

Automatisierung von Java Anwendungen (2)

Object Rexx, 2

Prozeduren, Funktionen, Rexx-Funktionen,
„Stem“-Variable

Prof. Dr. Rony G. Flatscher

Sprungmarken

- Bezeichner, gefolgt von einem Doppelpunkt (:)
- Dienen als Ziel für
 - **CALL**-Anweisungen (Unterprogramm- und Funktionsaufrufe)
 - **SIGNAL**-Anweisungen (Ablaufveränderungen, ähnlich der "**GOTO**"-Anweisung in anderen Programmiersprachen)
 - Ausnahmebedingungen (**SIGNAL ON** bzw. **CALL ON**)

```
DO i = 1 TO 3
  SAY "Oho!" i
  IF i = 1 THEN SIGNAL fin
END
fin : SAY "C'est la fin!"
```

Ausgabe:

```
Oho! 1
C'est la fin!
```

Prozeduren (Unterprogramme), 1

- Zusammenfassung von Anweisungen, die wiederholt von unterschiedlichen Stellen aus durchlaufen werden sollen
- Beginnen mit einer Sprungmarke
- Aufruf
 - **CALL** Sprungmarke
 - Unterprogrammanweisungen werden ausgeführt
 - Die **RETURN**-Anweisung gibt die Kontrolle zurück (an die Anweisung, die unmittelbar auf den Unterprogrammaufruf folgt)

Prozeduren (Unterprogramme), 2

```
/* Ein Rexx-Programm ... */  
CALL ZeitStempel /* Prozeduraufruf */  
CALL SysSleep 10 /* 10 Sekunden schlafen */  
CALL ZeitStempel /* Prozeduraufruf */  
EXIT /* Programm verlassen */  
  
ZeitStempel : /* Sprungmarke der Prozedur */  
    SAY "Es ist jetzt ziemlich spät ..."  
    RETURN
```

Ausgabe:

```
Es ist jetzt ziemlich spät ...  
Es ist jetzt ziemlich spät ...
```

Funktionen, 1

- Prozeduren, die einen Wert ("Funktionswert") mit Hilfe der **RETURN**-Anweisung an den Aufrufer zurückgeben
- Aufruf
 - Variante 1
 - Aufruf, indem Sprungmarke unmittelbar von einer öffnenden und schließenden runden Klammer gefolgt wird
 - Anstelle des Aufrufs wird nach Rückkehr aus der Funktion der errechnete Funktionswert eingesetzt

```
heute = DATE()
```

- Variante 2
 - Aufruf, wie ein Unterprogramm
 - Interpreter sichert Rückgabewert in Variable **RESULT**

```
CALL DATE  
heute = result
```

Funktionen, 2

```
/* Ein Rexx-Programm ... */
SAY  ZeitStempel() /* Funktionsaufruf */
CALL SysSleep 10 /* 10 Sekunden schlafen */
CALL ZeitStempel /* Aufruf als Prozedur */
SAY  result /* Gib Funktionswert aus */
EXIT /* Programm verlassen */

ZeitStempel : /* Sprungmarke der Funktion */
RETURN "Es ist jetzt ziemlich spät ..."
```

Ausgabe:

```
Es ist jetzt ziemlich spät ...
Es ist jetzt ziemlich spät ...
```

Spezielle REXX-Variablen

- Nach dem Aufruf von Prozeduren, Funktionen und Kommandos werden von der REXX-Laufzeitumgebung folgende Variablen mit Werten versehen, die nach der Rückkehr zur Verfügung stehen
 - **RESULT**
Funktionswert wird zugewiesen, d.h. der Wert, der in der **RESULT**-Anweisung angeführt ist
 - **RC**
"Return Code" von (externen) Kommandos und (externen) Aufrufen
 - **SIGL**
"Signal Line" - Nummer jener Quellcodezeile, die eine Ausnahme (Fehler, Ausnahmebedingung) verursacht hat
[REXX-Funktion **SourceLine(sigl)** gibt jene Quellcodezeile aus, in der die Ausnahmebedingung erzeugt wurde]

Alle Funktionen der Sprache Rexx

- Rexx stellt folgende, zur Sprache gehörenden Funktionen zur Verfügung:

