

Automatisierung von Java Anwendungen (1)

Object REXX, 1

Einführung, Überblick

Anweisungen, Prozeduren, Funktionen

Prof. Dr. Rony G. Flatscher

Ziele der Lehrveranstaltung

- Bogen der Lehrveranstaltung
 - Einführung in die Konzepte des prozeduralen und objektorientierten Programmierens
 - Skriptsprache Object Rexx
 - Automatisierung von Java Anwendungen
 - BSF
 - JSR-223
- Grundlagen-Knowhow
 - Dauerhaftes Grundlagen- und Anwendungswissen
 - Mit geringem Aufwand können **Sie** es auf Dauer **aktuell** halten !
 - Wettbewerbsvorteil gegenüber von KollegInnen im Berufsleben

Übersicht, 1

- Lehrveranstaltung
 - Grundlegende Konzepte des objektorientierten Paradigmas
 - Object Rexx
 - Architektur von COM, WSH
 - Automatisierung von Windows Anwendungen
 - Gruppenarbeit
 - Kleingruppen von 2-3 Personen
 - Zusammensetzung ändert sich von Termin zu Termin
 - Vorlage

<http://wi.wu-wien.ac.at/Studium/LVA-Unterlagen/rgf/autowin/fohlen>

Übersicht, 2

- Warum Rexx bzw. Object Rexx?
 - Einfache Syntax ("human centric language")
 - Schnell und leicht zu erlernen
 - Mächtiges Objektmodell
 - Alle wesentlichen Konzepte des OO-Paradigmas verfügbar
 - Windows Scripting Engine (WSE)
 - Volle Automatisierung von Windows-Anwendungen möglich
- Software
 - Auf CD-ROM in der Lehrveranstaltung oder unter:
<http://www.ibm.com/software/info/university/>

Übersicht, 3

- Erster Teil
 - Object Rexx
 - Ausführliche Einführung/Wiederholung des OO-Paradigmas
 - Windows-bezogene Erweiterungen
- Zweiter Teil
 - Windows Automatisierung
 - OLE-Automation
 - Windows Scripting Host (WSH)
 - Erstellung eines kleinen Projektes und dessen Präsentation

- "Interaktive" Lehrveranstaltung
 - Bitte Fragen stellen!
 - Keine Angst vor "falschen", "leichten" oder "lächerlichen" Fragen
 - Fragesteller/in konzentriert sich besonders auf die Antwort (leichteres bzw. vertieftes Erlernen von neuen Konzepten)
 - Fragesteller/in kann sich üblicherweise darauf verlassen, daß andere Zuhörer/innen dieselbe Frage stellen würden
- Benutzen Sie in Ihrem eigenen Interesse den Listserver !

- 1979 - IBM (**Mike F. Cowlshaw**, IBM-Fellow)
 - Nachfolger von einer kryptischen Skriptsprache ("EXEC") auf IBM Mainframes
 - Ziel: menschenfreundliche ("human centric") Sprache
 - Interaktiv (Interpreter)
 - REXX Akronym für "**RE**structured **eX**tended **eX**ecutor"
- Ab 1987 IBM's "SAA" für "Procedural Language"
 - Strategische Skriptsprache für sämtliche IBM-Plattformen
 - Entwicklung von kommerziellen und freien Versionen der Sprache, defacto für alle Betriebssysteme verfügbar
- ANSI REXX Standard in 1996
 - ANSI "Programming Language - REXX", X3.274-1996

Geschichte, 2

- Seit Anfang der 90er
 - auf Initiative der einflußreichen IBM-Benutzervereinigung "SHARE" Entwicklung einer objektorientierten Version von REXX
- "Object-based REXX" a.k.a. "Object REXX"
 - Vollständig kompatibel zum herkömmlichen (prozeduralen) Rexx
 - Intern vollständig objektorientiert aufgebaut
 - *Prozedurale Anweisungen werden intern in objektorientierte umgeformt !*
 - Mächtiges Objektmodell (z.B. Metaklassen, Mehrfachvererbung)
 - Weiterhin einfache Syntax
 - Verfügbarkeit
 - 1997 für OS/2 Warp 4 (frei) und Warp 3 (frei) (mit [SOM](#))
 - 1998 für AIX (Version zum Evaluieren) und [Linux](#) (frei)
 - 1998 für Windows 95 und Windows NT (mit [OLEAutomation/ActiveX](#))

