0% DropOuts, From 0 To GUI Programming in 1 Semester

- After 1st month (after four weeks)
  - ooRexx: Loops, routines, functions, classes (attributes, methods), multiple inheritance, commands with redirection & piping, JSON, curl, interacting with LLMs
- After 2nd month (after 7th week)
  - ooRexx: Windows OLE/COM, VBS/VBA (Visual Basic Script/Application), programming wsh (Windows shell), MSIE (Internet Explorer) and DOM (document object model), MS Office (Word, Excel), ADSI (active directory service interfaces), WMI (Windows management instrumentation) et.al.
- After 3rd month (after 11th week)
  - ooRexx-Java bridge: GUI (java.awt, java.swing), Internet programming with sockets (java.net, javax.net, javax.net.ssl) either OpenOffice/LibreOffice or ML (machine learning employing Weka), XML parsing (SAX, DOM with recursion) and XSLT
- After 4th month (after 14th week)
  - ooRexx-Java bridge: External Java class libraries, CLASSPATH, modular Java, Java startup options, Jsoup, Java scripting framework (javax.script), RexxScriptEngine, JavaFX, NetRexx, JDOR (Java2D ooRexx command language)

Rony G. Flatscher

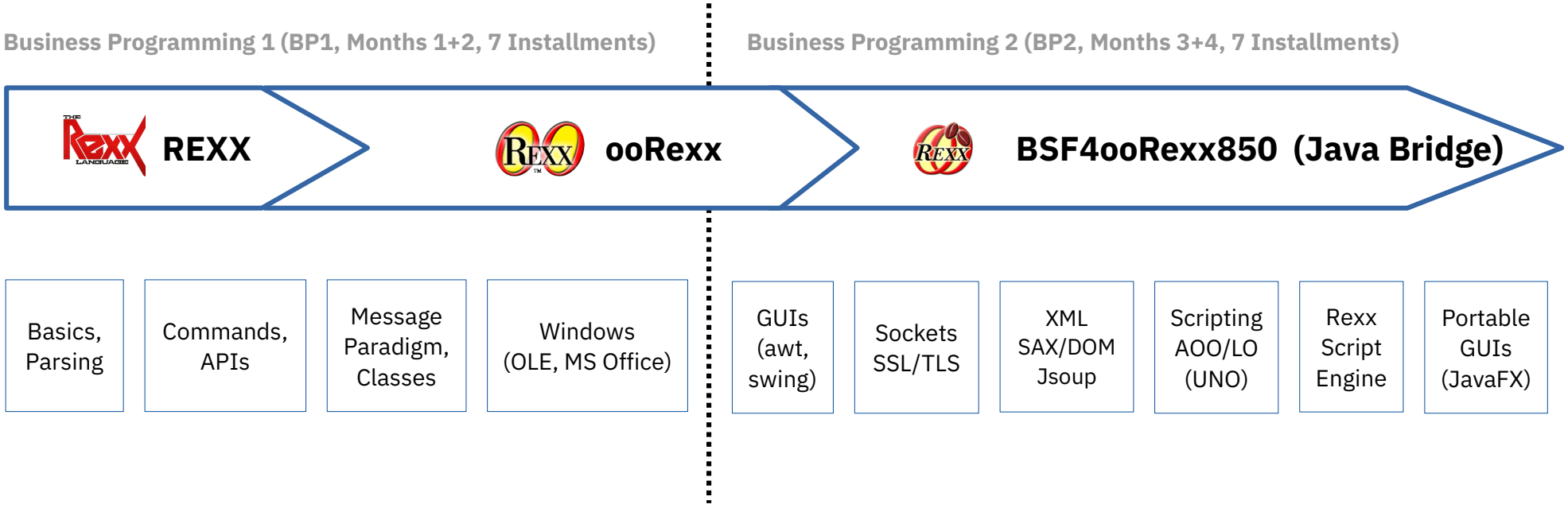
Department of Information Systems and Operations Management

EAPROG - Erfolgsbedingungen für Anfänger zum Erlernen der Programmierung portabler GUIs in einem Semester

(Critical Factors from Zero to Portable GUI Programming in Four Hours)

SE25-Workshop (Karlsruhe, 2026-02-23)

Rony G. Flatscher & Till Winkler



Overview

- **Part 1**: Background, Goals, Cognitive Load Theory, Critical Success Factors
- **Part 2**: Critical Success Factor "Programming Language"

  Rexx/ooRexx: Overview, Concepts and Nutshell Examples
- **Part 3**: Critical Lectures

  1st Installment: Onboarding

  3rd Installment: Messaging and Object-Orientation

  5th Installment: COM/OLE for Windows Applications

  8th Installment: Java and ooRexx Java Bindings
- **Part 4**: Hands-on: Installations, Running Nutshell Examples
- **Workshop Roundup, Links, Addendum**

Rony G. Flatscher

Part 1

Background

Goals

Cognitive Load Theory

Critical Success Factors

"Business Programming" – What Students Learn

- Teach novices in a single semester (four months, 4h per week)
  - **First half of the semester** (two months)
    - Object-oriented programming
    - Programming of Windows OLE applications including MS Office
  - **Second half of the semester** (two months)
    - Programming using Java classes and interacting with Java objects
      - Possible because Java gets camouflaged
      - Platform independence all programs run *unchanged* on Windows, macOS and Linux
    - Learn to creat GUIs (awt/swing, JavaFX), learn client/server socket programming, learn to process XML and HTML files, interface with OpenOffice/LibreOffice
- Total teaching load 8 ECTS points (total of 200 hours)

WU (Business Administration University)

- Located in Vienna, Austria
  - WU (acronym from "*W*irtschafts*u*niversität")
  - Founded 1898 as a "World Trade High School" ("Hochschule für Welthandel")
- More than 20,000 students
  - One of the largest universities of its kind
- Department of Information Systems (IS) and Operations Management
  - One of eleven departments at WU
  - Currently eight institutes, in alphabetic order
    - "Complex Networks", "Data, Energy, and Sustainability", "Data, Process, and Knowledge Management", "Digital Economy", "Distributed Ledgers and Token Economy", "Information Management and Control", "Information Systems and Society", "Production Management"

Evolution of "Business Programming"

