

2024 REXxLA International Rexx Language Symposium Proceedings

René Vincent Jansen (ed.)

THE REXX LANGUAGE ASSOCIATION
REXXLA Symposium Proceedings Series
ISSN 1534-8954

Publication Data

©Copyright The Rexx Language Association, 2024

All original material in this publication is published under the Creative Commons - Share Alike 3.0 License as stated at <https://creativecommons.org/licenses/by-nc-sa/3.0/us/legalcode>.

A publication of **RexxLA Press**

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

The RexxLA Symposium Series is registered under ISSN 1534-8954
The 2024 edition is registered under ISBN 978-9-4-0-36-0-000-0



2024-03-12 First Edition
2024-05-05 Second Edition

Introduction

History of the International REXX Language Symposium

In 1990, Cathie Dager of SLAC¹ convened the organizing committee for the first independent REXX² Symposium for Developers and Users. SLAC continued to organize this annual event until the middle of the 1990's when the REXXLA took over that responsibility. Symposia have been held annually since 1990.

About REXXLA

During the 1993 Symposium in La Jolla, California, plans for a REXX User Group materialized. The REXX Language Association (REXXLA), as it was called, is an independent, non-profit organization dedicated to promoting the use and understanding of the REXX programming language. REXXLA manages several open source implementations of REXX.

The selection procedure

Presentation proposals are solicited yearly using a CFP³ procedure, after which the REXXLA symposium committee reviews them and votes which presentations are selected for the symposium. The presentations are peer reviewed before being presented. Presenters are not compensated for their presentations.

Location

The 2024 symposium was held in Brisbane, Australia and online from 3 Mar 2024 to 6 Mar 2024.

Organizing Committee

- Jon Wolfers
- Mark Hessling
- René Jansen

¹Stanford Linear Accelerator Center, since 2008 SLAC National Accelerator Laboratory

²Cowlshaw, M. F., **The REXX Language** (second edition), ISBN 0-13-780651-5, Prentice-Hall, 1990.

³Call For Papers.

Contents

1	Tutorial: From Rexx to ooRexx – Rony G. Flatscher	1
2	Tutorial: Stems a Different Way – Rony G. Flatscher	19
3	Tutorial: Making Java easy using ooRexx - The Bean Scripting Framework for ooRexx – Rony G. Flatscher	36
4	The State of Rexx 2024 – René Vincent Jansen	59
5	SBC Arm Linux Rexx Stack Build Environment – Tony Dycks	70
6	JDOR - Java2D for ooRexx (and Other Programming Languages) – Rony G. Flatscher	88
7	Rosetta Pearls – Walter Pachl	98
8	The Unicode Tools Of Rexx (TUTOR) – Josep Maria Blasco	109
9	A tokenizer for Rexx and ooRexx – Josep Maria Blasco	151
10	z/OS Control blocks and the Rexx Storage BIF – René Vincent Jansen	180
11	Multithreading in ooRexx: Concepts, Nutshell Examples – Rony G. Flatscher	206
12	Debugging Multithreaded ooRexx Programs – Rony G. Flatscher	216
13	Releasing the ooRexx-Java Bridge BSF4ooRexx850 – Rony G. Flatscher	230
14	Design Changes for next release of Rexx/SQL – Mark Hessling	241
15	RXPipes for z/OS – Willy Jensen	252
16	Creating Powerful and Portable GUIs with JavaFX – Rony G. Flatscher	262
17	The RexxKnowledge Repository – Till Winkler	287
18	ooRexx and Character Sets (Dealing with UTF-8) – Rony G. Flatscher	292
19	Using ooRexx and JSoup for XML and HTML Processing and Conversions – Rony G. Flatscher	299

Tutorial: From Rexx to ooRexx – Rony G. Flatscher

Date and Time

3 Mar 2024, 04:00:00 CET

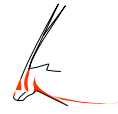
Presenter

Rony G. Flatscher

Presenter Details

Rony works as a professor for Business informatics (“Wirtschaftsinformatik”) at the Vienna University of Economics and Business Administration (Wirtschaftsuniversität Wien) and uses Open Object REXX for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooREXX, the ooREXX-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