Geschichte, 3

- Seit 1996 Entwicklung von "NetRexx" durch den Autor von Rexx M.F. Cowlshaw
 - Java in den "Kleidern" von Rexx
 - NetRexx-Programme werden in Java übersetzt
 - Vereinfachte Java-Programmierung, aufgrund der Rexx-Syntax ~30% weniger Code (syntaktischen Elementen) als in Java selbst
 - Da im wesentlichen Rexx-Syntax, relativ einfach zu erlernen
- URLs zu Rexx, Object Rexx, NetRexx

<http://www.RexxLA.org/>

<http://www.software.ibm.com/ad/obj-rexx/>

<http://www2.hursley.ibm.com/rexx/>

<http://www2.hursley.ibm.com/orexx/>

<http://www2.hursley.ibm.com/netrexx/>

<news:comp.lang.rexx>

Grundlagen

Minimales Rexx-Programm

```
/* Kommentar beginnt in 1. Zeile, 1. Spalte */  
SAY "Hallo, liebe Welt"
```

Ausgabe:

```
Hallo, liebe Welt
```

Grundlagen

RexxTry.cmd

- "RexxTry.cmd"
 - Rexx-Programm zum Interaktiven Ausführen von Rexx-Anweisungen
 - Leitet jede Rexx-Anweisung dem Rexx-Interpreter zur Ausführung zu
 - Zeigt Erfolg oder Mißerfolg von Rexx-Anweisungen an
 - Wird durch die Eingabe von **EXIT** beendet
- Windows
 - "RexxTry.**rex**"
 - Aufruf von der Kommandozeile:

```
rexex RexxTry.rex
```

```
rexex rexstry
```

Grundlagen

Schreibweise von Programmtext

- Groß- und Kleinschreibung irrelevant
 - Die Buchstaben einer Rexx-Anweisung werden zunächst in Großbuchstaben umgewandelt und erst dann ausgeführt
 - Ausnahme: Inhalt von Zeichenketten bleibt unangetastet
 - Zeichenketten werden in Apostrophen (') oder Anführungszeichen (") eingeschlossen, z.B.

"Anton" , *'Anton'* , *"\{[]}\gulp!öäüß!{niX }"*

- Mehrere Leerzeichen werden auf ein Leerzeichen reduziert
 - Beispiel

```
saY      "\{[]}\gulp!öäüß!{niX }"      reverse(      Abc  )
```

wird zu:

```
SAY "\{[]}\gulp!öäüß!{niX }" REVERSE( ABC )
```

Grundlagen

Zeichen

- Zeichen außerhalb von Zeichenketten und Kommentaren dürfen nur aus dem folgenden Zeichenvorrat stammen
 - Leerzeichen
 - **a** bis **z**
 - **A** bis **Z**
 - **0** bis **9**
 - Ausrufungszeichen (!), Backslash (\), Fragezeichen (?), Gleichheitszeichen (=), Komma (,), Minus (-), Plus (+), Punkt (.), Schrägstrich (/), runde Klammern (()), eckige Klammern ([]), Sternzeichen (*), Strichpunkt (;) sowie Unterstrich (_)

Grundlagen Variablen

- Variablen erlauben das Speichern, Ändern und das Abrufen von Zeichenketten mit Hilfe eines willkürlich vergebenen *Bezeichners*

```
A = "Hallo, liebe Welt"  
a="Hallo, liebe Variable"  
A = a " - wieder geändert."  
say a
```

Ausgabe:

```
Hallo, liebe Variable - wieder geändert.
```

- Bezeichner für Variable dürfen mit den lateinischen Buchstaben, dem Rufezeichen, dem Fragezeichen und dem Unterstrich beginnen und dürfen zusätzlich diese Zeichen sowie Punkte und Ziffern enthalten

Grundlagen Konstanten

- Konstanten verändern im Gegensatz zu Variablen ihren Wert nie und erhalten in Rexx auch keine Bezeichner ("Literale")
 - Sollen Konstante über Bezeichner abrufbar sein, stehen grundsätzlich zwei Varianten zur Verfügung
 - der konstante Wert wird einer Variablen zugewiesen, deren Wert in keinem Programmteil mehr verändert wird

```
Pi = 3.14159
```

