Automatisierung von Java Anwendungen (3)

Object Rexx, 3

Ausnahmen (Exceptions), Referenzen, Direktiven (::routine, ::requires)

Prof. Dr. Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Augasse 2-6 ■ A-1090 Wien

Classic Rexx
Abarbeitung von Programmen

- Datei mit Programm wird geladen
- 1. Zeile, 1. Spalte, Beginn der Zeichenkette **/***
- Anschließend Zeile für Zeile:
  - Einlesen einer Anweisung
  - Syntaktische Überprüfung
  - Ausführung (Interpretation) der Anweisung
- Zeilen, die nicht aufgesucht wurden, bleiben unüberprüft!
  - Z.B. In **IF**-Anweisungen der **THEN**- oder **ELSE**-Teil
  - Potentielle Zeitbombe:
    - *Irgendwann (auch erst nach Jahren!) kann eine Anweisung ausgeführt werden, die syntaktisch falsch ist und daher zu einem Programmabbruch führen kann*

Object Rexx, Regina
Abarbeitung von Programmen

- Datei mit Programm wird geladen
- ***Sämtliche Zeilen*** werden eingelesen
  - *Sämtliche Anweisungen* werden syntaktisch überprüft und in einen komprimierten Zwischencode (englisch: "tokenized image") übersetzt, der später ausgeführt wird

    → ***Keine Zeitbomben!***
- **::REQUIRES**-Direktiven werden ausgeführt
- Restliche Direktiven (**::ROUTINE**, **::CLASS**, **::METHOD**) werden befolgt
- Programm startet mit der allerersten Anweisung vor der allerersten Direktive
  - Kann daher auch für die Initialisierung von Modulen benutzt werden

Ausnahmebedingungen (Exceptions)

- Kategorien (Bedingungen, Conditions)
  - **SYNTAX** Falscher Aufbau einer Anweisung
  - **FAILURE** Fehler in externem Aufruf
  - **ERROR** Fehler in externem Aufruf, wenn nicht mit "FAILURE" oder "ANY" abgefangen
  - **HALT** Strg-C (Strg-Break) gedrückt
  - **NOVALUE** Nichtinitialisierte Variable verwendet
  - **USER** Benutzerdefinierte Ausnahmen
  - **LOSTDIGITS** Mehr Ziffern als **NUMERIC DIGITS**
  - **NOMETHOD**, **NOSTRING**, **NOTREADY** (später ...)
  - **ANY** Fängt ***alle*** Ausnahmen ab

Ausnahmebedingungen (Exceptions)

- Aufruf der vorgesehenen (programmierten) Ausnahmebehandlung mit
  - **CALL** als Unterprogramm
  - **SIGNAL** als Sprung ohne Wiederkehr
- Abfangen von Ausnahmen aktivieren mit **ON**, deaktivieren mit **OFF**
- Angabe der Kategorie, bei **USER** die benutzerdefinierte Unterkategorie
- Optionale Sprungmarke nach **NAME**
  - wenn keine Sprungmarke angegeben ist, wird eine Sprungmarke gesucht, die denselben Namen wie die Ausnahmebedingung (Kategorie) trägt

Ausnahmebedingungen (Exceptions)

- **Hinweis:** Windows-Entwicklungsumgebung
  - Abfangen der Ausnahmebedingung **ANY** ist nicht möglich!
  - Entwicklungsumgebung fängt selbst sämtliche nicht abgefangenen Ausnahmebedingungen mit **ANY** ab
  - Analysiert und erzeugt Fehlermeldungen anstatt des Interpreters
- Alle Ausnahmebedingungen können nur im Geltungsbereich des aufrufenden Programms abgefangen werden
  - Damit wird das Auslösen einer Ausnahmebedingung mit der **RAISE**-Anweisung erst im Geltungsbereich des aufrufenden Programms abfangbar
  - Ausnahme: **SYNTAX**

