse erlaubt

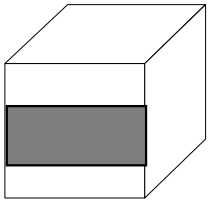
Hierarchie & VererbungII



DCL 1 SmallCar CLASS(Car), ——————— Superklasse
2 accelerateTo PROC(BIN FIXED(31));

```
SmallCar.accelerateTo: PROC(speedKmh);
 DCL speedKmh BIN FIXED(31);

 IF speedKmh <= 100 THEN
  CALL parent.accelerateTo(speedKmh);
 ELSE
   PUT SKIP LIST('Too fast!);
 END;
```

Datenkapselung



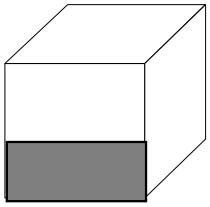
- Datenelemente sind PRIVATE (explizit)
- oder PUBLIC (implizit)
- PRIVATE Elemente dürfen nur innerhalb von Methoden
- angesprochen werden

```
DCL 1 Car CLASS,
      ...,
      2 currentSpeed BIN FIXED(31) PRIVATE;

DCL aBMW Car;
```

Fehler!

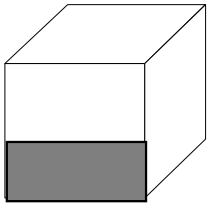

```
aBMW.currentSpeed = 150;
```

Polymorphismus



- jede Klasse kann mehrere Methoden mit gleicher Bezeichnung aufweisen, solange die Parameter unterschiedlichen Typs sind
- die Typen aller Parameter einer Methode ergeben die sog. Signatur einer Methode
- die Aufrufparameter bestimmen, welche Implementationsvariante einer Methode aufgerufen wird
- jede Methode (inkl. Konstrukturen, ohne Destruktoren) qualifiziert für Polymorphismus in diesem Sinne

Polymorphismus II



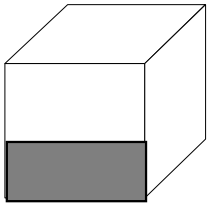
```
DCL 1 Car CLASS,
        2 accelerateTo PROC(BIN FIXED(31)),
        2 accelerateTo PROC(CHAR(10)),
        ...;
```

```
Car.accelerateTo: PROC(speedKmh);
 DCL speedKmh BIN FIXED(31);
 currentSpeed = speedKmh;
END;
```

```
Car.accelerateTo: PROC(speedKmh);
 DCL speedKmh CHAR(10);
 DCL picSpeed PIC'99999';
 ...
END;
```

```
DCL aBMW Car;

CALL aBMW.accelerateTo(20);

CALL aBMW.accelerateTo("20");
```

Polymorphismus III



- die Signatur einer Methode setzt sich aus den sog. Basistypen von OO PL/1 zusammen:

PL/1 Attribut	**Basistyp**
BIN FIXED(x)	NUMBER
CHAR(x)	STRING
PIC'x'	NUMBER
PTR	PLIPTR
DECIMAL(x,y)	NUMBER

OO PL/I Precompiler

Der OO PL/I Precompiler ...

- ist eigentlich ein Parser, kein "Compiler"
- versucht, OO PL/I in "echtes PL/I" umzuwandeln
- erkennt keine PL/I Fehler, sondern nur Syntaxfehler innerhalb OO PL/I Statements!
- erstellt eine Fehler/Warnungsliste

- Syntax

TSO OOPLI Input-DSN Output-DSN

- Input-DSN / Output-DSN sind qualifizierte DSN
- (inklusive Member-Name!)
- Output-DSN wird ohne Warnung gelöscht!

Fragen?