- **Original challenge:** "is it possible to teach interested novice BA students programming in a single semester such that the students become able to program MS Office?"
- **More than 35 years of evolution** (appr. 120 lectures – two each semester)
  - Each lecture's installment got systematically analyzed
  - Observing and analyzing students' problems in understanding taught concepts
  - Constantly reworking focus areas, slides, nutshell examples accordingly
  - Experimenting with various programming languages (VBA, VBS, Java, REXX/ooRexx)
- **Current status** (2025)
  - BA students learn in a *four hour lecture (8 ECTS points)* in a *single semester (four months)*
    - Fundamentals of object-oriented programming
    - Windows and MS Office, OpenOffice/LibreOffice programming via COM/OLE
    - Platform independent programming via Java: GUIs, client/server, OpenOffice/LibreOffice, ...
- **Key success factors:** Programming language ooRexx

What to Learn and to Apply, 1

- Statement, comment, symbol, variable, block, comparison, branch, loop
- Routine, label, scope, function, associative arrays, commands
- Exception, handler, routine and requires directive, arguments by reference
- OO: Abstract datatype (ADT), class/type, attribute, method, creating objects/instances/values, message
- Class hierarchy, inheritance, collection classes and iteration
- Windows: COM, OLE, Windows registry, ooRexx class OLEObject to camouflage Windows, MS Excel, MS Word, OpenOffice/LibreOffice as ooRexx
- Fundamentals of HTML and XML, Instrumentating InternetExplorer (sic!) via OLE, and cURL using redirected commands from ooRexx

What to Learn and to Apply, 2

- Introduction to Java and the ooRexx Java bridge (BSF4ooRexx850)
- Thanks to employing Java all programs run on Windows, macOS and Linux
  - GUI concepts with events (and callbacks), Socket programming (client/server)
  - OpenOffice/LibreOffice: UNO architecture, swriter, scalc, simpress
  - XML: concepts, using SAX (callbacks) and DOM to parse XML text files
  - HTML: concepts, applying the Jsoup class library
  - Java scripting framework: BSF4ooRexx850' RexxScriptEngine (allows ooRexx to be used as a Java scripting language in all Java applications)
  - JavaFX: concepts, creating most complex GUIs in an easy manner

Some Challenges

- There are **many different concepts to learn** and to apply
- Students should **master the course and *not* drop out**!
- Everything should be taught and learned in a *single* semester only

How can this reliably be achieved?

What has to be taken into account for teaching all of this, how do humans learn?

In a Nutshell, 1 (Sweller, 1988; Garner, 2002; Sweller & Van Merriënboer, 2005; Paas et al. 2003)

- Knowledge is stored in **long-term memory** as schemata
  - A schema is treated as a single element by our brain, but can itself be made up of several elements
- Learning means constructing a new (more complex) schema by combining existing schemata with the help of **working memory**
  - The goal of teaching is to enable the construction of increasingly complex schemata and to facilitate their automation through practice (Paas et al., 2003)
  - **Learning requires active involvement of the working memory**
    - Working memory can only deal in parallel with a **limited number of elements/schemata**

In a Nutshell, 2 (Sweller, 1988; Garner, 2002; Sweller & Van Merriënboer, 2005; Paas et al. 2003)

- Three cognitive load types competing for the limited working memory
  - **Intrinsic cognitive load:** the complexity of the subject to learn
  - **Extraneous cognitive load:** the context of learning, e.g. the burden induced by bad teaching techniques (e.g. complicated explanations) or laborious research needs
  - **Germane cognitive load:** learning through thinking about new information and concepts
- The more working memory capacity is available (low intrinsic/external cognitive load), the faster learning takes place (high germane cognitive load)

CSF # 1: Use an *easy to learn* programming language!

- Saves precious lecture time which is better used for explaining and digesting, understanding, applying fundamental programming concepts
- Experimented with various languages (e.g. Pascal, BASIC, VBS/VBA, Java, Rexx/ooRexx), *surprisingly* Rexx was *the most efficient language for novices*
- **ooRexx** (acronym for "open object Rexx")
  - Students learned it fastest, the saved time can be used to teach additional content
  - Human-centric design
    - Easy syntax, reads almost like pseudo code
    - Incorporates object-oriented concepts to "play with"
  - Interpreter, can be used interactively (rexxtry.rex, trace)
  - Developed originally by IBM handed over to the non-profit special interest group RexxLA.org
    - Professional and powerful programming language
  - Open-source and free for all major platforms (Windows, Linux, macOS)

CSF # 2: Pareto principle

- Impossible to teach everything in detail in an *introductory* course, therefore
  - Teach conceptual, overview knowledge
  - Select the most "important concepts"
    - E.g., object-oriented paradigm, COM/OLE on Windows and MS Office, Java interface to be able to create portable (Windows, Linux, MacOS), GUI, Internet (socket) programs, OpenOffice/LibreOffice, parsing XML and HTML text
- **Pareto principle:** "teach 80% of the most important concepts in 20% of the time"
  - Rather than targeting 100% which would impose an additional 80% of time, which is not available
  - ***If the students become curious they will research on their own!***

CSF # 3: Humboldt's ideal

- Observe the students
  - What do they understand immediately, what questions do they ask?
  - What problems do they get and why (complex concept or missed classes)?
- If necessary
  - Create new paths to ease understanding
  - Rework or add new slides, remove complex slides and improve nutshell examples
  - Retest the new/updated slides and nutshell examples
- Allows to gradually improve the course and its materials over time

CSF # 4: No student is left alone

- Create groups of two students (pair programming)
  - Inhibits drop-outs
  - Enables direct help
- Mix students' skills if possible at all
  - A skilled student becomes "buddy tutor"
  - for a Zero-skilled student in that group

CSF # 5: Searching the Internet

- Modern programming is about searching the Internet!
- Find one own's coding problems and possible solutions
  - Follow links to explanations and tutorials for the problem at hand
- Find additional learning resources in all media forms on the Internet
  - E.g., tutorials for concepts that are not yet understood, Youtube-videos for demonstrating the handling of development tools, AI-supported research and explanations of concepts

CSF # 6: Nutshell examples

- Make it as easy as possible to learn programming
- Use easy to understand, small ("nutshell") programs
  - *As short as possible*
  - Demonstrate a single concept, if possible at all
  - Allow for experimenting with the code and by doing so experimenting with the concept
- Show the output of nutshell programs on the slides
  - "Seeing is better than believing"

CSF # 7: Weekly coding assignments

- Create two short (!) programs together in the group
  - Students become able to help each other
  - Novices can usually handle short assignments and are normally also able to understand short programs from other groups on their own
- Weekly assignments must be shared with all students
  - Allows studying other students' programs
  - Stimulus for programming ideas

CSF # 8: Concluding project assignment

- Students suggest three projects combining with ooRexx
  - Three Windows programs ("Business Programming 1")
    - After two months, at the end of the first half of the semester
  - Three Java jar class libraries ("Business Programming 2")
    - After another two months, at the end of the (second half of the) semester where
      - JRE (Java runtime environment) counts as a proper jar class library
      - In addition JavaFX counts as a proper jar class library
- One project will be picked and needs to be implemented within a week
  - Project gets presented and demonstrated
  - Students experience success
  - Students realize the skills and knowledge they have acquired in the course!

