Vertragsmodell

- **Konzept des Vertragsmodells**
- **Zusicherungen**
- **Realisierung von Zusicherungen mit Pragmas**
- **Exkurs Ausnahmebehandlung**
- **Zusicherungen mittels Ausnahmebehandlung**

Erinnerung

```
INTERFACE Ordner;
PROCEDURE LegeTextAb (t : TEXT);
PROCEDURE EntnehmeText () : TEXT ;
PRODEDURE IstVoll () : BOOLEAN ;
PROCEDURE IstLeer () : BOOLEAN ;
PROCEDURE Beschrifte (t : TEXT);
PROCEDURE GibBeschriftung () : TEXT;
PROCEDURE Initialisiere ();
```

- **Feststellung:**
  - `LegeTextAb` und `Entnehme` sind ***nicht in jedem Zustand*** des Ordners sinnvoll:
    - ein voller Ordner kann keine weiteren Texte aufnehmen; ein leerer keine herausgeben.
  - Solche Operationen sind nur in ***bestimmten Situationen*** (abhängig von bestimmten Bedingungen) sinnvoll.
  - Um den sicheren Umgang mit einem solchen Objekt zu gewährleisten, stellen wir an der Schnittstelle entsprechende ***Testfunktionen*** (Sicherheitsabfragen) wie `IstLeer` oder `IstVoll` zur Verfügung.

Konzept des Vertragsmodells

Einsatz der Testfunktionen

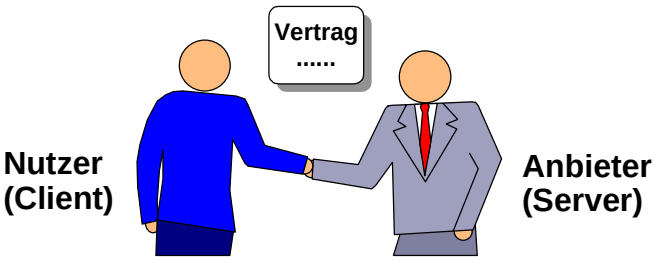
```
Ordner.Initialisiere;

IF Ordner.IstVoll() THEN
  SIO.PutLine ("Ordner ist bereits voll");
ELSE
  Ordner.LegeTextAb ("Nicht immer sind bequeme Stuehle ...");
END;

IF Ordner.IstLeer() THEN
  SIO.PutLine ("Ordner ist leer");
ELSE
  t := Ordner.EntnehmeText();
END;
```

Prüfen, ob die Operation angewendet werden darf.

- **Frage:**
  - Wer soll ***sicherstellen***, dass eine Operation ausgeführt werden darf?
  - ***Nutzer*** oder ***Anbieter***? oder
  - zur Sicherheit auf ***beiden Seiten***?

Konzept des Vertragsmodells

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.
- **Frage**
  - Wie kann ein solcher Vertrag ***programmtechnisch*** realisiert werden?

Zusicherungen

Erstes Beispiel

```
INTERFACE OrdnerADT;

TYPE Ordner <: REFANY;

PROCEDURE LegeTextAb (VAR o: Ordner; t : TEXT);
  require NOT IstVoll(o)
  ensure  NOT IstLeer(o)

PROCEDURE EntnehmeText (VAR o: Ordner; ) : TEXT ;
  require NOT IstLeer(o)
  ensure  NOT IstVoll(o)

PROCEDURE IstVoll ( o: Ordner; ) : BOOLEAN ;
PROCEDURE IstLeer ( o: Ordner; ) : BOOLEAN ;
. . .

END OrdnerADT.
```

Vorbedingung

Nachbedingung

Zusicherungen

Arten und Nutzung

- **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 formal definieren
  - verbessern die ***Testbarkeit***
  - verbessern die ***Fehlersuche*** (debugging)
  - verlangen vom Entwickler ***abstraktes*** Denken

Zusicherungen

Vor- und Nachbedingungen

- **Vorbedingung**
  - beschreibt eine Bedingung, die der ***Nutzer*** (Aufrufer) einer Operation ***einhalten*** muß, damit die Operation korrekt arbeitet
- **Nachbedingung**
  - beschreibt einen Zustand, der nach dem erfolgreichen Ausführen der Operation vorhanden ist
  - diese garantiert der ***Anbieter***

Die Technik der Zusicherungen sollte insb. bei der Entwicklung von Programmkomponenten (Modulen) eingesetzt werden, die wiederver-wendet werden sollen!