Tutorial "From Rexx to ooRexx"



The 2024 International Rexx Symposium
Brisbane, Queensland, Australia
March 3rd – March 6th 2024

© 2024 Rony G. Flatscher (Rony.Flatscher@wu.ac.at)
Wirtschaftsuniversität Wien, Austria (<http://www.wu.ac.at>)



Agenda



- Brief History
- Getting Object Rexx
- Some new features like
 - `USE ARG`
- New message expressions (“message paradigm”)
- New instruction type: Directives
 - `::ROUTINE`, `::REQUIRES`
 - `::CLASS`, `::ATTRIBUTE`, `::METHOD`
 - `(::ANNOTATE`, `::CONSTANT`, `::OPTIONS`,
`::RESOURCE)`
- Roundup



Brief History, 1



- Begin of the 90s, an IBM product
 - OO-version of REXX (Object REXX) presented to the IBM user group "SHARE"
 - Developed since the beginning of the 90s
 - 1997 Introduced with OS/2 Warp 4
 - *Support of SOM and WPS*
 - 1998 Free Linux version, trial version for AIX
 - 1998 Windows 95 and Windows/NT

3



Brief History, 2



- RexxLA and IBM negotiate
 - 2004 IBM handed over source code to RexxLA
 - "Open Object REXX (ooRexx) 3.0"
 - Open source version of IBM's Object REXX
 - Released by RexxLA: 2005-03-25
 - ooRexx 3.1 (2006), ooRexx 3.2 (2008)
 - ooRexx 4.0 (2009)
 - New kernel, 32- and 64-bit became possible
 - ooRexx 4.1 (2011), ooRexx 4.2 (2014)
 - ooRexx 5.0 (2022)
 - ooRexx 5.1 in beta (2024)

4



REXX Instructions



- REXX is based on three instruction types
 - Assignment instruction
 - Second token is an equal sign, e.g.
`age = 12`
 - Keyword instruction
 - First token is a REXX keyword instruction, e.g.
`say "1/3:" 1/3`
 - Command instruction
 - Any other statement is regarded to be a command, e.g.
`"echo hello world"`

5



ooRexx Instructions



- All REXX instructions
 - Assignment instruction
 - Keyword instruction
 - Command instruction
- ooRexx adds a new instruction type, the directive instruction
 - Placed at the end of the program
 - Processed in the ooRexx setup phase of the program („package“)
 - Led in by two colons
 - Example:
`::options digits 20`

4

6





ooRexx Message Expressions



- ooRexx adds message expressions consisting of a
 - Receiver
 - Message operator ~ (tilde, a.k.a. „twiddle“)
 - Message name
 - Can carry arguments in parentheses
- Example
 - `'abc'~reverse`
 - Receiver is the string `'abc'`
 - Message name is `reverse`
 - Result is the reversed string `cba`
- For strings this is a new alternative to using the REXX built-in functions (BIF), in this example `reverse('abc')`

7



ooRexx Processing



- The ooRexx interpreter processes Rexx and ooRexx programs in three phases
 - Phase 1: loading phase
 - Program text gets loaded and syntax checked
 - If a syntax condition gets raised the processing stops
 - Phase 2: setup phase
 - All directive instructions get carried out by ooRexx
 - Allows e.g. for creating classes (types, structures), routines, carrying out (calling) and more
 - Phase 3: execution phase
 - Program starts by executing the first statement at the top

5

8



Some New Features



- Compatible with classic REXX, TRL 2
 - **New** sequence of execution of REXX programs:
 - (Load) Phase 1: **Full syntax check** of the REXX program upfront
 - (Setup) Phase 2: Interpreter carries out all directives (lead in with ":::")
 - (Execution) Phase 3: Start of program execution with line # 1
- **rexxc[.exe]**: compiles REXX programs
 - If *same bitness* and *same endianness*, on *all* platforms
- **USE ARG** in addition to **PARSE ARG**
 - among other things allows for retrieving stems *by reference* (!)
- Line comments, led in by two dashes ("--")
 - *comment until the line ends*