Part 2

CSF # 1: "Programming Language"

Rexx/ooRexx:

Overview, Concepts and Nutshell Examples

Developing Business Programming

- Specialisation in **"(Business) Information Systems"**
  - As customary at the time, the *most popular languages* were used to teach beginners: Pascal, BASIC, COBOL, C, PROLOG, Visual Basic Script (VBS) / Applications (VBA), Java, ...
- **Surprise** when experimenting with the Rexx programming language
  - Novices learn **much faster and more in-depth** than with popular languages
  - Analysing the **critical success** factors showed that the most important aspect was **the programming language**
- 35 years of **participant observation** (two lectures per semester)
  - Observed difficulties yielded changes in: content, slides, nutshell examples, infrastructure, presentation, ...

Some Historical Bits on Rexx

- Created for IBM mainframes to make programming easier compared to the rather awkward EXEC2
  - **Rexx design goals:** "human centric", "keep the language small", "easy to learn", "easy to understand hence easy to maintain"
  - **Rexx** is **still instrumental for IBM mainframe operating systems** today!

- Extremely successful in the 80'ies
  - Companies selling Rexx interpreters successfully, **ANSI/INCITS standard** (!)
- Object-oriented successor ("Object Rexx") in the 90'ies by IBM
  - **Open-sourced** in 2005 by RexxLA.org – "open object Rexx" (ooRexx)
    - Available for **all major operating systems**
    - Possible to program even MS Windows applications via OLE ...

Fundamental Rexx Concepts, 1

- "Everything is a string"
  - If a string represents a number, one can carry out arithmetic
- Three instruction types
  1) Assignment
     - Variable name followed by the assignment operator (=) and an expression
  2) Keyword instruction
     - Keywords are English words conveying the intent of the keyword instruction, e.g. SAY, DO, IF, LOOP, CALL, PARSE, SELECT, ITERATE, LEAVE, INTERPRET, ...
     - Makes Rexx code legible as if it was pseudocode
  3) Commands
     - A string passed to the operating system for execution (as if typed in a window)

Fundamental Rexx Concepts, 2

- White space can be freely used to format code for better legibility
  - Space around operators gets removed
  - White space between symbols will be reduced to a single space serving as concatenation operator
  - Hence indentations (for better legibility) with white space not significant
- Case of symbols *irrelevant*
  - Rexx uppercases everything outside of quoted strings
  - No (frustrating) casing errors for novices
- **Rexx** nutshell examples to stress fundamental concepts
  - Illustrate the language
  - Same examples in the popular **Python** language to allow direct comparisons

Instructions

```
    /* an assignment instruction:      */
a="hello world"   /* assigns "hello world" to a variable named a  */

    /* a keyword instruction:          */
say a             /* output: hello world */

    /* a command instruction:          */
    /* a command (could be typed into a command line window)          */
"echo Hello World 2" /* execute command                               */
   /* variable RC contains the command's return code, 0 means success */
if rc=0 then say "success!"
        else say "some problem occurred, rc="rc /* show return code   */
```

Output:

```
hello world
Hello World 2
Success!
```

```
# an assignment instruction
a="hello world"     # assigns "hello world" to a variable named a

# no keyword instruction for output, using built-in function print()
print(a)

# no command instruction using module subprocess instead
import subprocess   # import subprocess module
# execute command
completedProcess=subprocess.run("echo Hello World 2", shell=True) # run
rc=completedProcess.returncode  # fetch return code, an int
if rc==0:
    print("success!") # indentation mandatory (forcing a block)
else: # must use + (concatenation operator) with str() function
    print("some problem occurred, rc="+str(rc)) # turn rc into a string
```

Output:

```
hello world
Hello World 2
Success!
```

Blocks, Selection, Multiple Selections

```
max=5              /* number of repetitions   */
loop a=1 to max    /* loop block              */
   select          /* nested block # 1        */
      when a=1 then say a": first round"
      when a=2 then say a": second round"
      when a=3 then say a": third round"
      otherwise say "(a="a")"
   end

   if a=max then
   do              /* nested block # 2        */
      say "-> a=max"
      say "-> last round!"
      say "-> loop will end"
   end
end
```

Output:

```
1: first round
2: second round
3: third round
(a=4)
(a=5)
-> a=max
-> last round!
-> loop will end
```

```
max=5       # number of repetitions
for a in range(1,max+1): # loop with range() function, must add 1 to max
            # must use str() function with + (concatenation operator)
    match a:    # must be indented, "match" needs Python 3.10 or higher
        case 1: print(str(a)+": first round")   # nested block # 1
        case 2: print(str(a)+": second round")  # nested block # 1
        case 3: print(str(a)+": third round")   # nested block # 1
        case _: print("(a="+str(a)+")")  # default, nested block # 1

    if a==max:  # must be indented, must use == instead of =
        print("-> a==max")          # nested block # 2
        print("-> last round!")     # nested block # 2
        print("-> loop will end")   # nested block # 2
```

Output:

```
1: first round
2: second round
3: third round
(a=4)
(a=5)
-> a==max
-> last round!
-> loop will end
```

Parsing Strings

```
text = " John     Doe   Vienna Austria"
parse var text firstName lastName city country
say "first name:" firstName", last name:" lastName", city:" city
```

```
text = "Mary Doe Tokyo Japan"
parse var text firstName lastName city .   /* ignore country */
say "first name:" firstName", last name:" lastName", city:" city
```

Output:

```
first name: John, last name: Doe, city: Vienna
first name: Mary, last name: Doe, city: Tokyo
```

```
text      = " John     Doe   Vienna Austria"
words     = text.split()    # create list of words
firstName = words[0]        # assign to variable
lastName  = words[1]        # assign to variable
city      = words[2]        # assign to variable
print("first name:",firstName+",","last name:",lastName+",","city:",city)
```

```
text  = "Mary Doe Tokyo Japan"
words = text.split()    # create list of words
# assign multiple elements in a single statement
firstName, lastName, city = [words[i] for i in (0, 1, 2)]
print("first name:",firstName+",","last name:",lastName+",","city:",city)
```

Output:

```
first name: John, last name: Doe, city: Vienna
first name: Mary, last name: Doe, city: Tokyo
```

Some Thoughts ...