Allgemeine Ausnahmebehandlung

- Allgemeine Ausnahmebehandlung
  - am Ende des Geltungsbereiches
  - Aktivieren durch "SIGNAL ON"-Anweisung

```
SIGNAL ON ANY   /* no label, hence "ANY" */
  ... Rexx-code ...
ANY:  /* target for any exception     */
  exc_rc   = RC   /* save return code */
  exc_sigl = SIGL /* save line number */
  exc_type = CONDITION("C") /* get exception type */
  CALL say2stderr "REXX 'RC':" exc_rc
  CALL say2stderr "     type:" exc_type
  CALL say2stderr
  CALL say2stderr "  in line:" exc_sigl
  CALL say2stderr "          " SOURCELINE(exc_sigl)
  EXIT -1             /* indicate error   */
SAY2STDERR:           /* write to STDERR: */
  CALL LINEOUT "STDERR:", ARG(1)
  RETURN
```

Ausnahmen, Beispiel 1

```
/* */
SIGNAL ON SYNTAX NAME ANY /* target name "ANY" given */
SAY Nix   /* Variable not initialized! */
EXIT 0
ANY:  /* target for any exception     */
  exc_rc   = RC   /* save return code */
  exc_sigl = SIGL /* save line number */
  exc_type = CONDITION("C") /* get exception type */
  CALL say2stderr "REXX 'RC':" exc_rc
  CALL say2stderr "     type:" exc_type
  CALL say2stderr
  CALL say2stderr "  in line:" exc_sigl
  CALL say2stderr "          " SOURCELINE(exc_sigl)
  EXIT -1             /* indicate error   */
SAY2STDERR:           /* write to STDERR: */
  CALL LINEOUT "STDERR:", ARG(1)
  RETURN
```

Ausgabe:

```
NIX
```

Ausnahmen, Beispiel 2

```
/* */
SIGNAL ON NOVALUE NAME ANY
SAY Nix   /* Variable not initialized! */
EXIT 0
ANY:  /* target for any exception     */
  exc_rc   = RC   /* save return code */
  exc_sigl = SIGL /* save line number */
  exc_type = CONDITION("C") /* get exception type */
  CALL say2stderr "REXX 'RC':" exc_rc
  CALL say2stderr "     type:" exc_type
  CALL say2stderr
  CALL say2stderr "  in line:" exc_sigl
  CALL say2stderr "          " SOURCELINE(exc_sigl)
  EXIT -1             /* indicate error   */
SAY2STDERR:           /* write to STDERR: */
  CALL LINEOUT "STDERR:", ARG(1)
  RETURN
```

Ausgabe:

```
REXX 'RC': RC
     type: NOVALUE
  in line: 3
           SAY Nix
```

Aufwerfen von Ausnahmen

- Rexx-Interpreter bei Fehlern
- **RAISE**-Anweisung
  - **RAISE** Kategorie
    - Erzeugt ("wirft auf") die angegebene Ausnahmebedingung
  - **RAISE PROPAGATE**
    - Kann nur **während** einer Ausnahmebehandlung angegeben werden
    - Erzeugt dieselbe Ausnahmebedingung noch einmal im aufrufenden Programmteil
    - Leitet die Ausnahmebedingung an die übergeordnete Ebene weiter, sodaß sie auch dort entweder abgefangen wird oder zum Programmabbruch durch den Interpreter führt

Aufwerfen von Ausnahmen, Beispiel 1

```
/**/
SAY "hallo"
RAISE SYNTAX 9.1 /* Pretend syntax error # 9.1 */
EXIT 0
```

Ausgabe:

```
hallo
     3 *-* RAISE SYNTAX 9.1 /* Pretend syntax error # 9.1 */
Error 9 running C:\TEMP\wi-pub\lv\poolv\code\script5.rex line 3:
   Unexpected WHEN or OTHERWISE
Error 9.1:  WHEN has no corresponding SELECT
```

Aufwerfen von Ausnahmen, Beispiel 2

```
/**/
SIGNAL ON SYNTAX /* no label, hence "SYNTAX"   */
SAY "hallo"
RAISE SYNTAX 9.1 /* Pretend syntax error # 9.1 */
EXIT 0

SYNTAX:          /* target for any exception   */
  SAY "In SYNTAX-exception handling code."
  EXIT -1
```