9



Stem, Classic REXX "stemclassic.rex"



```
s.1="Entry # 1"
s.2="Entry # 2"
s.0=2          /* total number of entries in stem      */

call add2stem /* add to stem using an (internal) routine */

do i=1 to s.0 /* iterate over all stem array entries      */
  say "#" i:" s.i
end
exit

add2stem: procedure expose s, -- allow access to stem
  n=s.0+1      /* add after last current entry                    */
  s.n="Entry #" n "added in add2stem()"
  s.0=n        /* update total number of entries in stem          */
  return

/* yields:

# 1: Entry # 1
# 2: Entry # 2
# 3: Entry # 3 added in add2stem()

*/
```

6

10





Stem, REXX with USE ARG

"stemusearg.rex": No EXPOSE



```

s.1="Entry # 1"
s.2="Entry # 2"
s.0=2          /* total number of entries in stem      */

call add2stem s. /* supply stem as an argument!                    */

do i=1 to s.0   /* iterate over all stem array entries             */
  say "#" i ":" s.i
end
exit

add2stem: procedure /* no "expose s." needed anymore !                */
  use arg s.      /* USE ARG allows to directly refer to the stem */
  n=s.0+1         /* add after last current entry                  */
  s.n="Entry #" n "added in add2stem()"
  s.0=n          /* update total number of entries in stem      */
  return

/* yields:

# 1: Entry # 1
# 2: Entry # 2
# 3: Entry # 3 added in add2stem()

*/

```

11



Stem, ooRexx USE ARG

"stemroutine1.rex": No EXPOSE



```

s.1="Entry # 1"
s.2="Entry # 2"
s.0=2          /* total number of entries in stem      */

call add2stem s. /* supply stem as an argument!                    */

do i=1 to s.0   /* iterate over all stem array entries             */
  say "#" i ":" s.i
end

::routine add2stem
  use arg s.      /* USE ARG allows to directly refer to the stem */
  n=s.0+1         /* add after last current entry                  */
  s.n="Entry #" n "added in add2stem()"
  s.0=n          /* update total number of entries in stem      */
  return

/* yields:

# 1: Entry # 1
# 2: Entry # 2
# 3: Entry # 3 added in add2stem()

*/

```

7

12





Stem, ooRexx USE ARG

"stemroutine2.rex": No EXPOSE



```

s.1="Entry # 1"
s.2="Entry # 2"
s.0=2          /* total number of entries in stem */

call add2stem s. /* supply stem as an argument! */

do i=1 to s.0  /* iterate over all stem array entries */
  say "#" i:" s.i
end

::routine add2stem /* we can even use a different stem name */
  use arg abc. /* USE ARG allows to directly refer to the stem */
  n=abc.0+1 /* add after last current entry */
  abc.n="Entry #" n "added in add2stem()"
  abc.0=n /* update total number of entries in stem */
  return

/* yields:

# 1: Entry # 1
# 2: Entry # 2
# 3: Entry # 3 added in add2stem()

*/

```

13



About Directives in ooRexx



- Always placed at the end of a Rexx program
 - led in by "::" followed by the name of the directive
 - "routine", "class", "attribute", "method", ...
- Instructions to the ooRexx interpreter before program starts
 - Interpreter sequentially processes and carries out directives in the *setup phase* (phase 2) of startup
 - After all directives got carried out, the *execution phase of the Rexx program* starts by executing the first line
- An ooRexx program with directives
 - Defines a "package" of routines and classes
 - Rexx code before the first directive is also named "prolog"

8

14



::Routine Directive



- Syntax

`::routine name [public]`

- Interpreter maintains routines (and classes) per REXX program ("package")
- If optional keyword **public** is present, the routine can be also *directly invoked by another (!) REXX program*

