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
- 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**
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
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
• Represent a data structure with a stem variable
  - E.g. for the data type **Birthday**
    
    ```
    birthday.0.date = "2005-09-01"; birthday.0.time = "16:00"
    birthday.0.date = "2008-02-29"; birthday.0.time = "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!
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!
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*! :-)

REXX
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
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 */
    ...

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
Excurus: 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
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!
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
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
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
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
Abstract Data Type (ADT), 6
Implementing ADT "Person", 2

```rexx
::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
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"
Fun with Methods: INIT, 2
Creating Objects/Instances/Values

```rexx
p1=.person~new("Albert", "Einstein", 45000) /* create with values */
p2=.person~new("Vera", "WithAnyName", 25000) /* create with values */

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
```

```rexx
::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 */

::METHOD INIT /* constructor method */
    EXPOSE firstName lastName salary /* access attributes directly */
    USE ARG firstName, lastName, salary /* assign arguments to attributes */
```

Output:

Person 1: Albert Einstein 45000
Person 2: Vera WithAnyName 25000
sum of salaries: 70000
Fun with Methods: **UNINIT**, 1

Destroying Objects/Instances/Values

- **Objects/instances/values**
  - If values are not referenced anymore then the "garbage collector" destroys them

- If a method with the name **UNINIT** exists in a class, then the garbage collector will invoke it right before destroying the value
  - E.g. useful to release global locks, writing logs etc.
  - The **UNINIT**-method is also known as the "destructor method" or short: "destructor"
Fun with Methods: UNINIT, 2
Destroying Objects/Instances/Values

```
p1=.person~new("Albert", "Einstein", 45000) /* create with values */
p2=.person~new("Vera", "WithAnyName", 25000) /* create with values */
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
drop p2; drop p1 /* delete variables, objects become garbage */
call sysSleep 5 /* sleep five seconds */
say "end of main program!"
```

```
::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 */

::METHOD INIT /* constructor method */
EXPOSE firstName lastName salary /* access attributes directly */
USE ARG firstName, lastName, salary /* assign arguments to attributes */

::METHOD UNINIT /* destructor method */
EXPOSE firstName lastName salary /* access attributes directly */
say 'Object <'firstName lastName salary'> about to be destroyed...'
```

Output (maybe):
Person 1: Albert Einstein 45000
Person 2: Vera WithAnyName 25000
sum of salaries: 70000
end of main program!
Object <Vera WithAnyName 25000> about to be destroyed...
Object <Albert Einstein 45000> about to be destroyed...
Collecting Values, 1

• "Stem-arrays"
  - Convention
    • Stem variable with the tail "0" contains the number of stored values starting with the tail "1"
  - Only possibility in REXX to collect and to process values in an array like manner
  - ooRexx allows for collecting any kind of values in such stem arrays
Collecting Values, 2
"Stem-Arrays", 1

```rexx
person.1.firstName = "Albert"
person.1.lastName = "Einstein"
person.1.salary = 45000 /* <-- typical typing error! */

person.2.firstName = "Vera"
person.2.lastName = "WithAnyName"
person.2.salary = 25000

person.0 = 2

do i=1 to person.0 /* iterate over all persons */
    say "Person #" i":" person.i.firstName person.i.lastName person.i.salary
end
```

Output:

```
Person # 1: Albert Einstein
Person # 2: Vera WithAnyName 25000
```
Collecting Values, 3
"Stem-Arrays", 2

person.1 = .person~new("Albert", "Einstein", 45000)
person.2 = .person~new("Vera", "WithAnyName", 25000)
person.0 = 2

do i=1 to person.0 /* iterate over all persons */
    say "Person #" i:" person.i~firstName person.i~lastName person.i~salary
end

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

::METHOD INIT /* constructor method */
    EXPOSE firstName lastName salary /* access attributes directly */
    USE ARG firstName, lastName, salary /* assign arguments to attributes*/

Output:
Person # 1: Albert Einstein 45000
Person # 2: Vera WithAnyName 25000
Collecting Values, 4

ooRexx

• ooRexx has *real* arrays!
  - Simple to create
    • ooRexx 5.0 even allows creating them from a list
  - Easy to use and to iterate over the collection
    • E.g. DO...OVER

• Hint
  - ooRexx comes with many different kinds of classes/types that allow one to collect and process values!
Collecting Values, 5

ooRexx Has Real Arrays, 1

```rexx
persons=\.Array-new /* create an array */
persons[2]=.person~new("Vera", "WithAnyName", 25000)

do p over persons /* iterate over all persons */
say "Person:" p~firstName p~lastName p~salary
end

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

::METHOD INIT /* constructor method */
EXPOSE firstName lastName salary /* access attributes directly */
USE ARG firstName, lastName, salary /* assign arguments to attributes */
```

Output:

Person: Albert Einstein 45000
Person: Vera WithAnyName 25000
Collecting Values, 6

ooRexx

• Arrays can be sorted! :)  
  - Simply define a method named \texttt{compareTo}
  - Will receive the other value to compare to by the \texttt{sort}
    method defined in the \texttt{Array} class
  - Method must return the value
    - "1", if our value is regarded to be larger
    - "0", if both values are regarded to be the same
    - "-1", if other value is regarded to be larger
Collecting Values, 7
ooRexx Has Real Arrays, 2

```plaintext
persons=.Array~new  /* create an array */
persons[2]=.person~new("Vera", "WithAnyName", 25000)

do p over persons~sort /* iterate over all persons in sorted order */
say "Person:" p~firstName p~lastName p~salary
end
```

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

::METHOD INIT  /* constructor method */
  EXPOSE firstName lastName salary  /* access attributes directly */
  USE ARG firstName, lastName, salary  /* assign arguments to attributes*/

::METHOD compareTo  /* comparison method for sorting */
  EXPOSE salary  /* access attribute directly */
  use arg other  /* other person to compare to */
  if other~salary<salary then return 1  /* our salary is greater */
  if other~salary=salary then return 0  /* salaries are the same */
  return -1  /* other salary is greater */
```

Output:
Person: Vera WithAnyName 25000
Person: Albert Einstein 45000
Roundup

- REXX
  - Data structures can be hardly represented
  - Defining operations for data structures not possible

- ooRexx
  - Defining data structures incredibly easy
  - Defining operations for data structures: ditto! :)  
  - Very powerful and versatile
  - Values can be simply collected with the help of arrays and in addition can be easily sorted! 8-(
• RexxLA-Homepage (non-profit SIG, owner of ooRexx, BSF4ooRexx)
  <http://www.rexxla.org/>
• ooRexx 5.0 on Sourceforge
  <https://sourceforge.net/projects/oorexx/files/oorexx/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/bsf4oorexx/>
  - 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/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/GA/2.2.0/>
• Introduction to ooRexx (254 pages, covers ooRexx 4.2)
  <https://www.facultas.at>