Ausgabe:

```
hallo
In SYNTAX-exception handling code.
```

Aufwerfen von Ausnahmen, Beispiel 3

```
    /**/
    SIGNAL ON ANY    /* no label, hence "ANY"      */
    SAY "hallo"
    RAISE SYNTAX 9.1 /* Pretend syntax error # 9.1 */
    EXIT 0
    ANY:  /* target for any exception     */
      exc_rc   = RC   /* save return code */
      exc_sigl = SIGL /* save line number */
      exc_type = CONDITION("C") /* get exception type */
      CALL say2stderr "REXX 'RC':" exc_rc
      CALL say2stderr "     type:" exc_type
      CALL say2stderr
      CALL say2stderr "  in line:" exc_sigl
      CALL say2stderr "          " SOURCELINE(exc_sigl)
      EXIT -1             /* indicate error   */
    SAY2STDERR:           /* write to STDERR: */
      CALL LINEOUT "STDERR:", ARG(1)
      RETURN
```

Ausgabe:

```
hallo
REXX 'RC': 9
     type: SYNTAX
  in line: 4
           RAISE SYNTAX 9.1 /* Pretend syntax error # 9.1 */
```

Variable (Rexx)

- Zeichenkettenvariable
- Stem-Variable, die in ihren Indizes Zeichenketten gespeichert haben
- Argumente für Unterprogramme/Funktionen
  - **Nur** Zeichenkettenvariablen erlaubt
  - Keine Stem-Variable !
    - EXPOSE-Anweisung erlaubt den Zugriff auf die Stem-Variable im aufrufenden Programmteil

Variable (Object Rexx)

- Variable sind **Referenzen** auf Instanzen von Object Rexx-Klassen
  - Zeichenketten
  - Stems
  - … (später …)
- Argumente für Unterprogramme/Funktionen
  - **PARSE ARG**-Anweisung
    - ***Nur*** Zeichenkettenvariablen erlaubt
    - Keine Stem-Variable !
    - **EXPOSE**-Anweisung erlaubt den Zugriff auf die Stem-Variable im aufrufenden Programmteil
  - **USE ARG**-Anweisung
    - ***Sämtliche*** Objekte als Argumente erlaubt

Routinen (Object Rexx)

- Routinen sind Direktiven
  - Beginnen daher mit einem doppelten Doppelpunkt (**::**)
  - Werden bei der ersten syntaktischen Überprüfung im Geltungsbereich
    - des Programmes zugänglich gemacht, und
    - zusätzlich in allen übergeordneten (aufrufenden) Programmen zugänglich gemacht, ***wenn*** das Schlüsselwort **PUBLIC** angegeben wird
  - Können Unterprogramme als auch Funktionen darstellen
    - **EXPOSE**-Anweisung für Routine selbst nicht verfügbar
  - Definieren einen **eigenen Geltungsbereich**, wie wenn es sich um ein eigenes Programm handeln würde!
    - Sprungmarken ***innerhalb*** von Routinen zum Aufrufen von Unterprogrammen und Funktionen daher möglich

Routinen (Object Rexx): 1a

```
/**/
SAY  pp("hallo")
CALL oha          /* routine wird aufgerufen  */
SAY  pp("hallo")

EXIT 0
pp : RETURN "<<<" || ARG(1) || ">>>"

:: ROUTINE oha PUBLIC
  SAY  pp("holla")
  EXIT 0
  pp : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
<<<hallo>>>
[holla]
<<<hallo>>>
```

Routinen (Object Rexx): 1b

```
/**/
SAY  pp("hallo")
CALL oha          /* routine wird aufgerufen  */
SAY  pp("hallo")

EXIT 0
pp : RETURN "<<<" || ARG(1) || ">>>"

:: ROUTINE oha PUBLIC
   SAY  pp("holla")
   EXIT 0
   pp : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
<<<hallo>>>
[holla]
<<<hallo>>>
```

Routinen (Object Rexx): 1c

```
/**/
SAY  pp("hallo")
CALL oha          /* routine wird aufgerufen  */
SAY  pp("hallo")

EXIT 0
pp : RETURN "<<<" || ARG(1) || ">>>"
```

```
:: ROUTINE oha PUBLIC
  SAY  pp("holla")
  EXIT 0
  pp : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
<<<hallo>>>
[holla]
<<<hallo>>>
```

Routinen und Ausnahmebedingungen: 1

- Routinen wie externe Unterprogramme/Funktionen

```
/**/
SIGNAL ON USER ZU_KLEIN /* Benutzerausnahme abfangen */
CALL ueberpruefe 10
CALL ueberpruefe 3
CALL ueberpruefe 7
EXIT 0
ZU_KLEIN:                  /* Ausnahmebehandlung       */
  SAY "// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\"
  EXIT -1
::ROUTINE ueberpruefe
  PARSE ARG alter
  SAY "--> alter:" alter
  IF alter < 6 THEN RAISE USER zu_klein
               ELSE SAY "--> Überprüfung ist o.k."
  EXIT 0
```