- Popular languages are ***not*** the best choice for teaching ***novices*** programming!
- **Rexx**' *human centric design* allows novices to learn programming *much faster*
  - **Intrinsic load *much lower***
    - Less syntax rules
    - Keywords imply intent (intuitive, looks almost like pseudo code)
    - Very easy to instrumentalize one own's computer with commands
  - **Lower cognitive load is a critical success factor** for teaching *novices* programming with almost no drop-outs
- The learned programming concepts can be applied to any other programming language (**Java**, **Python**, ...) quickly
  - In effect, *additional languages can be learned in a fraction of the time!*

Fundamental ooRexx Concepts, 1

I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea. The big idea is "messaging". – Alan Kay (https://en.wikipedia.org/wiki/Alan_Kay)

- **ooRexx** has been influenced by **SmallTalk** including its ***message*** *paradigm*
- **ooRexx** adds *message expressions* and *directive instructions* to Rexx

Fundamental ooRexx Concepts, 2

- *"Everything is an object (synonyms: value, instance)"*
  - An object is *conceptually regarded as if it was a living thing*
  - One can only interact with an object by sending it *messages*
- A *message expression* consists of a *receiver*, the message operator **~** (tilde) and the *message name*, optionally followed by arguments in parentheses
  - The *receiver* will search a *method* by the *name* of the received message, invokes it and returns any result to the sender
  - No one can invoke methods directly but the *receiver* (encapsulation)
  - As a result the *sender* does not need to know anything about implementation details

Messages

```
say reverse("olleh")     -- classic Rexx BIF (built-in function)
say "olleh"~reverse      -- message to string object
```

Output:

```
hello
hello
```

```
a="dlrowolleh"     -- assign string to variable
   -- use built-in-functions (BIFs) reverse(), substr()
say substr(reverse(a),1,5) substr(reverse(a),6)

   -- use String methods reverse and substr
say a~reverse~substr(1,5) a~reverse~substr(6)
```

Output:

```
hello world
hello world
```

Concepts Added by ooRexx

- *Directive* instruction
  - If present then always placed at the end of a program
  - Led in by two consecutive colons (**::**) serving as an eye catcher
    - Directives that cause **ooRexx** to create classes with attributes and methods during the setup phase

      ::CLASS name, ::ATTRIBUTE name, ::METHOD name, …
- Classes with attributes and methods
  - Can be defined with directive instructions or dynamically at runtime
  - Instances get created by sending the class the message **new**
    - The **new** method will create the object and before returning it, the newly created object gets the message **init** sent with the arguments supplied to the **new** message, if any
      - Hence, defining *a method named **init** will always run at construction time* (constructor)

Creating A Class with Directives and Dynamically

```
say ".dog:" .dog     -- string value of the class
d=.dog~new           -- create and assign a dog
d~bark               -- let the dog bark
say "d:" d", an instance of:" d~class

::class dog          -- class directive
::method bark        -- method routine directive
  say "wuff!"        -- code to run
```

Output:

```
.dog: The DOG class
wuff!
d: a DOG, an instance of: The DOG class
```

Not for novices,
just for this audience! :)

```
clz=.object~subclass("DOG")   -- create the dog class
say "clz:" clz -- string value of the class
m  =.method~new("bark", 'say "wuff!"') -- create method
clz~define("bark",m) -- define as instance method for class

d=clz~new      -- create and assign a dog
d~bark         -- let the dog bark
say "d:" d", an instance of:" d~class
```

Output:

```
clz: The DOG class
wuff!
d: a DOG, an instance of: The DOG class
```


Rony G. Flatscher

Ad Messages, 1

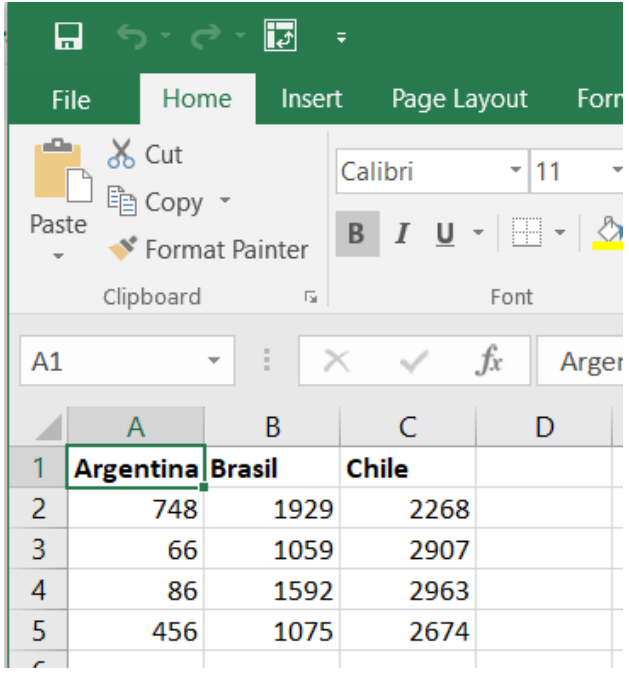
- Quickly familiar, intuitive for novices
- Seeing **objects as living things** makes it easy to accept behaviours and concepts like
  - The **new** method of a class will send the **init** message to the newly created object (a method named init is therefore a constructor)
  - An object using the *class hierarchy* to locate the method to invoke (inheritance)
  - *Multiple inheritance* (!) deviating the search carried out by the object
  - Intercepting messages for which no method could be found as the object then sends the **unknown** message to itself (simply implement a method **unknown**)
  - The variables **self** (reference to the object that invoked the method) and **super** (reference to the immediate superclass) in methods
  - As objects know how to find and invoke methods, the sender does not need to know that (black box) at all, alleviating the (novice) programmer

Ad Messages, 2

- Addressing complex software infrastructures can be made easy for message senders (programmers)
  - Create a proxy class in **ooRexx** and process the received messages, marshall the arguments and unmarshal the return value
- Example Windows and Windows programs
  - **ooRexx** for Windows has **ooRexx** classes for Windows support
  - The **OLEObject** class is the proxy class for interacting via OLE (Object Linking and Embedding) with any OLE Windows component
    - Its **unknown** method will intercept all messages for which no method can be found on the **ooRexx** side, such that it gets forwarded to the proxied Windows object by searching and invoking the appropriate Windows method
    - To exploit this functionality no implementation knowledge of COM or OLE is needed!