- der konstante Wert wird in die lokale (`.local`) oder globale (`.environment`) Umgebung eingetragen und über Umgebungssymbole ("environment symbol") abgerufen, die mit einem Punkt beginnen

```
.local~pi = 3.14159 /* Speichern eines Wertes */  
say .pi          /* Abrufen; gibt den Wert 3.14159 aus */
```

Grundlagen

Kommentare

- Kommentare können geschachtelt (ineinandergefügt) sein und auch mehrere Zeilen umfassen

```
say 3 + /* Das /**/ ist
      ein /* geschachtelter
          /* aha*/ Kommentar*/ der sich
          über mehrere Zeilen erstreckt */ 4
```

Ausgabe:

7

- Zeilenkommentare: am Ende einer Zeile Kommentare nach einem doppelten Bindestrich:

```
say 3 + 4 -- das ergibt nach Adam Riese sieben!
```

Ausgabe:

7

Grundlagen

Anweisungen, 1

- Anweisungen umfassen sämtliche Zeichen bis zum Strichpunkt (;)
- Es können beliebig viele Anweisungen in einer Zeile stehen
- Fehlt der Strichpunkt, so wird die Anweisung mit dem Ende der Zeile angenommen

```
/* Konvention: Kommentar beginnt in 1. Zeile, 1. Spalte */  
SAY "Hallo, liebe Welt";
```

Ausgabe:

```
Hallo, liebe Welt
```

Grundlagen

Anweisungen, 2

- Anweisungen können über mehrere Zeilen aufgeteilt werden
 - Beistrich als letztes Zeichen in der Zeile
 - Bindestrich als letztes Zeichen in der Zeile

```
/* Kommentar beginnt in 1. Zeile, 1. Spalte */  
SAY "Hallo," ,  
    "liebe Welt";
```

Ausgabe:

```
Hallo, liebe Welt
```

Grundlagen Block

- Ein Block ist eine Anweisung, die beliebig viele Anweisungen umschließt
- Ein Block beginnt mit dem Schlüsselwort **DO** und endet mit **END**

```
DO
  SAY "Hallo," ;
  SAY "liebe Welt" ;
END;
```

```
DO
  SAY "Hallo,"
  SAY "liebe Welt"
END
```

Ausgabe:

```
Hallo,
liebe Welt
```

Grundlagen

Vergleiche (Testausdrücke), 1

- Zwei Werte (Konstante, Variable, Funktionsergebnisse) können miteinander mit den folgenden (Infix) Operatoren verglichen werden (Ergebnis: 0=falsch oder 1=wahr)

=		gleich
<>	!= \=	ungleich
<		kleiner
<=		kleiner gleich
>		größer
>=		größer gleich

- Negation von Boole'schen Werten (0=falsch, 1=wahr)

\ Negator

Grundlagen

Vergleiche (Testausdrücke), 2

- Boole'sche Werte können miteinander verknüpft werden

& "und" (**wahr**: wenn beide Argumente wahr)

| "oder" (**wahr**: beliebiges Argument wahr)

&& "exklusives oder" (**wahr**: ein Argument wahr und eines falsch)

- Boole'sche Verknüpfungsausdrücke können mit Klammern ausdrücklich gereiht werden

0 & 1		1	Ergebnis: 1 (= wahr)
(0 & 1)		1	Ergebnis: 1 (= wahr)
0 & (1		1)	Ergebnis: 0 (= falsch)

Grundlagen

Vergleiche (Testausdrücke), 3

```
a=1  
b=2  
x="Anton"  
y=" Anton "
```

If a = 1 then ...	Ergebnis: 1 (= wahr)
If a = a then ...	Ergebnis: 1 (= wahr)
If a >= b then ...	Ergebnis: 0 (= falsch)
If x = y then ...	Ergebnis: 1 (= wahr)
If x == y then ...	Ergebnis: 0 (= falsch)
a <= b & (a = 1 b > a)	Ergebnis: 1 (= wahr)
\(a <= b & (a = 1 b > a))	Ergebnis: 0 (= falsch)
\a	Ergebnis: 0 (= falsch)