Ausgabe:

```
--> alter: 10
--> Überprüfung ist o.k.
--> alter: 3
// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\
```

Routinen und Ausnahmebedingungen: 2

- Routinen wie externe Unterprogramme/Funktionen

```
/**/
CALL ON USER ZU_KLEIN /* Benutzerausnahme abfangen */
CALL ueberpruefe 10
CALL ueberpruefe 3
CALL ueberpruefe 7
EXIT 0
ZU_KLEIN:                    /* Ausnahmebehandlung       */
  SAY "// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\"
  RETURN
::ROUTINE ueberpruefe
  PARSE ARG alter
  SAY "--> alter:" alter
  IF alter < 6 THEN RAISE USER zu_klein
               ELSE SAY "--> Überprüfung ist o.k."
  EXIT 0
```

Ausgabe:

```
--> alter: 10
--> Überprüfung ist o.k.
--> alter: 3
// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\
--> alter: 7
--> Überprüfung ist o.k.
```

Routinen und Ausnahmebedingungen: 3a

```
CALL ON ANY
CALL ON USER ZU_KLEIN /* Benutzerausnahme abfangen */
CALL ON USER ZU_gross /* Benutzerausnahme abfangen */
CALL ueberpruefe 10
CALL ueberpruefe 3
CALL ueberpruefe 7
EXIT 0

ANY     : SAY "in Zeile:" SIGL "Ausnahme:" CONDITION("C")     ;RETURN
ZU_KLEIN: SAY "// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\";RETURN
zu_gross: SAY "// Ausnahmebedingung 'zu_gross' aufgefangen \\";RETURN
::ROUTINE ueberpruefe
  PARSE ARG alter
  SAY '--> alter:' alter
  IF alter < 6 THEN RAISE USER zu_klein
               ELSE IF alter > 9 THEN RAISE USER zu_gross
                    ELSE SAY '--> Überprüfung ist o.k.'
  RAISE USER irgend_etwas_aufgeworfen
  EXIT 0
```

Ausgabe:

```
--> alter: 10
// Ausnahmebedingung 'zu_gross' aufgefangen \\
--> alter: 3
// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\
--> alter: 7
--> Überprüfung ist o.k.
in Zeile: 7 Ausnahme: USER IRGEND_ETWAS_AUFGEWORFEN
```

Routinen und Ausnahmebedingungen: 3b

```
CALL ON ANY
CALL ON USER ZU_KLEIN /* Benutzerausnahme abfangen */
CALL ON USER ZU_gross /* Benutzerausnahme abfangen */
CALL ueberpruefe 10
CALL ueberpruefe 3
CALL ueberpruefe 7
EXIT 0

ANY     : SAY "in Zeile:" SIGL "Ausnahme:" CONDITION("C")     ;RETURN
ZU_KLEIN: SAY "// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\";RETURN
zu_gross: SAY "// Ausnahmebedingung 'zu_gross' aufgefangen \\";RETURN
::ROUTINE ueberpruefe
  PARSE ARG alter
  SAY '--> alter:' alter
  IF alter < 6 THEN RAISE USER zu_klein
               ELSE IF alter > 9 THEN RAISE USER zu_gross
                    ELSE SAY '--> Überprüfung ist o.k.'
  RAISE USER irgend_etwas_aufgeworfen
  EXIT 0
```

Ausgabe:

```
--> alter: 10
// Ausnahmebedingung 'zu_gross' aufgefangen \\
--> alter: 3
// Ausnahmebedingung 'ZU_KLEIN' aufgefangen \\
--> alter: 7
--> Überprüfung ist o.k.
in Zeile: 7 Ausnahme: USER IRGEND_ETWAS_AUFGEWORFEN
```

Requires Direktive (Object Rexx)