Programming Excel Using ooRexx Messages

```
excApp = .OLEObject~new("Excel.Application") -- create Excel object
excApp~visible = .true           -- make Excel visible
sheet  = excApp~Workbooks~Add~Worksheets[1]  -- add and get sheet
    -- set titles from an ooRexx array
titleRange=sheet~range("A1:C1")  -- get title cell range
titleRange~value     = .array~of("Argentina", "Brasil", "Chile")
titleRange~font~bold = .true     -- make font bold
sheet~range("A2:C5")~value = createRows(4)   -- create and assign array
excApp~displayAlerts = .false    -- no alerts (should file exist already)
fileName=directory()"\test.xlsx" -- save in current directory
Say 'fileName:' fileName         -- show fully qualified file name
sheet~SaveAs(fileName)           -- save file (no alerts, see above)
excApp~quit                      -- quit (end) Excel

::routine createRows    -- return two-dimensional array with random data
  use arg items         -- fetch argument
  arr=.array~new        -- create Rexx array
  do i=1 to items       -- create random(min,max) numbers
     arr[i,1] = random(   0,1000)   -- Argentina
     arr[i,2] = random(1001,2000)   -- Brazil
     arr[i,3] = random(2001,3000)   -- Chile
  end
  return arr            -- return two-dimensional Rexx array
```



Possible Output:

```
fileName: C:\Program Files\JetBrains\IntelliJ IDEA\jbr\bin\test.xlsx
```

Ad Messages, 3

- Addressing complex software infrastructures can be made easy for message senders (programmers)
  - Create a proxy class in **ooRexx** and process the received messages, marshall the arguments and unmarshal the return value
- Example **Java** and **Java** class libraries
  - BSF4ooRexx850 for Windows, macOs and Linux implements an ooRexx-Java bridge
  - Its **BSF** class is the ooRexx proxy class for interacting with **Java**
    - Its **unknown** method will intercept all messages for which no method can be found on the **ooRexx** side, such that it gets forwarded to the proxied **Java** object by searching and invoking the appropriate **Java** method
    - To exploit this functionality no implementation knowledge of BSF4ooRexx850 is needed!

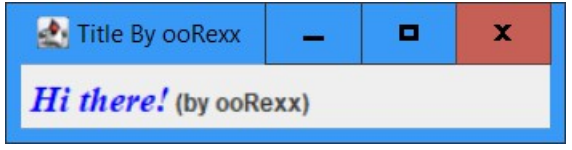
Communicating with Java Objects Using ooRexx Messages

```
dim=.bsf~new("java.awt.Dimension",111,222)
say "dim:          " dim", dim~class:" dim~class
say "dim~toString:" dim~toString -- Java method
    -- use Java fields as if ooRexx attributes
say "dim~width:    " dim~width    -- Java field
say "dim~height:   " dim~height   -- Java field
dim~setSize(333,444) -- Java method
say "dim~toString:" dim~toString -- Java method
    -- use Java fields as if ooRexx attributes
dim~width=555         -- setting Java field
dim~height=666        -- setting Java field
say "dim~toString:" dim~toString -- Java method

::requires "BSF.CLS" -- get ooRexx-Java bridge
```

Output:

```
dim:          java.awt.Dimension@1c4af82c, dim~class: The BSF class
dim~toString: java.awt.Dimension[width=111,height=222]
dim~width:    111
dim~height:   222
dim~toString: java.awt.Dimension[width=333,height=444]
dim~toString: java.awt.Dimension[width=555,height=666]
```

```
jf  = .bsf~new("javax.swing.JFrame", "Title By ooRexx")  -- create JFrame
style   = 'style="color: blue; font-family: serif; font-size: 18;"'
lblText = '<html><em' style'> Hi there!</em> (by ooRexx) </html>'
lbl = .bsf~new("javax.swing.JLabel", lblText)   -- create JLabel
jf~add(lbl)              -- add JLabel to JFrame
jf~setSize(280,70)       -- set size
jf~setLocation(50,200)   -- set JFrame's location on screen
jf~visible=.true         -- make JFrame visible
jf~toFront               -- place JFrame in front of all windows
say 'Hit <enter> on the keyboard to proceed (end) ...'
parse pull data          -- wait until user presses <enter>


::requires "BSF.CLS"     -- get ooRexx-Java bridge
```



Output:

```
Hit <enter> on the keyboard to proceed (end) ...
```

Some Thoughts ...

- Message paradigm
  - ***Easy* and *intuitive for novices!***
  - All important object-oriented concepts can be informally (!) explained and understood by novices
- **Proxy classes** allow **the message paradigm to be extended to other software systems**
  - Windows COM/OLE, proxy class OLEObject (supplied by **ooRexx**)
  - **Java**, proxy class **BSF** (supplied by BSF4ooRexx850)
  - ***Novice students do not care and are not afraid! :-)***
    - They "only" send messages and need not know any implementation details!
    - The supplied nutshell examples allow novices to exploit OLE and Java
      - Windows: MS Excel, MS Word, MS PowerPoint, AOO swriter, LO scalc, ...
      - **Java**: from (secure!) socket programming to JavaFX GUIs!

Part 3
Critical Lectures

Overview of the Slides

1st Installment: Onboarding, Fundamental Programming Concepts

3rd Installment: Messaging and Object-Orientation

5th Installment: COM/OLE for Windows Applications

8th Installment: **Java** and **ooRexx** Java Bindings

Slides and Critical Installments

- Slides and their nutshell examples are made freely available
  - See link section in the back of this presentation
- Led-in number in filename suggests the sequence position in the course
- Filename includes the version of the slides or nutshell zip archives
- Example filenames
  - If it starts with "010_ooRexx" then
    - "010_ooRexx_V11.**odp**" ... Apache OpenOffice (AOO a.k.a. OOo) presentation file
      - **odp** ("**o**pen **d**ocumeent **p**resentation") files are standardized and can be usually processed by other presentation programs like LibreOffice, PowerPoint , Keynote and the like
      - AOO is open source , cf. https://**O**pen**O**ffice.**o**rg which also supplies download links
    - "010_ooRexx_V11.**pdf**" ... PDF version of the slides
    - "010_ooRexx**_code**_V11.**zip**" ... zip archive of the nutshell examples in the slides
- Critical installments are highlighted in red in the addendum

