

# An Introduction to Procedural and Object-oriented Programming (Object Rexx) 3

Exceptions, References, Directives  
(::routine, ::requires)

**Prof. Rony G. Flatscher**

# Classic Rexx

## Execution of Programs

- File containing the program gets loaded
- Usually 1<sup>st</sup> line, 1<sup>st</sup> column start of the string: /\*
- Thereafter line by line
  - Read statement
  - Check statement for syntactical errors
  - Execute (interpret) statement
- Lines, which are not visited are usually not checked for syntax errors!
  - I.e. in **IF** statements the **THEN**- or the **ELSE**-branch
  - Potential time bombs:
    - *Sometimes (maybe even after years!) a statement may be visited, which is syntactically wrong and therefore causes the program to be aborted ("all of a sudden")*

# Object Rexx and Regina (cf. <http://www.rexx.org>)

## Execution of Programs

- File containing the program gets loaded
- ***All lines*** are read
  - ***All statements*** are syntactically checked and translated into a compressed intermediary code ("tokenized image"), which later gets executed
    - ***No syntactic time bombs!***
- **::REQUIRES** directives are carried out
- Remaining directives (**::ROUTINE**, **::CLASS**, **::METHOD**) are carried out
- Program starts with the very first statement before the first directive
  - For modules the program (all statements before the first directive) can be used to initialise the modules themselves

# Exceptions

- Categories (Conditions)
  - **SYNTAX** Statement not syntactically correct
  - **FAILURE** Error in external program
  - **ERROR** Error in external program, not intercepted with "**FAILURE**" oder "**ANY**"
  - **HALT** Ctl-C (Ctl-Break): user aborts program
  - **NOVALUE** Using a non-initialised variable
  - **USER** User-defined exceptions
  - **LOSTDIGITS** Needs more digits than **NUMERIC DIGITS**
  - **NOMETHOD, NOSTRING, NOTREADY** (later ... )
  - **ANY** Intercepts (represents) *all* exceptions

# Exceptions

- Invoking the intended (programmed) exception handling statements with
  - **CALL {on|off} category [NAME label]**  
use a procedure to deal with the exception (from which one can return)
  - **SIGNAL {on|off} category [NAME label]**  
transfer control to the statements at the given label
- Intercepting ("catching") exceptions can be activated with the keyword **ON**, and deactivated with **OFF**
- One of the aforementioned categories, if using the user defined exception category **USER**, then it gets followed by the userdefined exception identifier
- **NAME** optional, allows for defining a label which serves as the **CALL** or **SIGNAL** target
  - If no explicit label is given, then the interpreter looks for a label which has the same name as the exception

# Exceptions

- **Hint:** Windows Workbench
  - It is not possible to use the category **ANY** for interception!
  - The Workbench intercepts all unhandled exceptions by using **ANY**
- All exceptions can only be intercepted in the scope of the calling program
  - Hence, the triggering of an exception with the **RAISE** statement is only interceptable in the caller
  - Exception: **SYNTAX**

# Dealing with Exceptions in a General Manner

- Generally dealing with exceptions
  - Copy the label and its code to the end of your programs
  - Activate the exception handling with the "SIGNAL ON" statement at the beginning of your program

```
SIGNAL ON ANY /* no label, hence "ANY" */
... Your 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
```

# Exceptions, Example 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
```

## Output:

**NIX**



# Exceptions, Example 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
```

## Output:

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

# Raising Exceptions

- Usually, the Rexx-Interpreter raises exceptions  
... but you can do it also
- **RAISE** statement
  - **RAISE** category
    - Creates ("raises") the given exception
  - **RAISE PROPAGATE**
    - Can only be given **during** exception handling
    - Re-creates the same exception in the caller, which allows the caller to also intercept it

# Raising Exceptions, Example 1

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

## Output:

```
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
```

# Raising Exceptions, Example 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
```

## Output:

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

# Raising Exceptions, Example 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
```

## Output:

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

# Variables (Rexx)

- Strings
- Stem-Variables, which allow storing strings
- Arguments for procedures/functions
  - **Only** strings allowed in classic Rexx, hence
  - No Stem-Variable allowed as an argument!
    - **EXPOSE** statement allows access to stem variables of the caller by breaking the (desired) insulation of the local scope (created with the **PROCEDURE** statement right after the label)

# Variables (Object Rexx)

- Variables are **References** to instances of Object Rexx classes
  - Strings
  - Stems
  - ... (more later ...)
- Arguments for procedures/functions
  - **PARSE ARG** statement
    - *Only* Strings allowed
    - No Stem-Variable !
    - **EXPOSE** statement allows access to a stem variable defined in the caller
  - **USE ARG** statement
    - *All* Objects are allowed as arguments

# Routines (Object Rexx)

- Routines are directives
  - Therefore they start with a double-colon (::)
  - Routines represent procedures and functions
    - There is no **EXPOSE** statement available to the routine
  - After a successful syntax check they are made available in the scope
    - of the program itself, and
    - in addition in all superordinate (calling) programs, *if* the keyword **PUBLIC** is given
  - Define their **own scope**, as if they were a program of their own!
    - Therefore labels are available *within* routines zum Aufrufen von Unterprogrammen und Funktionen daher möglich