- Nach Aufruf eines Programmes werden sämtliche öffentlichen Routinen, die darin verfügbar sind, sichtbar
- **::Requires**-Direktive
  - Gibt ein Rexx-Programm an
    - Hinweis: auch Datentyp angeben (für Portierung wichtig)
  - Führt zum Aufruf des angegebenen Programmes während der Initialisierungsphase ***ehe*** die restlichen Direktiven (**::Routine**, **::Class**, **::Method**) befolgt werden
  - Anschließend stehen sämtliche öffentlichen Routinen und öffentlichen Klassen zur Verfügung, die in den aufgerufenen Programmen definiert sind

CALL-Anweisung und öffentliche Routinen: 1 / 2

```
/* cmd1.rex */
SAY "In" "cmd1.rex"
CALL cmd2
SAY "In" pp("cmd1.rex")
```

```
/* cmd2.rex */
SAY "  /1/ In" pp("cmd2.rex")
CALL cmd3
SAY "  /2/ In" pp("cmd2.rex")
EXIT 0

pp :
  RETURN "c2[" || ARG(1) || "]c2"
```

```
/* cmd3.rex */
SAY "    \1\ In" pp("cmd3.rex")
CALL cmd4
SAY "    \2\ In" pp("cmd3.rex")
EXIT 0

::ROUTINE pp
  RETURN "c3<<" || ARG(1) || ">>c3"
```

```
/* cmd4.rex */
SAY "          In" pp("cmd4.rex")
EXIT 0

pp :
  RETURN "c4<" || ARG(1) || ">c4"

::ROUTINE pp PUBLIC
  RETURN "c4<<" || ARG(1) || ">>c4"
```

Ausgabe:

```
In cmd1.rex
  /1/ In c2[cmd2.rex]c2
    \1\ In c3<<cmd3.rex>>c3
          In c4<cmd4.rex>c4
    \2\ In c3<<cmd3.rex>>c3
  /2/ In c2[cmd2.rex]c2
In c4<<cmd1.rex>>c4
```

CALL-Anweisung und öffentliche Routinen: 2 / 2

```
/* cmd1.rex */
SAY "In" "cmd1.rex"
CALL cmd2
SAY "In" pp("cmd1.rex")
```

```
/* cmd2.rex */
SAY "  /1/ In" pp("cmd2.rex")
CALL cmd3
SAY "  /2/ In" pp("cmd2.rex")
EXIT 0

pp :
  RETURN "c2[" || ARG(1) || "]c2"
```

```
/* cmd3.rex */
SAY "    \1\ In" pp("cmd3.rex")
CALL cmd4
SAY "    \2\ In" pp("cmd3.rex")
EXIT 0

::ROUTINE pp
  RETURN "c3<<" || ARG(1) || ">>c3"
```

```
/* cmd4.rex */
SAY "          In" pp("cmd4.rex")
EXIT 0

pp :
  RETURN "c4<" || ARG(1) || ">c4"

::ROUTINE pp PUBLIC
  RETURN "c4<<" || ARG(1) || ">>c4"
```

Ausgabe:

```
In cmd1.rex
  /1/ In c2[cmd2.rex]c2
    \1\ In c3<<cmd3.rex>>c3
          In c4<cmd4.rex>c4
    \2\ In c3<<cmd3.rex>>c3
  /2/ In c2[cmd2.rex]c2
In c4<<cmd1.rex>>c4
```

Requires-Direktive und öffentliche Routinen

```
/* cmd1.rex */
SAY "In" pp("cmd1.rex")

::REQUIRES cmd2.rex
```

```
/* cmd2.rex */
SAY "  /1/ In" pp("cmd2.rex")
EXIT 0

pp :
  RETURN "c2[" || ARG(1) || "]c2"

::Requires cmd3.rex
```

```
/* cmd3.rex */
SAY "    \1\ In" pp("cmd3.rex")
EXIT

:: requires cmd4.rex

::ROUTINE pp
  RETURN "c3<<" || ARG(1) || ">>c3"
```

```
/* cmd4.rex */
SAY "          In" pp("cmd4.rex")
EXIT

pp :
  RETURN "c4<" || ARG(1) || ">c4"

::ROUTINE pp PUBLIC
  RETURN "c4<<" || ARG(1) || ">>c4"
```

Ausgabe:

```
              In c4<cmd4.rex>c4
        \1\ In c3<<cmd3.rex>>c3
      /1/ In c2[cmd2.rex]c2
    In c4<<cmd1.rex>>c4
```