1st Installment: Onboarding, 1

- Goals
  - Make the students comfortable, assure they can manage and complete the course
- Introduction round, overview and organization of the course, 90'
  - Introduction round, each student tells
    - Name, prior school, study program at University, any programming experiences (if so which languages, which skills), why in this course, what does the student expect from the course
      - Students will see that there are novices and maybe experts, novices will see they are not alone
  - Encourage "stupid questions"
    - There are no stupid questions, those who ask concentrate on the answer
  - Pair programming: no one is on her/his own
    - If possible mix a novice with a skilled student who takes on a tutor role

1st Installment: Onboarding, 2

- Introduction round, overview and organization of the course (continued)
  - Homework assignment
    - Two programs, ***as short (!) as possible***, applying newly learned concepts
    - Send in the two programs via a shared mailing list, such that everyone can see each others' code (and potentially rehearse the concepts by studying the code of others)
  - Stressing that it is important to help each other and *to ask actively for help!*
  - Concluding project after two months
    - Students come up with three project ideas (can be funny!), one gets picked and assigned
    - Students will program the project, create a presentation and demo the program
- Important *to start slowly* to introduce the students to **010_ooRexx**, 90'
  - Take time for explaining each slide!
  - Ask whether there are questions about each slide before going on!
    - Take time to answer any questions!

3rd Installment: Messaging and Object-Orientation, 1

- Overwhelming!
  - **040_ooRexx**, 90'
    - Confront the students with the most important OO terminology and concepts, no details
  - **050_ooRexx**, 90': *repeats* **040_ooRexx**, adds details, allows students to digest
- Message paradigm
  - *Easy to understand for novices!*
  - *Difficult* for people *who know to program already* and have never been exposed to it
  - Allows to conceptually picture objects as living things with which programmers interact by sending messages and receiving answers if any returned
    - The objects are conceptually responsible for looking up and invoking the methods named after the received message thereby abstracting the resolution process
    - The programmer does not need to know about any complexities a message may induce

3rd Installment: Messaging and Object-Orientation, 2

- Message paradigm (continued)
  - Explicit message operator **~** (tilde), receiving object is always on the left, message name on the right, optionally with arguments in parentheses
  - *Message chaining*: result (answer) of a message becomes receiver of a new message
  - *Message cascading* (always returns the object that receives the message)
    - Receiver of a message gets returned such that the next message is aimed at the same object
    - A difficult concept at first, students are relieved when they learn that one can forgo them
      - Students can accept that this concept is taught for "pedagogical reasons"
    - Message cascading, once understood, can considerably simplify certain coding needs

3rd Installment: Messaging and Object-Orientation, 3

- Object-orientation
  - Terminology *confounding* at first!
    - Synonyms "object"="instance"="value", "class"="type"="structure"
      - Intentionally speak out all three synonyms in this lecture to accustom the students
      - Note: these terms get informally defined
    - Homonym "object"
      - Generic term for an instance of a class/type/structure or denotes the root class named "Object"?
  - Class hierarchy, inheritance
    - Nutshell "Animal SIG": distracts by modelling and implementing normal dogs, little and big dogs
  - Multithreading: expose students to the concept, then tell them to forget about it ;)
    - Inter object multithreading
    - Intra object multithreading

5th Installment: COM/OLE for Windows Applications, 1

- **110_AutoWin**, 90'
  - Important to understand *conceptually*
    - COM ("**c**omponent **o**bject **m**odel") and OLE ("**o**bject **l**inking and **e**mbedding")
      - OLE: standardized way of interacting with methods, attributes, events, constants
      - Windows programs can communicate via COM/OLE with other Windows programs
    - The Windows registry
      - Central database organized in hives, each COM Windows program stored in this registry
      - Windows can consult the registry to find and run COM classes on behalf of ourselves
  - **ooRexx** proxy class OLEObject
    - Can be used to instantiate or fetch any OLE Windows program
    - Messages to OLEObject instances get forwarded to Windows via OLE *if unknown*
      - Published methods, attributes, events and constants can be queried via **ooRexx**
      - Marshalling and unmarshalling done transparently, no need to know any details!

5th Installment: COM/OLE for Windows Applications, 2

- **110_AutoWin** (continued)
  - Explain important **ooRexx** OLE nutshell samples (ooRexx/samples/ole)
    - Empowers the students to study the remaining nutshells in their groups
    - Empowers students to create programs that interact with MS Office and AOO/LO
    - MS Internet Explorer (MSIE)
      - Deprecated by Microsoft, yet in Windows 10 available via OLE
      - "Spectacular" for students to be able to "remote control" MSIE and navigate
    - MS Excel
      - Explain conceptually the MS Excel model and why it is important to make MS Excel visible
      - Explain all details of the MS Excel nutshell such that students understand all of it
    - Nutshell using Windows OLE program with the ProgID "Wscript.Network"
      - Explain **0**-based **C** which shines through this sample (**ooRexx** is **1**-based)
      - Explain programming technique to use a single dimensioned array to represent data that would be better represented in a two dimensional array

8th Installment: Java and ooRexx Java Bindings, 1

- **210_AutoJava_BSF4ooRexx**, 180' (entire installment)
  - Nutshell example "java.awt.Dimension" to demonstrate how easy it is to use **Java** classes and objects from **ooRexx**
  - Important to understand *conceptually* **Java**
    - Stress differences to ooRexx: static language, strictly typed, case sensitive, compiled
    - Stress primitive types (boolean, byte, char, short, int, long, float, double)
      - Boxing to and unboxing from Java wrapper classes like e.g. java.lang.Boolean, ...
      - Values of primitive types can be represented as simple strings in Rexx
    - Stress access rights public, private, protected and package private (no access modifier)
      - BSF4ooRexx850 allows acess only to public classes, fields and methods and to inherited protected fields and methods
    - Stress platform independence, i.e. compiled **Java** classes do not need to be recompiled

8th Installment: Java and ooRexx Java Bindings, 2

- **210_AutoJava_BSF4ooRexx** (continued)
  - Stress "javadoc" making it possible to find all **Java** documentation on the Internet
    - Find any **Java** documentation on the Internet and look it up with any browser: easy and fast to find and complete documentation with links to related documentation
  - Make explicitly clear that
    - **Java** classes correspond to **ooRexx** classes
    - **Java** fields correspond to **ooRexx** attributes (object variables)
    - **Java** methods correspond to **ooRexx** methods
  - Introduce the **ooRexx-Java** bridge BSF4ooRexx850, explain its name
    - **BSF**: "**B**ean **S**cripting **F**ramework" (**Java** scripting framework from Apache/ASF)
    - **8**: **Java** version 8 or later
    - **50**: **ooRexx** version 5.0 or later
    - Developed for more than 25 years by Rony G. Flatscher