Zusicherungen

Verpflichtung <-> Nutzen

- **Vorbedingungen sind**
  - Verpflichtungen für den Benutzer
  - Nutzen für den Anbieter
- **Nachbedingungen sind**
  - Nutzen für den Benutzer
  - Verpflichtungen für den Anbieter

Operation `EntnehmeText`

	Verpflichtungen	**Nutzen**
Nutzer	Der Ordner darf nicht leer sein.	Der zuletzt eingegebene Text wird entnommen und zurückgegeben.
Anbieter	Verändere den Ordner so, daß der zuletzt eingegebene Text entnommen wird.	Ist sicher, daß es noch einen Text im Ordner gibt

Zusicherungen

Invariante

- **Invariante beschreibt Bedingungen,**
  - die erfüllt sind, wenn Objekte eines ADTs ***erzeugt*** werden
  - die während der ***gesamten*** Lebenszeit der Objekte gelten
  - d.h. von allen Operationen ***nicht verletzt*** werden

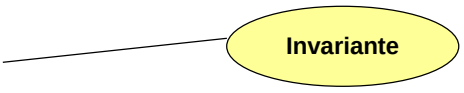
- **Beispiel: ADT Ordner**
  - zu jedem Zeitpunkt im "Leben" eines Ordnerobjekts gilt:
  - **`anzahlTexte >= 0 AND anzahlTexte <= MaxTexte`**
  - ADT Ordner wird um eine interne Funktion erweitert, die die Invariante prüft.

```
PROCEDURE Invariante (o : Ordner): BOOLEAN =
BEGIN
  RETURN (o^.anzahlTexte >= 0 AND o^.anzahlTexte <= MaxTexte);
END Invariante;
```


Zusicherungen

ADT Ordner mit Invariante

```
INTERFACE OrdnerADT;

TYPE Ordner <: REFANY;

PROCEDURE LegeTextAb (VAR o: Ordner; t : TEXT);
  require NOT IstVoll(o)
  ensure  NOT IstLeer(o)
  ensure  Invariante(o)

PROCEDURE EntnehmeText (VAR o: Ordner; ) : TEXT ;
  require NOT IstLeer(o)
  ensure  NOT IstVoll(o)
  ensure  Invariante(o)

PROCEDURE IstVoll ( o: Ordner; ) : BOOLEAN ;
PROCEDURE IstLeer ( o: Ordner; ) : BOOLEAN ;
. . .
END OrdnerADT.
```



Realisierung mit Pragmas

Pragmas in M3

- **Pragmas in Modula-3**
  - Pragmas sind Anweisungen an den ***Übersetzer***
  - Sie ändern die ***Semantik*** des Programmes nicht
  - Die Implementierung eines Pragmas ist ***übersetzerspezifisch***
  - Pragmas können ***irgendwo*** im Programmtext auftreten
  - Pragmas müssen einer vordefinierte Syntax entsprechen
    - *<* Pragmaname Parameter *>*
- **Das Pragma ASSERT**
  - `<* ASSERT AUSDRUCK *>`
  - Der AUSDRUCK muß einen Wert vom Typ BOOLEAN liefern
  - Der Ausdruck wird zur ***Laufzeit*** ausgewertet
  - Ist das Ergebnis des Ausdrucks FALSE
    - wird ein ***Laufzeitfehler*** ausgelöst
    - und das Programm bricht ab!

Realisierung mit Pragmas

Zusicherungen mit Pragma ASSERT

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

  t := o^.ordnerInhalt[o^.anzahlTexte];
  o^.ordnerInhalt[o^.anzahlTexte] := "";
  o^.anzahlTexte := o^.anzahlTexte - 1;

 <* ASSERT NOT IstVoll(o)  *>
 <* ASSERT Invariante(o)   *>

  RETURN t;
END EntnehmeText;
```

```
VAR ordner1, ordner2 : OrdnerADT.Ordner; t : TEXT;

BEGIN
  ordner1 := OrdnerADT.Anlegen();
  OrdnerADT.Beschrifte (ordner1, "Kleine Gedichte");
  OrdnerADT.LegeTextAb (ordner1, "Nicht immer...");
  t := OrdnerADT.EntnehmeText(ordner1);
  t := OrdnerADT.EntnehmeText(ordner1);
```

Laufzeitfehler

Realisierung mit Pragmas

Diskussion: Einsatz von ASSERT

- **ASSERT**
  - bietet eine einfache Möglichkeit, Zusicherungen zu implementieren.
- **Nachteile**
  - "brutale" Implementierung: Programmabbruch
  - es wird keine Information über die Verletzung des Vertrages geliefert
  - es gibt keine Möglichkeit, auf die Verletzung des Vertrages zu reagieren
- **Bessere Lösung**
  - Entwerfen eines Moduls zur Prüfung von Zusicherungen