# Routines (Object Rexx): 1a

```
/**/  
SAY pp("hello")  
CALL oha          /* routine is called */  
SAY pp("hello")  
  
EXIT 0  
pp : RETURN "<<<" || ARG(1) || ">>>"  
  
:: ROUTINE oha PUBLIC  
   SAY pp("holla")  
   EXIT 0  
   pp : RETURN "[" || ARG(1) || "]"
```

## Output:

```
<<<hello>>>  
[holla]  
<<<hello>>>
```

# Routines (Object Rexx): 1b

```
/**/  
SAY pp("hello")  
CALL oha          /* routine is called */  
SAY pp("hello")  
  
EXIT 0  
pp : RETURN "<<<" || ARG(1) || ">>>"
```

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

## Output:

```
<<<hello>>>  
[holla]  
<<<hello>>>
```

# Routines (Object Rexx): 1c

```
/**/  
SAY pp("hello")  
CALL oha          /* routine is called */  
SAY pp("hello")  
  
EXIT 0  
pp : RETURN "<<<" || ARG(1) || ">>>"
```

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

## Output:

```
<<<hello>>>  
[holla]  
<<<hello>>>
```



# Routines and Exceptions: 2

- Routines are like external procedures/functions

```
/**/  
CALL ON USER TOO_SMALL /* intercept a user exception */  
CALL checkAge 10  
CALL checkAge 3  
CALL checkAge 7  
EXIT 0  
  
TOO_SMALL: /* dealing with the user exception */  
  SAY "// caught exception 'TOO_SMALL' \\  
  RETURN  
  
::ROUTINE checkAge  
  PARSE ARG age  
  SAY "--> age:" age  
  IF age < 6 THEN RAISE USER too_small  
  ELSE SAY "--> checked o.k."  
  
EXIT 0
```

## Output:

```
--> age: 10  
--> checked o.k.  
--> age: 3  
// caught exception 'TOO_SMALL' \  
--> alter: 7  
--> checked o.k.
```

# Routines and Exceptions: 3a

```
CALL ON ANY          /* intercept anything that is not caught explicitly */
CALL ON USER TOO_SMALL /* intercept a user exception */
CALL ON USER too_big  /* intercept a user exception */
CALL checkAge 10
CALL checkAge 3
CALL checkAge 7
EXIT 0
```

```
ANY      : SAY "in line:" SIGL "exception:" CONDITION("C"); RETURN
Too_small: SAY "// caught exception 'TOO_SMALL' \\";      RETURN
TOO_BIG:  SAY "// caught exception 'TOO_BIG' \\";        RETURN
```

```
::ROUTINE checkAge
  PARSE ARG age
  SAY '--> age:' age
  IF age < 6 THEN RAISE USER too_small
                ELSE IF age > 9 THEN RAISE USER too_big
                ELSE SAY '--> checked o.k.'
  RAISE USER something_raised
  EXIT 0
```

## Output:

```
--> age: 10
// caught exception 'TOO_BIG' \
--> age: 3
// caught exception 'TOO_SMALL' \
--> age: 7
--> checked o.k.
in line: 7 exception: USER SOMETHING_RAISED
```

# Routines and Exceptions: 3b

```
CALL ON ANY          /* intercept anything that is not caught explicitly */
CALL ON USER TOO_SMALL /* intercept a user exception */
CALL ON USER too_big  /* intercept a user exception */
CALL checkAge 10
CALL checkAge 3
CALL checkAge 7
EXIT 0
```

```
ANY      : SAY "in line:" SIGL "exception:" CONDITION("C"); RETURN
Too_small: SAY "// caught exception 'TOO_SMALL' \\";          RETURN
TOO_BIG:  SAY "// caught exception 'TOO_BIG' \\";            RETURN

::ROUTINE checkAge
  PARSE ARG age
  SAY '--> age:' age
  IF age < 6 THEN RAISE USER too_small
                  ELSE IF age > 9 THEN RAISE USER too_big
                  ELSE SAY '--> checked o.k.'
  RAISE USER something_raised
  EXIT 0
```

## Output:

```
--> age: 10
// caught exception 'TOO_BIG' \
--> age: 3
// caught exception 'TOO_SMALL' \
--> age: 7
--> checked o.k.
in line: 7 exception: USER SOMETHING_RAISED
```

# Requires Directive (Object Rexx)

- **::Requires** directive
  - Allows naming a Rexx program
    - Hint: for porting purposes, enclose the filename in quotes (Unix is case sensitive)
  - The interpreter will call the required program before carrying out any of the other directives (**::Routine**, **::Class**, **::Method**)
  - Thereafter all of its *public* routines (and *public* classes!) are made available



# CALL-Statement and Public Routines: 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-Statement and Public Routines: 2/2

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

```
/* 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"
```

## 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
```

```
/* cmd2.rex */  
SAY " /1/ In" pp("cmd2.rex")  
CALL cmd3  
SAY " /2/ In" pp("cmd2.rex")  
EXIT 0  
  
pp :  
RETURN "c2[" || ARG(1) || "]c2"
```

```
/* cmd4.rex */  
SAY " In" pp("cmd4.rex")  
EXIT 0  
  
pp :  
RETURN "c4<" || ARG(1) || ">c4"
```

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

# Requires-Directive and Public Routines

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

```
::REQUIRES cmd2.rex
```

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

```
:: requires cmd4.rex
```

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

**Ausgabe:**

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

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

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

```
::Requires cmd3.rex
```

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

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

```
::ROUTINE pp PUBLIC
```

```
RETURN "c4<<" || ARG(1) || ">>c4"
```