15



::ROUTINE Directive, Example "routine.rex"



```
r=" 1 "  
s=2  
say "x="pp(x)  
say "s="pp(s)  
say  
say "The result of 'r || 3 ' is:" pp(r || 3 )  
say "The result of 's || 3 ' is:" pp(s || 3 )  
say "The result of 'r + 3' is:" pp(r + 3)  
say "The result of 's + 3' is:" pp(s + 3)  
say  
say "The result of 'r s' is:" pp(r s)  
say "The result of 'r || s' is:" pp(r || s)  
say "The result of 'r+s' is:" pp(r+s)  
  
::routine pp -- enclose argument in square brackets  
  parse arg value  
  return ["value"]  
  
/* yields:  
  
  r=[ 1 ]  
  s=[2]  
  
  The result of 'r || 3 ' is: [ 1 3]  
  The result of 's || 3 ' is: [23]  
  The result of 'r + 3' is: [4]  
  The result of 's + 3' is: [5]  
  
  The result of 'r s' is: [ 1 2]  
  The result of 'r || s' is: [ 1 2]  
  The result of 'r+s' is: [3]  
*/
```

9

16





::ROUTINE Directive, Example "toolpackage.rex"



```
-- collection of useful little REXX routines

::routine pp public -- enclose argument in square brackets
  parse arg value
  return "["value"]"

::routine quote public -- enclose argument in double-quotes
  parse arg value
  return "'" || value || "'"
```

17



::ROUTINE Directive, Example "call_package.rex"



```
call toolpackage.rex -- get access to public routines in "toolpackage.rex"
say quote('hello, my beloved world')

r=" 1 "
s=2
say "r="pp(r)
say "s="pp(s)
say
say "r="quote(r)
say "s="quote(s)
say
say "The result of 'r || 3 ' is:" pp(r || 3 )
say "The result of 's || 3 ' is:" quote(s || 3 )
say "The result of 'r + 3' is:" pp(r + 3)
say "The result of 's + 3' is:" quote(s + 3)

/* yields:

"hello, my beloved world"
r=[ 1 ]
s=[2]

r=" 1 "
s="2"

The result of 'r || 3 ' is: [ 1 3]
The result of 's || 3 ' is: "23"
The result of 'r + 3' is: [4]
The result of 's + 3' is: "5"

*/
```

10

18



REXX Directive



- Syntax

`::requires "package.rex"`

– Interpreter in (setup) phase 2 will either

- Call (execute) the REXX program in the file named "package.rex" on behalf of the current REXX program and make all its public routines and classes upon return directly available to us
- Or if the interpreter already has *required* that "package.rex" it will *immediately* make all its public routines and classes available to us
 - In this case "package.rex" will *not* be called (executed) anymore!

19



REXX-Directive, Example "requires_package.rex"



```
say quote('hello, my beloved world')

r=" 1 "
s=2
say "r="pp(r)
say "s="pp(s)
say
say "r="quote(r)
say "s="quote(s)
say
say "The result of 'r || 3 ' is:" pp(r || 3 )
say "The result of 's || 3 ' is:" quote(s || 3 )
say "The result of 'r + 3' is:" pp(r + 3)
say "The result of 's + 3' is:" quote(s + 3)

::requires toolpackage.rex - get access to public routines in "toolpackage.rex"

/* yields:

"hello, my beloved world"
r=[ 1 ]
s=[2]

r=" 1 "
s="2"

The result of 'r || 3 ' is: [ 1 3]
The result of 's || 3 ' is: "23"
The result of 'r + 3' is: [4]
The result of 's + 3' is: "5"

*/
```

11

20



The Message Paradigm, 1



- A programmer sends messages to objects
 - The *object* looks for a method routine with the same name as the received message
 - If arguments were sent the *object* forwards them
 - The *object* returns any value the method routine returns
- C.f. <https://en.wikipedia.org/wiki/Alan_Kay>
 - One of the fathers of Smalltalk's "object-orientation"
- Programming languages with this paradigm, e.g.
 - Smalltalk, Objective C, ...