```
PROCEDURE Require (expr : BOOLEAN);
PROCEDURE Ensure  (expr : BOOLEAN);
```

- **Frage**
  - Was soll geschehen, wenn eine Zusicherung verletzt wird?
  - Verletzung: Ausnahmesituation!

Exkurs Ausnahme-behandlung

Ausnahmen: Bsp. und Def.

- **Beispiel für Ausnahmesituationen**
  - Während des Schreibens einer Datei auf Diskette wird die Diskette entfernt.
  - Kein Plattenplatz mehr verfügbar.
  - Berechnung eines Addition ist größer als LAST(INTEGER).
  - Falsche Daten werden von einer Datei eingelesen.
- **Definition: Ausnahme**
  - IEEE Glossary: "An event that causes suspension of normal program execution. Types include addressing exception, data exception, operation exception, overflow exception, protection exception, underflow exception."
  - Ausnahmen sind Programmzustände, die ***nicht im normalen*** Programmablauf vorgesehen sind.

Exkurs Ausnahme-behandlung

Merkmale von Ausnahmen

- **Merkmale**
  - Ausnahmen entstehen zur ***Laufzeit.***
  - Einige Programmiersprachen (Java, Ada, Modula-3) erlauben, benutzerdefinierte ***Ausnahmen*** zu deklarieren und ***Ausnahmebehandlung*** durchzuführen.
  - Beispiel: vorgegebene Ausnahme
    - *SIO.Error*
      Wird von den IO-Modulen erweckt, wenn Datei-Operationen nicht wie intendiert durchgeführt werden können.
- **Deklaration benutzerdefinierter Ausnahmen**
  - EXCEPTION <name>;
  - Wird eine Ausnahme von einer Schnittstelle exportiert, können auch die Klienten diese Ausnahme generieren.
- **Erwecken einer Ausnahme**
  - RAISE-Anweisung

Exkurs Ausnahme-behandlung

Ausnahme und Ausnahmebehandlung

```
MODULE Ausnahme EXPORTS Main;
IMPORT SIO;
VAR eingabe : INTEGER;
BEGIN
  LOOP
    TRY
      SIO.PutLine ("Geben Sie bitte eine ganze Zahl ein:");
      eingabe := SIO.GetInt();
      EXIT;
    EXCEPT
      SIO.Error => SIO.PutLine ("*** Eingabeformat falsch");
                   line := SIO.GetLine();
    END;
  END;
  . . .
END Ausnahme.
```

Schützt Anweisungen, bzgl. dem Auftreten von Ausnahmen

Ausnahmebehandlung "exception handler"

Exkurs Ausnahme-behandlung

Weiterleiten von Ausnahmen

- **Idee:**
  - Wenn in einer Prozedur eine Ausnahme nicht behandelt werden soll, dann kann diese Prozedur die Ausnahme an die ***sie rufende Prozedur*** weiterleiten.
  - So können Ausnahmen über ***mehrere Stufen*** weitergeleitet und an der entsprechenden Stelle behandelt werden.
- **Weiterleitung von Ausnahmen**
  - muss bei der Prozedur-Deklaration angegeben werden
  - PROCEDURE <name> <signature> RAISES {exc1, .. excN}
- **Beispiele:**
  - viele Operationen des Moduls `SIO` leiten die Ausnahme `Error` weiter
  - `PROCEDURE GetChar(rd: Reader := NIL): CHAR RAISES {Error};`

Exkurs Ausnahme-behandlung

Kontrollfluß bei Ausnahmen

- Eine Ausnahme tritt in TRY-EXCEPT-Anweisung auf und die Ausnahme wird dort behandelt, dann
  - werden die in dem Ausnahmebehandler für die Ausnahme stehenden Anweisungen durchgeführt,
  - Das Programm wird anschließend nach dem END der TRY-EXCEPT-Anweisung ***fortgeführt***.
- Eine Ausnahme tritt in einem ***ungeschützten*** Bereich einer Prozedur auf und die Ausnahme ist Element der RAISES-Liste der Prozedur, dann,
  - wird die Prozedur abgebrochen und die Ausnahme an die die Prozedur rufende Prozedur ***weitergeleitet***
- Kann eine Ausnahme weder behandelt noch weitergeleitet werden, dann,
  - wird das Programm mit einem Laufzeitfehler ***abgebrochen***

Zusicherungen mittels Ausnahmebeh.

Schnittstelle des Moduls `Assertion`

```
INTERFACE Assertion;

EXCEPTION Violated;
          Terminate;