ABBREV()	CHARS()	FORM()	RANDOM()	TRUNC()
ABS()	COMPARE()	FORMAT()	REVERSE()	VALUE()
ADDRESS()	COPIES()	FUZZ()	RIGHT()	VAR()
ARG()	COUNTSR()	INSERT()	SETLOCAL()	VERIFY()
B2X()	D2C()	LASTPOS()	SIGN()	WORD()
BEEP()	D2X()	LEFT()	SOURCELINE()	WORDINDEX()
BITAND()	DATATYPE()	LENGTH()	SPACE()	WORDLENGTH()
BITOR()	DATE()	LINEIN()	STREAM()	WORDPOS()
BITXOR()	DELSTR()	LINEOUT()	STRIP()	WORDS()
C2D()	DELWORD()	LINES()	SUBSTR()	X2B()
C2X()	DIGITS()	MAX()	SUBWORD()	X2C()
CENTER()	DIRECTORY()	MIN()	SYMBOL()	XRANGE()
CHANGESTR()	ENDLOCAL()	OVERLAY()	TIME()	
CHARIN()	ERRORTXT()	POS()	TRACE()	
CHAROUT()	FILESPEC()	QUEUED()	TRANSLATE()	

Externe Rexx-Funktionspakete

- Standardisierte Schnittstellen von und zu Rexx
- Funktionspakete, die Rexx neue Funktionen zur Verfügung stellen, ohne daß sie Bestandteil der Sprache werden, z.B.
 - Direkte Zugriffe auf die wichtigsten relationalen Datenbanken (DB2, Oracle, SQL-Server, Sybase, etc.)
 - Z.B. Mark Hessling's "RexxSQL"
 - Ftp- bzw. TCP/IP-Socketprogrammierung (wird mitgeliefert)
 - Laden von externen Rexx-Funktionspaketen, z.B. von "RexxUtil" (wird mitgeliefert):


```
IF RxFuncQuery("SysLoadFuncs") THEN DO
    CALL RxFuncAdd "SysLoadFuncs","RexxUtil","SysLoadFuncs"
    CALL SysLoadFuncs /* ohne Hochkommata! */
END
```

Rexx-Funktionspaket "RexxUtil" (Ausschnitt)

- "RexxUtil"-Funktionspaket (eine DLL)
 - Umfaßt betriebssystemabhängige, "nützliche" Funktionen
 - ca. 90% der Funktionen in allen Implementierungen verfügbar
 - z.B. (Ausschnitt aus der Windows-Implementierung)

RxMessageBox()	SysFileSystemType()	SysSetPriority()
SysCls()	SysFileTree()	SysShutdownSystem()
SysCurPos()	SysMkDir()	SysSleep()
SysCurState()	SysOpenEventSem()	SysSwitchSession()
SysDriveInfo()	SysQueryRexxMacro()	SysTempFileName()
SysDriveMap()	SysQuerySwitchList()	SysTextScreenRead()
SysElapsedTime()	SysRmDir()	SysWaitForShell()
SysFileDelete()	SysSaveRexxMacroSpace()	SysWaitNamedPipe()
SysFileSearch()	SysSearchPath()	SysWildcard(), ...

Suchreihenfolge von Prozeduren und Funktionen, 1

- Suchreihenfolge von Prozeduren/Funktionen
 - Selbstprogrammierte Prozeduren/Funktionen im selben Programm
 - In die Sprache eingebaute Prozeduren/Funktionen
 - Externe Prozeduren/Funktionen (z.B. Rexx-Programme)
- Eingebaute Namen für Prozeduren/Funktionen können verwendet werden
 - Überdecken die entsprechenden Prozeduren/Funktionen
 - Originalfunktion kann jederzeit aufgerufen werden, indem  Funktion in **Großbuchstaben** in Anführungszeichen eingeschlossen wird

Suchreihenfolge von Prozeduren und Funktionen, 2

```
/* */  
SAY date() /* ruft selbstprogrammierte Funktion auf */  
SAY "DATE"() /* ruft eingebaute (!) REXX-Funktion auf */  
EXIT  
  
DATE : /* "DATE" ist eigentlich eine REXX-Funktion! */  
      RETURN "Date(), selbst programmiert!"
```