21



The Message Paradigm, 2 ooRexx



- Proper message operator "~" (tilde, "twiddle")
- In ooRexx everything is an "object"
 - Hence one can send messages to everything!
- Example

```
say "hi, Rexx!"~reverse
-- same as in classic REXX:
say reverse("hi, Rexx!")
-- both yield (actually run the same code):
!xxeR ,ih
```

12

22



▼ The Message Paradigm, 3 ooRexx



- Creating "values" a.k.a. "objects", "instances"
Classic Rexx-style (strings only)
`str="this is a string"`
ooRexx-style (any class/type including `.string` class)
`str=.string~new("this is a string")`

23



▼ About Classic REXX Structures, 1 Important Usage of Stems



- Whenever structures ("records") are needed, *stems* get used in classic REXX
- Example
 - A person may have a name and a salary, e.g.
`p.name = "Doe, John"`
`p.salary= "10500"`
 - E.g. a collection of data with a person structure
`p.1.name = "Doe, John"; p.1.salary=10500`
`p.2.name = "Doe, Mary"; p.2.salary=8500`
`p.0 = 2`

13

24



About Classic REXX Structures, 2 Important Usage of Stems



- Whenever *structures* ("records") need to be processed, every Rexx programmer *must* know the *exact stem encoding!*
- *Everyone* must implement routines like increasing the salary *exactly* like everyone else!
- If *structures* are simple and not used in many places, this is o.k., but the more complex the more places the *structure* needs to be accessed, the more error prone this becomes!

25



About ooREXX Structures, 1 Classes (Types, Structures)



- Any object-oriented language makes it easy to define and implement *structures!*
 - That is what they were designed for!
- The *structure* ("*class*", "*type*") usually consists of
 - *Attributes* (data elements like "*name*", "*salary*"), a.k.a. "*object variables*", "*fields*", ...
 - *Method* routines (like "*increaseSalary*")

14

26



▼ About ooREXX Structures, 2 Classes (Types, Structures)



- **::CLASS** Directive
 - Denotes the name of the *structure*
 - Can optionally be public
- **::ATTRIBUTE** Directive
 - Denotes the name of a *data element, field*
- **::METHOD** Directive
 - Denotes the name of a routine of the *structure*
 - Defines the *Rexx code* to be run, when invoked

27



▼ About ooREXX Structures, 3 Classes (Types, Structures)



- Once
 - A *structure* ("*class*", "*type*" both of which are synonyms of each other) got defined
 - One can create an *unlimited (!) number* of persons (or "*instances*", "*objects*", "*values*")
 - *Each person* will have its *own copy of attributes* (*data elements, fields*)
 - *All persons* will share/use the *same method routines* that got defined for the structure (class, type)

15

28





ooRexx Structure "Person" "personstructure.rex"



```
p=.person~new("Doe, John", 10500)
say "name: " p~name
say "salary:" p~salary
```

```
::class person           -- define the name

::attribute name        -- define a data element, field, object variable
::attribute salary     -- define a data element, field, object variable

::method  init         -- constructor method routine (to set the attribute values)
  expose name salary    -- establish direct access to attributes
  use arg name, salary  -- fetch and assign attribute values

/* yields:
   name:  Doe, John
   salary: 10500
*/
```

29



Defining the ooRexx Class (Type) "person.cls"



```
::class person PUBLIC  -- define the name, this time PUBLIC

::attribute name        -- define a data element, field, object variable
::attribute salary     -- define a data element, field, object variable

::method  init         -- constructor method routine (to set the attribute values)
  expose name salary    -- establish direct access to attributes
  use arg name, salary  -- fetch and assign attribute values
```

16

30





Defining the ooRexx Class (Type)



"requires_person.rex"

```
p.1 = .person~new("Doe, John", 10500)
p.2 = .person~new("Doe, Mary", 8500)
p.0 = 2
```

```
sum=0
do i=1 to p.0
  say p.i~name "earns:" p.i~salary
  sum=sum+p.i~salary
end
say
say "Sum of salaries:" sum
```

```
::requires person.cls -- get access to the public class "person" in "person.cls"
```

```
/* yields:
```

```
  Doe, John earns: 10500
  Doe, Mary earns: 8500
```

```
  Sum of salaries: 19000
*/
```

31



ooRexx Classes and Beyond ...