PROCEDURE Require (expr : BOOLEAN; procName: TEXT)
                   RAISES {Violated};

PROCEDURE Ensure  (expr : BOOLEAN; procName: TEXT)
                   RAISES {Violated};

PROCEDURE EnableAssertions();
PROCEDURE DisableAssertions();

END Assertion.
```

Ausnahme `Violated` wird generiert, wenn eine Zusicherung verletzt wird

Prüfung der Zusicherungen kann unterdrückt werden

Zusicherungen mittels Ausnahmebeh.

Implementierung `Assertion` - 1

```
MODULE Assertion;
IMPORT SIO;

VAR enabled : BOOLEAN;

PROCEDURE Require (expr : BOOLEAN; procName: TEXT)
                   RAISES {Violated} =
BEGIN
  IF enabled AND NOT expr THEN
    SIO.PutLine("*** Precondition violated: " & procName);
    RAISE Violated;
  END;
END Require;

PROCEDURE Ensure  (expr : BOOLEAN; procName: TEXT)
                   RAISES {Violated} =
BEGIN
  IF enabled AND NOT expr THEN
    SIO.PutLine("*** Postcondition violated: " & procName);
    RAISE Violated;
  END;
END Ensure;
```

Geschützte Variable

Zusicherungen mittels Ausnahmebeh.

Implementierung Assertion - 2

```
PROCEDURE EnableAssertions()=
BEGIN
  enabled := TRUE;
END EnableAssertions;

PROCEDURE DisableAssertions()=
BEGIN
  enabled := FALSE;
END DisableAssertions;


BEGIN
  EnableAssertions();
END Assertion.
```


Zusicherungen mittels Ausnahmebeh.

Ordner-Operationen mit Assertion - 1

```
INTERFACE OrdnerADT;

IMPORT Assertion AS As;

TYPE Ordner <: REFANY;

PROCEDURE LegeTextAb     (VAR o: Ordner; t : TEXT)
                          RAISES {As.Violated};
PROCEDURE EntnehmeText   (VAR o: Ordner): TEXT
                          RAISES {As.Violated};
PROCEDURE IstVoll        (o: Ordner): BOOLEAN;
PROCEDURE IstLeer        (o: Ordner): BOOLEAN;
PROCEDURE Beschrifte     (VAR o: Ordner; t : TEXT);
PROCEDURE GibBeschriftung(o: Ordner): TEXT;
PROCEDURE Anlegen        ( ) : Ordner RAISES {As. Violated};

END OrdnerADT.
```

Zusicherungen mittels Ausnahmebeh.

Ordner-Operationen mit Assertion - 2

```
PROCEDURE LegeTextAb (VAR o: Ordner; t : TEXT)
                     RAISES {As. Violated} =
BEGIN
  As.Require (NOT IstVoll(o), "OrdnerADT.LegeTextAb");
  o^.anzahlTexte := o^.anzahlTexte + 1;
  o^.ordnerInhalt[o^.anzahlTexte] := t;
  As.Ensure (NOT IstLeer(o), "OrdnerADT.LegeTextAb");
  As.Ensure (Invariante(o), "OrdnerADT.LegeTextAb");
END LegeTextAb;

PROCEDURE EntnehmeText (VAR o: Ordner) : TEXT
                       RAISES {As. Violated} =
VAR t : TEXT;
BEGIN
  As.Require (NOT IstLeer(o), "OrdnerADT.EntnehmeText");
  t := o^.ordnerInhalt[o^.anzahlTexte];
  o^.ordnerInhalt[o^.anzahlTexte] := "";
  o^.anzahlTexte := o^.anzahlTexte - 1;
  As.Ensure (NOT IstVoll(o), "OrdnerADT.EntnehmeText");
  As.Ensure (Invariante(o), "OrdnerADT.EntnehmeText");
  RETURN t;
END EntnehmeText;
```


Zusicherungen mittels Ausnahmebeh.

Umgang mit den Ausnahmen - 1

- **Empfehlung**
  - Der aufrufende Block klammert alle Anweisungen in einer TRY-EXCEPT-Anweisung.
  - Im EXCEPT-Teil werden noch mögliche Abschluß-Operationen durchgeführt (z.B. Schließen von Dateien) und eine entsprechende Meldung ausgegeben und die Ausnahme `Terminate` generiert.