Ausgabe:

```
Date(), selbst programmiert!  
15 Mar 2004
```

Geltungsbereiche ("Reichweite"), 1

- Geben an, welche Variable und Sprungmarken in welchen Abschnitten eines Rexx-Programmes sichtbar sind
 - Grundsätzlich sind alle Variablen innerhalb eines Programmes global "sichtbar" und gehören daher **demselben Geltungsbereich** an
 - Sprungmarken sind innerhalb eines Programmes immer global
 - Wenn das Schlüsselwort **PROCEDURE** einer Sprungmarke folgt, dann wird für die Prozedur/Funktion ein eigener Geltungsbereich definiert

Soll trotzdem der Zugriff auf Variablen des Aufrufers ermöglicht werden, muß der **PROCEDURE**-Anweisung die **EXPOSE**-Anweisung mit einer Liste jener Variablen des Aufrufers folgen, auf die innerhalb des Geltungsbereiches der Prozedur/Funktion zugegriffen werden soll

Geltungsbereiche ("Reichweite"), 2

```
/* */  
a = 1  
b = 2  
SAY "a=" a "b=" b  
CALL rechne  
SAY "a=" a "b=" b  
EXIT
```

```
rechne :  
  a = a * 2  
  b = b * 3 / 4  
  RETURN
```

Ausgabe:

```
a= 1 b= 2  
a= 2 b= 1.5
```

Geltungsbereiche ("Reichweite"), 3

```
/* */  
a = 1  
b = 2  
SAY "a=" a "b=" b  
CALL rechne  
SAY "a=" a "b=" b  
EXIT
```

```
rechne: PROCEDURE /* kein Zugriff auf "a" und "b" ! */  
  a = 5 /* Variable "a" muß daher lokal definiert werden */  
  b = 6 /* Variable "b" muß daher lokal definiert werden */  
  a = a * 2  
  b = b * 3 / 4  
  RETURN
```

Ausgabe:

```
a= 1 b= 2  
a= 1 b= 2
```

Geltungsbereiche ("Reichweite"), 4

```
/* */  
a = 1  
b = 2  
SAY "a=" a "b=" b  
CALL rechne  
SAY "a=" a "b=" b  
EXIT
```

```
rechne: PROCEDURE EXPOSE b /* kein Zugriff auf "a",  
                             jedoch auf "b" ! */  
    a = 5 /* Variable "a" muß daher lokal definiert werden */  
    a = a * 2  
    b = b * 3 / 4  
    RETURN
```

Ausgabe:

```
a= 1 b= 2  
a= 1 b= 1.5
```


"Stamm"- ("stem"-) Variable (Assoziative Felder/Arrays), 1

- "Stamm"- (engl.: "stem"-) Variable
 - Bezeichner enthält einen oder mehrere **Punkte**
 - Die Zeichenkette von Anfang an bis zum ersten Punkt wird *Stamm* (englisch: *stem*) genannt
 - Beispiele:

```
a.n           = "aha"  
a.EinS       = 1  
a.1          = "Anton"  
Oesterreich.Tirol      = 750000  
Oesterreich.Tirol.Innsbruck = 135000  
SAY a.1 a.n a.EinS  
SAY Oesterreich.Tirol
```

Ausgabe:

```
Anton aha 1  
750000
```

"Stamm"- ("stem"-) Variable (Assoziative Felder/Arrays), 2

- Manche Funktionen aus Rexx-Funktionspaketen (z.B. *SysFileTree()* in *RexxUtil*) folgen einer Konvention, die nach dem Punkt nur eine ganze Zahl erlauben

– Stamm.0

- enthält die Anzahl der "Elemente", die mit eins beginnend bis inklusive dem Wert numeriert werden, der bei **Stamm.0** gespeichert ist

```
datei.1 = "max.doc"  
datei.2 = "moritz.doc"  
datei.0 = 2      /* maximale Anzahl an "Elementen" */  
DO i=1 TO datei.0  
    SAY datei.i  /* "i" wird auch als Index bezeichnet */  
END
```

Ausgabe:

```
max.doc  
moritz.doc
```

Zerlegungsanweisung – PARSE, 1

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
text = " Stiegler Seppl Stumm Zillertal/Tirol"  
PARSE VAR text fname vname rest  
SAY fname  
SAY vname  
SAY rest  
EXIT
```

Ausgabe:

```
Stiegler  
Seppl  
Stumm Zillertal/Tirol
```

Zerlegungsanweisung – PARSE, 2

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
lineal = COPIES("1234+6789|", 5)
text   = " Stiegler Seppl Stumm Zillertal/Tirol"
PARSE VAR text fname vname rest
SAY lineal; SAY text ; SAY
SAY pp(fname); SAY pp(vname)
SAY pp(lineal); SAY pp(rest)
EXIT
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
1234+6789|1234+6789|1234+6789|1234+6789|1234+6789|
  Stiegler   Seppl   Stumm   Zillertal/Tirol

[Stiegler]
[Seppl]
[1234+6789|1234+6789|1234+6789|1234+6789|1234+6789| ]
[ Stumm      Zillertal/Tirol]
```

Zerlegungsanweisung – PARSE, 3

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
          /*          10          20          30          40
          1234+6789|1234+6789|1234+6789|1234+6789| */
text = "  Ruaniger Annelle  Stumm  Zillertal / Tirol "
PARSE VAR text vorher " / " nachher
SAY pp(vorher)
SAY pp(nachher)
EXIT
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
[  Ruaniger Annelle  Stumm  Zillertal ]
[ Tirol ]
```

Zerlegungsanweisung – PARSE, 4

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
muster = "/"
/*          10          20          30          40
1234+6789|1234+6789|1234+6789|1234+6789| */
text = "  Ruaniger Annelle   Stumm   Zillertal / Tirol "
PARSE VAR text vorher (muster) nachher
SAY pp(vorher)
SAY pp(nachher)
EXIT
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
[  Ruaniger Annelle   Stumm   Zillertal ]
[ Tirol ]
```

Zerlegungsanweisung – PARSE, 5

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
/*           10           20           30           40
1234+6789|1234+6789|1234+6789|1234+6789| */
text = "  Ruaniger Annelle  Stumm  Zillertal / Tirol  "
PARSE VAR text 3 fname +8 12 vname ort .
SAY pp(fname)
SAY pp(vname)
SAY pp(ort)
EXIT
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
[Ruaniger]
[Annelle]
[Stumm]
```

Zerlegungsanweisung – PARSE, 6

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
text = "Sattler;Cilli;Stumm;Zillertal/Tirol"  
PARSE VAR text fname ";" vname ";" ort  
SAY pp(fname)  
SAY pp(vname)  
SAY pp(ort)  
EXIT  
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
[Sattler]  
[Cilli]  
[Stumm;Zillertal/Tirol]
```


Zerlegungsanweisung – PARSE, 7

PARSE-Anweisung erlaubt das Zerlegen von Zeichenketten und das Zuweisen zu Variablen in einem Schritt

```
text = ";Sattler;Cilli;Stumm;Zillertal/Tirol,"  
PARSE VAR text 1 a +1 fname (a) vname (a) ort (a) .  
SAY pp(fname)  
SAY pp(vname)  
SAY pp(ort)  
EXIT  
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
[Sattler]  
[Cilli]  
[Stumm]
```

Eingaben von „STDIN:“ (Tastatur)

PARSE PULL, PULL

PARSE PULL-Anweisung erlaubt das Zerlegen von Zeichenketten, die über die Tastatur eingegeben werden, und das Zuweisen zu Variablen in einem Schritt

```
SAY "1. Wie heißt Du denn?"      /* Eingabe: "Max" */
PARSE PULL name
SAY "Du heißt:" pp(name)
SAY "2. Wie heißt Du denn?"      /* Eingabe: "moritz" */
PULL name
SAY "Du heißt:" pp(name)
EXIT
PP : RETURN "[" || ARG(1) || "]"
```

Ausgabe:

```
1. Wie heißt Du denn?
Max
Du heißt: [Max]
2. Wie heißt Du denn?
moritz
Du heißt: [MORITZ]
```

Argumente entgegennehmen

PARSE ARG

PARSE ARG-Anweisung erlaubt das Zerlegen von Argumenten und das Zuweisen zu Variablen in einem Schritt

```
/* */  
a = 1; b = 2  
SAY "a=" a "b=" b  
CALL rechte a , b  
SAY "a=" a "b=" b  
EXIT
```

```
rechte: PROCEDURE /* kein Zugriff auf "a" und "b" !*/  
  PARSE ARG a , b  
  SAY "rechte: a=" a "b=" b  
  a = a * 2  
  b = b * 3 / 4  
  SAY "rechte: a=" a "b=" b  
  RETURN
```

Ausgabe:

```
a= 1 b= 2  
rechte: a= 1 b= 2  
rechte: a= 2 b= 1.5  
a= 1 b= 2
```