Grundlagen

Verzweigung, 1

- Eine Verzweigung wählt aufgrund eines Vergleiches aus, ob eine Anweisung (ein Block) ausgeführt werden soll oder nicht
 - **IF** test_ausdruck **THEN** anweisung;
- Üblicherweise kann aufgrund eines Tests auch eine alternative Anweisung zum Abarbeiten vorgesehen werden
- **IF** test_ausdruck **THEN** anweisung; **ELSE** anweisung;

- Beispiel:

```
IF Alter < 19 THEN SAY "Jung."
```

- Beispiele:

```
IF Alter < 19 THEN SAY "Jung.";  
ELSE SAY "Alt."
```

```
IF Alter < 1 THEN  
DO  
    SAY "Hallo,"  
    SAY "liebe Welt"  
END
```

Grundlagen

Verzweigung, 2

- Mehrfachauswahl (**SELECT**)

SELECT

```
WHEN testausdruck THEN anweisung;
```

```
WHEN testausdruck THEN anweisung;
```

```
/* ... beliebig viele weitere WHEN-Anweisungen */
```

```
OTHERWISE anweisung;
```

END

Mehrfachauswahl (**SELECT**) - Beispiel

```
SELECT
```

```
WHEN Alter = 1 THEN SAY "Baby." ;
```

```
WHEN Alter = 6 THEN SAY "Volksschulkind." ;
```

```
WHEN Alter >= 10 THEN SAY "Großes Kind." ;
```

```
OTHERWISE SAY "Uninteressant." ;
```

```
END
```


Grundlagen

Wiederholung, 1

- Grundsätzlich kann ein Block wiederholt durchlaufen werden

```
DO 3  
    SAY "Aua! "  
    SAY "Oh! "  
END
```

Ausgabe:

```
Aua!  
Oh!  
Aua!  
Oh!  
Aua!  
Oh!
```

Grundlagen

Wiederholung, 2

- Variable Wiederholungen

```
a = 3
...
DO a
    SAY "Aua! "; SAY "Oh!"
END
```

Ausgabe:

```
Aua!
Oh!
Aua!
Oh!
Aua!
Oh!
```

Grundlagen

Wiederholung, 3

- Wiederholungen mit Laufvariable

```
DO i = 1 TO 3
  SAY "Aua! "; SAY "Oh! " i
END
```

Ausgabe:

```
Aua!  
Oh! 1  
Aua!  
Oh! 2  
Aua!  
Oh! 3
```

Grundlagen

Wiederholung, 4

- Wiederholungen mit Laufvariable

```
DO i = 1 TO 3 BY 2  
    SAY "Aua!"; SAY "Oh!" i  
END
```

Ausgabe:

```
Aua!  
Oh! 1  
Aua!  
Oh! 3
```

Grundlagen

Wiederholung, 5

- Wiederholungen mit Laufvariable

```
DO i = 3.1 TO 5.7 BY 2.1  
  SAY "Aua!"; SAY "Oh!" i  
END
```

Ausgabe:

```
Aua!  
Oh! 3.1  
Aua!  
Oh! 5.2
```

Grundlagen

Wiederholung, 6

- Bedingte Wiederholungen

```
i = 2  
DO WHILE i < 3  
    SAY "Aua!"; SAY "Oha!" i  
    i = i + 1  
END
```

Ausgabe:

Aua!

Oha! 2

Grundlagen

Wiederholung, 7

- Bedingte Wiederholungen

```
i = 3
DO WHILE i < 3
    SAY "Aua!"; SAY "Oha!" i
    i = i + 1
END
```

➔ **Keine Ausgabe, da Block nicht durchlaufen wird!**

Grundlagen

Wiederholung, 8

- Bedingte Wiederholungen

```
i = 3
DO UNTIL i > 1
    SAY "Aua!"; SAY "Oha!" i
    i = i + 1
END
```

Ausgabe:

Aua!

Oha! 3

Grundlagen

Ausführung , 1

```
/* */  
a = 3  
b = "4"  
say a b  
say a b  
say a || b  
say a + b
```

Ausgabe:

```
3 4  
3 4  
34  
7
```

Grundlagen

Ausführung , 2

```
/* */  
"del *.*"
```

oder:

```
/* */  
ADDRESS CMD "del *.*"
```

oder:

```
/* */  
a = "del *.*"  
a
```

oder:

```
/* */  
a = "del *.*"  
ADDRESS CMD a
```