8th Installment: Java and ooRexx Java Bindings, 3

- **210_AutoJava_BSF4ooRexx** (continued)
  - The **ooRexx** proxy class BSF is defined in the *package* BSF.CLS (an **ooRexx** program)
    - Requiring BSF.CLS makes all its public classes and public routines available
      - E.g the public proxy class BSF or the public routines box() and unbox() for primitive Java types
    - The proxy class BSF camouflages **Java** objects as **ooRexx** objects
      - Camouflaged **Java** objects are therefore able to process plain **ooRexx** messages
        - BSF forwards unknown messages to the **ooRexx-Java** bridge in which the corresponding **Java** method gets looked up, any arguments marshalled, the **Java** method invoked and its return value unmarshalled and returned to **ooRexx**
      - The case of messages does not need to match the **Java** case of field or method names, the **ooRexx-Java** bridge will resolve any case mismatches transparently

8th Installment: Java and ooRexx Java Bindings, 4

- **210_AutoJava_BSF4ooRexx** (continued)
  - **Java** arrays are strictly typed, have a predefined size and have `0`-based indices
    - Returned **Java** arrays get automatically camouflaged as **ooRexx** arrays by the bridge
    - BSF.CLS includes public routines to ease the creation of **Java** arrays directly from **ooRexx** and automatically camouflages them as **ooRexx** arrays
  - BSF4ooRexx850 camouflages **Java** arrays as **ooRexx** arrays hence
    - 1-based indices as if they were an **ooRexx** array
    - **ooRexx** array methods like makeArray, supplier, at, put are available
      - Among other things this support allows for iterating over **Java** arrays with do ... over !

Part 4

Hands-on: Installations, Running Nutshell Examples

ooRexx 5 or Higher

Java 8 or Higher (***with*** *JavaFX!*)

ooRexx Java Bindings (BSF4ooRexx850)

Nutshell Examples

Installation, 1

- General remarks ad Windows and macOS
  - Files downloaded from the Internet get flagged as dangerous
  - Unzipping zip archives using Windows or MacOS supplied tools will flag all extracted files as well
  - If executables are signed then they will still execute, unsigned binaries will not
    - Signing costs money (on a yearly basis) and many open-source projects can not afford it
- Allowing open-source binaries to run ("de-quarantize")
  - Windows
    - Right mouse click to get the properties of the downloaded file, click "unblock" and "apply"
    - Or in a command prompt issue: `powershell Unblock-File filename`
  - MacOS
    - In a terminal window issue: `xattr -d com.apple.quarantine filename`

Installation, 2

- **ooRexx**
  - URL (as of 2026-02-10)
    - Recommended: https://sourceforge.net/projects/oorexx/files/oorexx/5.2.0beta/
    - Released version: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0/
    - "portable" subdirectory : contains portable versions that can be used without installation
    - Installation package: system wide installation, needs administrator rights to install
  - macOS
    - There is a package that installs both, **ooRexx** and BSF4ooRexx850, see BSF4ooRexx850
      - Needs **Java** already installed because of the contained **ooRexx-Java** bridge

Installation, 3

- **Java**/**OpenJDK**
  - **Java** name rights with Oracle , **OpenJDK** *same as* **Java** but by others and OSS license
    - **OpenJDK Java** e.g. from Amazon, IBM, Microsoft, SAP, and many more ...
    - Make sure you install the package with the **JavaFX** modules, e.g.
      - https://bell-sw.com/pages/downloads/ choose "Package Type" and set "*Full* JRE" or "*Full* JDK"
      - https://www.azul.com/downloads/?package=jre-fx#zulu choose "Java Package" and set "JRE *FX*" or "JDK *FX*"
  - Notes on Version 8 (LTS, **l**ong **t**erm **s**upport)
    - Last non-modular **Java** version (released 2014), supported at least until March 2031!
    - Runs **Java** programs that exploit **Java** internals which may be prohibited in modular Java
  - Notes on modular versions of **OpenJDK Java**
    - Bi-annually a new version, LTS versions are "long-term support" and used by businesses
    - Continuous development, rolled out much earlier (for testing) than non-modular versions

Installation, 4

- BSF4ooRexx850
  - Prerequisite: **ooRexx** 5+ and **Java** 8+ need to be installed/available via PATH
  - URL (as of 2026-02-10)
    - https://sourceforge.net/projects/bsf4oorexx/files/GA/BSF4ooRexx-850.20240304-GA/
      - Release, no reported open bugs!
    - Installation package: system wide installation, needs administrator rights to install
      - Change into bsf4oorexx/install/{windows|linux} and run install.{cmd|sh}
    - Portable: de-quarantize, unzip, change into "bsf4oorexx/install", run "rexx setupBSF.rex" use resulting shell scripts
  - macOS
    - There is a universal package that installs both, **ooRexx** and BSF4ooRexx850
    - URL (as of 2026-02-10)
      - https://sourceforge.net/projects/bsf4oorexx/files/GA/BSF4ooRexx-850.20240304-GA/
      - De-quarantize, unzip and run installer

Nutshell Examples, 1

- ooRexx (as of 2026-02-10), selection
  - oorexx/samples: demonstrate important ooRexx capabilities
    - oorexx/samples/0ReadMe.first: brief overview of samples directories
    - oorexx/samples/api: C and C++ samples to demonstrate writing libraries
    - oorexx/samples/misc: a drop file handler sample
    - oorexx/samples/oodialog: a set of samples of the ooDialog GUI framework for ooRexx
    - oorexx/samples/ole: OLE (Object Linking and Embedding) samples
    - oorexx/samples/ole/adsi: Active directory service samples (managing Windows)
    - oorexx/samples/ole/apps: Windows Shell, MS Office, OpenOffice/LibreOffice samples
    - oorexx/samples/ole/wmi: Windows management instrumentation (managing Windows)
    - oorexx/samples/ole/methinfo: Windows GUI to inspect OLE Windows methods