- ooRexx comes with a wealth of *classes*
 - A lot of tested functionality for "free" ;-)
 - E.g., the collection classes augment what stems are capable of doing!
 - Explore the collection classes and you will immediately be much more productive!
 - If seeking arrays, you have them: `.Array` class
 - Consult the pdf-books coming with ooRexx, e.g.,
 - "ooRexx Reference" ([rexref.pdf](#))
 - "ooRexx Programming Guide" ([rexpg.pdf](#))

17

32



Roundup



- ooRexx is great and compatible to classic REXX
 - You can continue to program in classic REXX, yet use ooRexx on Linux, MacOS, Windows, s390x...
- ooRexx adds a lot of flexibility and power to the REXX language and to your fingertips
 - One can take advantage of all of it immediately
 - Simple to use because of the *message paradigm*
 - Send ooRexx *messages* to Windows and MS Office ...
 - Send ooRexx *messages* to Java ...
 - Send ooRexx *messages* to ...
- **Get it and have fun! :-)**

33



Links



- RexxLA-Homepage (non-profit SIG, owner of ooRexx, BSF4ooRexx)
<<http://www.rexxla.org/>>
- ooRexx 5.0 on Sourceforge
<<https://sourceforge.net/projects/oorex/5.0.0/>>
 - Introduction to ooRexx on Windows, Slides ("Business Programming 1")
 - <<http://wi.wu.ac.at/rgf/wu/lehre/autowin/material/foils/>>
- BSF4ooRexx850 on Sourceforge (ooRexx-Java bridge)
<<https://sourceforge.net/projects/bsf4oorex/>>
 - Introduction to BSF4ooRexx (Windows, Mac, Unix), Slides ("Business Programming 2")
 - <<http://wi.wu.ac.at/rgf/wu/lehre/autojava/material/foils/>>
- Students' work, including ooRexx, BSF4ooRexx
<<http://wi.wu.ac.at/rgf/diplomarbeiten/>>
- JetBrains "IntelliJ IDEA", powerful IDE for all operating systems
 - <<https://www.jetbrains.com/idea/download/>>, free "Community-Edition"
 - Students and lecturers can use the professional edition for free
 - Alexander Seik's ooRexx-Plugin with readme (as of: 2023-05-09)
 - <<https://sourceforge.net/projects/bsf4oorex/files/Sandbox/aseik/ooRexxIDEA/GA/2.2.0/>>
- Introduction to ooRexx (254 pages, covers ooRexx 4.2)
<<https://www.facultas.at>>

18

34



Tutorial: Stems a Different Way – Rony G. Flatscher

Date and Time

3 Mar 2024, 05:00:00 CET

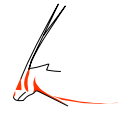
Presenter

Rony G. Flatscher

Presenter Details

Rony works as a professor for Business informatics (“Wirtschaftsinformatik”) at the Vienna University of Economics and Business Administration (Wirtschaftsuniversität Wien) and uses Open Object REXX for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooREXX, the ooREXX-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

Tutorial: Stems a Different Way - Introducing 'oo' in 'ooRexx'



The 2024 International Rexx Symposium
Brisbane, Queensland, Australia
March 3rd – March 6th 2024

© 2024 Rony G. Flatscher (Rony.Flatscher@wu.ac.at, <http://www.ronyRexx.net>)
Wirtschaftsuniversität Wien, Austria (<http://www.wu.ac.at>)



Overview



- Data type, abstract data type
 - REXX: strings, stem variables ("stems")
 - ooRexx in addition: Classes, Attributes, Methods
- Collecting values
 - REXX (and ooRexx): "Stem arrays"
 - ooRexx: *real* arrays
- Roundup



▼ Data Type (DT), 1



- Data type
 - Defines set of valid values
 - Defines operations with those values (e.g. addition, concatenation)
 - Example 1
 - Data type **Birthday**
 - Defined values consist of a combination of
 - A valid **date** attribute and a valid **time** attribute
 - Defined operations
 - Set, query and change its **date** and **time** attributes

3



▼ Data Type (DT), 2



- Example 2
 - Data type **Person**
 - Defined values consist of a combination of
 - **firstName**, **lastName**, **salary** attributes
 - Defined operations
 - Set, query and change its **firstName**, **lastName**, **salary** attributes
 - **increaseSalary**