```
PROCEDURE ArbeiteMitOrdner() RAISES {As.Terminate} =
VAR ordner1 : OrdnerADT.Ordner;
BEGIN
  TRY
    ordner1 := OrdnerADT.Anlegen();
    OrdnerADT.Beschrifte (ordner1, "Kleine Gedichte");
    SIO.PutLine (OrdnerADT.EntnehmeText(ordner1));
    ...
  EXCEPT
    As.Violated =>
      SIO.PutLine ("*** Called from: ArbeiteMitOrdner");
      RAISE As.Terminate;
  END;
END ArbeiteMitOrdner.
```

Zusicherungen mittels Ausnahmebeh.

Umgang mit den Ausnahmen - 2

```
MODULE Ordner_Test EXPORTS Main;

IMPORT Assertion AS As;
IMPORT OrdnerADT,  SIO;

PROCEDURE ArbeiteMitOrdner()RAISES {As.Terminate}=
BEGIN
  ...
END ArbeiteMitOrdner;

BEGIN
  As.EnableAssertions();
  TRY
    ArbeiteMitOrdner();
  EXCEPT
    As.Terminate =>
      SIO.PutLine ("*** Program terminated ");
  END
END Ordner_Test.
```

```
*** Precondition violated: OrdnerADT.EntnehmeText
*** Called from: ArbeiteMitOrdner
*** Program terminated
```


Zusicherungen mittels Ausnahmebeh.

Arbeiten mit Zusicherungen - 1

- **Im Rumpf einer Operation darf die Vorbedingung nicht geprüft werden**
  - Widerspricht dem herkömmlichen ***defensiven*** Programmieren
- **Vorteil**
  - Schon mittelgroße Systeme enthalten ca. 10 -20% Code, um solche Eingangsprüfungen durchzuführen.
  - Dabei entstehen komplexe Prüfungen.
  - Prüfroutinen sind häufig Ursachen für Fehler.

```
PROCEDURE Wurzel (x: REAL): REAL is
  require X >= 0
BEGIN
END Wurzel ;

PROCEDURE Wurzel (x: REAL): REAL is
  require X >= 0
BEGIN
  if x < 0 then
    “handle error”
  else
    RETURN (x * x);
  end
END Wurzel ;
```

Zusicherungen mittels Ausnahmebeh.

Arbeiten mit Zusicherungen - 2

- **Zusicherungen sollen nicht verwendet werden, um Spezialfälle zu behandeln**
  - Zusicherungen sind Aussagen über die ***korrekte*** Nutzung.
  - Sollen Spezialfälle behandelt werden, dann werden herkömmliche ***Kontrollanweisungen*** verwendet (IF oder CASE).
- **Beispiel**
  - Wenn `Wurzel` den Fall x < 0 als Spezialfall behandeln soll, dann ist das zu programmieren.
  - Wenn x >= 0 die Vorbedingung ist, dann ist ein Aufruf

    ```
    Wurzel (-1);
    ```

    nicht erlaubt, also eine ***nicht vertragskonforme Benutzung***!
- **Wird eine Zusicherung verletzt (zur Laufzeit)**
  - Vorbedingung: Nutzer hat den Vertrag verletzt
  - Nachbedingung: Anbieter hat den Vertrag verletzt

Was haben wir gelernt?

- **Vertragsmodell: Vertrag zwischen Nutzer (Client) und Anbieter (Server),**
- **Konsistenz durch Vor- und Nachbedingungen sowie Invarianten**
- **Pragmas, Pragma `Assert`**
- **Ausnahmen: Motivation, Ausnahmedeklaration, Erwecken einer Ausnahme, Behandlung durch Ausnahmebehandler, Weiterreichen einer Ausnahme**
- **Zusicherungen mit Ausnahmen, allgemeingültiges Ausnahmebehandlungsmodul `Assertion`, Verwendung bei Clienten und bei Server**

Glossar

- **Vertragsmodell, Vorbedingungen und Benutzerverpflichtung, Nachbedingungen und Anbieterverpflichtung**
- **Formalisierung von Zusicherungen, Vorbedingungen, Nachbedingungen, Invarianten**
- **ADT-Schnittstelle mit Zusicherung**
- **Pragmas in Modula-3, vordefiniertes Pragma `Assert`, Nutzung für Zusicherungsrealisierung**
- **Ausnahmesituationen, Definition Ausnahme, (vordefinierte und) benutzerdefinierte Ausnahme, Ausnahmedeklaration, Erwecken einer Ausnahme, Ausnahmebehandlung und TRY-EXCEPT-Anweisung**
- **Ausnahmebehandlung im Ausnahmebehandler, ungeschützte Ausnahmen und weiterreichen, falls gerufene Prozedur diese Ausnahme im Kopf angibt, Programmabbruch, falls weder behandelt noch weitergeleitet**
- **Ausnahmen für die Realisierung von Zusicherungen**