21

4





▼ Data Type (DT), 3 REXX-Problems

- No means to *explicitly* define *data structures*
- No means to *explicitly* define *operations* restricted to certain data types
- *Data structures* can be mimicked with
 - Strings
 - Stem variables

5



▼ Data Type (DT), 4 REXX, Possible Solution, 1

- Encode a data structure in a string
 - E.g. for the data type **Birthday**
 - "2005-09-01 16:00"
 - "2008-02-29 19:19"
 - E.g. for the data type **Person**
 - "Albert Einstein 45000"
 - "Vera WithAnyName 25000"
- Processing possible *only* if *everyone* knows
 - Number and sequence of encoded fields/attributes
 - Where the fields/attributes start and end

22

6





Data Type (DT), 5

REXX, Possible Solution, 2

- Represent a data structure with a stem variable
 - E.g. for the data type **Birthday**

```
birthday.0date="2005-09-01"; birthday.0time="16:00"  
birthday.0date="2008-02-29"; birthday.0time="19:19"
```
 - E.g. using a "stem-array" for data type **Person**

```
person.1.firstname="Albert"; person.1.lastname="Einstein";  
person.1.salary="45000"  
person.2.firstname="Vera"; person.2.lastname="WithAnyName";  
person.2.salary=25000
```
- Processing possible if name of fields/attributes is known!

7



Data Type (DT), 6

REXX, Considerations

- DT-Structure
 - Encoding as strings or in stems
 - Crook, as implementation dependent!
 - Error-prone!
- DT-Operations
 - No means to define operations restricted to data types!
- No means to hide values/instances of data types from the programmer in order to shelter them from programming errors!
 - *Everyone* must know implementation (encoding) details!

23

8





Abstract Data Type (ADT), 1



- Abstract Data type (ADT)
 - *Schema* for implementing data types
 - Definition of *attributes*
 - Yields the data structure
 - Definition of *operations* ("methods")
 - Yields the *behaviour*
 - *Schema* must be implemented
 - REXX is *not designed* for it, hence not suitable!
 - ooRexx is an object-oriented language and *hence predestined ! :-)*

9



Abstract Data Type (ADT), 2



- Implement any ADT in ooRexx with *directives*
 - `::CLASS name`
 - `::ATTRIBUTE name`
 - `::METHOD name`
 - Hint: Rexx method routines are able to directly access attributes of its class by using as their first instruction the `EXPOSE` keyword instruction listing the attributes
- "Instances" ("objects", "values")
 - Distinct to any other instance/object/value
 - Possess all the same structure and behaviour

24

10





Abstract Data Type (ADT), 3 Implementing ADT "Birthday", 1

```
/* an ooRexx program that implements an ADT! */
```

```
::CLASS    Birthday      /* name of the structure/class      */
::ATTRIBUTE date
::ATTRIBUTE time
```

- Creating values/instances/objects
 - Simply send the message **NEW** to the Rexx-Class named **.Birthday**
 - Message operator is the tilde (~), hence e.g.

```
bd1=.Birthday~new /* create a value      */
bd2=.Birthday~new /* create another value */
...
```

11



Abstract Data Type (ADT), 4 Implementing ADT "Birthday", 2

```
/* an ooRexx program that implements an ADT! */
```

```
bd1=.Birthday~new
bd1~date="2005-09-01"
bd1~time ="16:00"
```

```
bd2=.Birthday~new
bd2~date="2008-02-29"
bd2~time ="19:19"
```

```
say "BirthDay 1:" bd1~date bd1~time
say "BirthDay 2:" bd2~date bd2~time
```

```
::CLASS    Birthday      /* name of the structure/class      */
::ATTRIBUTE date
::ATTRIBUTE time
```

Output:

```
BirthDay 1: 2005-09-01 16:00
BirthDay 2: 2008-02-29 19:19
```

25

12





Excursus: Scopes, 1 REXX

- Scopes
 - Determine the visibility of variables, attributes, routines and classes
- REXX-Scopes
 - *Standard-Scope*
 - Labels and variables are visible throughout the program
 - *Procedure-Scope*
 - Variables of internal routines followed by the **PROCEDURE** keyword statement are locally visible only

13



Excursus: Scopes, 2 ooRexx, 1

- Additional ooREXX-Scopes
 - *Program-Scope*
 - All **Routine**-directives and **Class**-directives of a program are visible in the entire program
 - In addition all public routines and public classes defined in another program become visible and directly accessible after that program got invoked/required !

26

14





Excursus: Scopes, 3 ooRexx, 2

- Additional ooREXX-Scopes
 - *Routine-Scope*
 - Managed as if it was a separate REXX-Programm
 - *Standard-Scope*
 - Can, therefore, include internal routines
 - *Procedure-Scope*
 - Can access all the routines and classes of the program (package) in which it gets defined
 - *Program-Scope*

15



Excursus: Scopes, 4 ooRexx, 3

- Additional ooREXX-Scopes
 - *Method-Scope*
 - Like *Routine-Scope*
 - In addition
 - Direct access to **attributes** of its class possible
 - First instruction must be the **EXPOSE**-keyword instruction with blank delimited attribute names

27

16





Excursus: Scopes, 5

Overview

- REXX and ooRexx
 - *Standard-scope*: labels, variables
 - *Procedure-scope*: local variables
- ooRexx
 - *Programm-scope*: routines, classes
 - *Routine-scope*
 - Like a separate program
 - Scopes: *Standard, Procedure, Program*
 - *Method-Scope*
 - Like *Routine-Scope*
 - Additionally: **EXPOSE** allows direct access to **attributes** of the class

17



Abstract Data Type (ADT), 5

Implementing ADT "Person", 1

```

p1=.person~new          /* create an instance/value/object */
p1~firstName ="Albert"
p1~lastName ="Einstein"
p1~salary   =45000

p2=.person~new          /* create an instance/value/object */
p2~firstName ="Vera"
p2~lastName ="WithAnyName"
p2~salary   =25000

say "Person 1:      " p1~firstName p1~lastName p1~salary
say "Person 2:      " p2~firstName p2~lastName p2~salary
say "sum of salaries:" p1~salary + p2~salary

::CLASS Person          /* name of the structure/class */
::ATTRIBUTE firstName
::ATTRIBUTE lastName
::ATTRIBUTE salary
  
```

Output:

```

Person 1:      Albert Einstein 45000
Person 2:      Vera WithAnyName 25000
sum of salaries: 70000
  
```

28

18





Abstract Data Type (ADT), 6

Implementing ADT "Person", 2

```

p1=.person~new          /* create an instance/value/object */
p1~firstName ="Albert"
p1~lastName="Einstein"
p1~salary  =45000

p2=.person~new          /* create an instance/value/object */
p2~firstName ="Vera"
p2~lastName="WithAnyName"
p2~salary  =25000

say "Person 1:      " p1~firstName p1~lastName p1~salary
say "Person 2:      " p2~firstName p2~lastName p2~salary
p1~increaseSalary(10000) /* increase salary */
say "Person 1:      ->" p1~firstName p1~lastName p1~salary
say "sum of salaries: ->" p1~salary + p2~salary

::CLASS Person          /* name of the structure/class */
::ATTRIBUTE firstName
::ATTRIBUTE lastName
::ATTRIBUTE salary

::METHOD increaseSalary /* increaseSalary method */
EXPOSE salary          /* access "salary" attribute directly */
USE ARG increaseBy     /* fetch increase amount */
salary=salary+increaseBy /* add and save result in attribute */

```

Output:

```

Person 1:      Albert Einstein 45000
Person 2:      Vera WithAnyName 25000
Person 1:      -> Albert Einstein 55000
sum of salaries: -> 80000

```

19



Fun with Methods: INIT, 1

Creating Objects/Instances/Values

- Objects/instances/values
 - Can be simply created by sending the message **NEW** to the class which will return a newly created object
- If a method **INIT** exists in the class then it will be invoked from the **NEW** method
 - If one supplies arguments to the **NEW**-message, then they will be forwarded to **INIT** in the same order!
 - The **INIT**-method carries also the name "constructor method" or short: "constructor"