Nutshell Examples, 2

- BSFooRexx850 (as of 2026-02-10), selection
  - BSF4ooRexx850/samples: demonstrate important BSF4ooRexx850 capabilities
    - Hint: open the index.html file, it briefly documents each sample in the directory and allows for changing into subdirectories that have index.html files for the same purpose!
    - BSF4ooRexx850/samples/clr: .Net/CLR samples, needs Windows and Java 8
    - BSF4ooRexx850/samples/{DOM|SAX} samples processing XML files
    - BSF4ooRexx850/samples/{Java|NetRexx}: Java/NetRexx samples to demonstrate the Java scripting framework, implementing Rexx *exit handlers* in Java/NetRexx
    - BSF4ooRexx850/samples/OOo: numerous Apache OpenOffice (LibreOffice) samples
    - BSF4ooRexx850/samples/LeePeedin: samples demonstrating swing and dialog related GUI functionalities, including formatting
    - BSF4ooRexx850/samples/ReneJansen: samples demonstrating XSLT and JDBC (MySQL/MariaDB, Apache Derby, HyperSQL, PostgreSQL, SQLite, H2)

Workshop Roundup

- "Business Programming": introducing novices to programming and its application
  - Four weekly contact hours for one semester (four months)
  - 8 ECTS points, total net teaching load of 200 hours
  - *Novices* get empowered by being able to learn programming quickly
  - *Novices* get empowered by learning and applying the concepts on Windows with OLE (e.g., MS Office programming) and portable (exploiting the **ooRexx Java** bindings) client/server programming, GUI-programming, XML SAX & DOM parsing, web scraping (cURL, Jsoup), ...
- *Critical success factor "programming language": **ooRexx***
  - ***Popular languages** are usually **not adequate to teach novices** programming!*
  - **ooRexx**: easy to learn programming and much more, especially suited for novices!
  - Learning and exploiting **Java** directly becomes possible in a fraction of the time usually needed
- *All materials (slides, software for all platforms) are freely available*
- Last but not least: ***no dropouts** (!)* despite the overwhelming learning outcomes!

Links (As of 2026-02-10), 1

- **WU (English)**: https://www.wu.ac.at/en/the-university/about-wu/facts-figures/studierende/
  - **Business Programming 1 (BP1)**: first half of semester (two months)
    - Syllabus (German use e.g. Google translate, deepl.com) 2025/26: http://wi.wu.ac.at/rgf/wu/lehre/autowin/2025wBP1/BP1-autowin-2025w-uebersicht.pdf
    - Slides (English): https://wi.wu.ac.at/rgf/wu/lehre/autowin/material/foils/
  - **Business Programming 2 (BP2)**: second half of semester (two months)
    - Syllabus (German use e.g. Google translate, deepl.com) 2025/26: https://wi.wu.ac.at/rgf/wu/lehre/autojava/2025wBP2/BP2-autojava-2025w-uebersicht.pdf
    - Slides (English): https://wi.wu.ac.at/rgf/wu/lehre/autojava/material/foils/
  - **Some seminar papers, Bachelor and Master theses with ooRexx, BSF4ooRexx**: https://wi.wu.ac.at/rgf/diplomarbeiten/
- **Software**
  - **ooRexx** 5.1: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/
  - **Java/OpenJDK *with* JavaFX** modules, e.g. https://www.azul.com/downloads/?package=jdk-fx#zulu
  - **BSF4ooRexx850**: https://sourceforge.net/projects/bsf4oorexx/files/GA/BSF4ooRexx-850.20240304-GA/
- Hock-Chuan, Chua: *"Java Game Programming: 2D Graphics, Java2D and Images"*; *AffineTransformDemo*: https://www3.ntu.edu.sg/home/ehchua/programming/java/J8b_Game_2DGraphics.html#zz-2.2
- REXX history (initial specification): https://speleotrove.com/rexxhist/REXinitspec-1979.pdf

Links (As of 2026-02-10), 2

- **JavaFX SceneBuilder:** https://www.jetbrains.com/idea/download/
  - Interactive JavaFX GUI editor (create and edit FXML GUI definitions): https://gluonhq.com/products/scene-builder/
- **JetBrain's IntelliJ**: https://www.jetbrains.com/idea/download/
  - Community edition for free, education license for additional professional tools
  - ooRexx plugin and directions: https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/GA/
- **RexxLA:** https://www.rexxla.org/
  - Non-profit interest group developing and maintaining open-source Rexx related software and standards
  - US based, but members from all over the world
  - Organizes yearly international Rexx symposium: https://www.rexxla.org/events/
  - Members encompass creators and maintainers of various Rexx software, including the creator of Rexx, Mike F. Cowlishaw
    - Membership free: https://www.rexxla.org/members/index.rsp?action=join

Some Literature: ooRexx

- Flatscher, R. G. (2024). Introduction to Rexx and ooRexx. RexxLA, ISBN 9789403 739298 (glossy white paper, preferred) or ISBN 9789403 755038 (regular white paper).
- Flatscher, R. G., & Müller, G. (2021). "Business Programming" – Critical Factors from Zero to Portable GUI Programming in Four Hours. In Marko Kolakovic, Tin Horvatinovic, Ivan Turcic (Ed.), 6th Business and Entrepreneurial Economics 2021 - Conference Proceedings (pp. 76-82): https://research.wu.ac.at/files/32933925/2021_BusinessProgramming_BEE2021_accordingToGuidelines.pdf
- Flatscher, R. G. (2023). Proposing ooRexx and BSF4ooRexx for Teaching Programming and Fundamental Programming Concepts. In 2023 Program Guide ISECON: Information Systems Education Conference (pp. 89-102): https://research.wu.ac.at/files/41301564/ISECON23_Flatscher_Proposing_ooRexx_article.pdf
- Winkler, T., & Flatscher, R. G. (2023). Cognitive Load in Programming Education: Easing the Burden on Beginners with REXX. in Central European Conference on Information and Intelligent Systems (S. 171-178). Faculty of Organization and Informatics. https://research.wu.ac.at/files/46150789/CECIIS_CLT_REXX.pdf
- Flatscher, R. G., & Winkler, T. (2024). Concepts that Allow Learning the Programming Language Rexx Much Faster than Other Languages. Accepted paper for MIPRO 2024, 47th Convention, engineering education track. https://research.wu.ac.at/files/64505014/mipro24_9192_Flatscher_Winkler_LearningProgrammingFast_final-5.pdf
- Flatscher, R. G., & Winkler, T. (2024). Employing the Message Paradigm to Ease Learning Object-oriented Concepts and Programming. Accepted paper for MIPRO 2024, 47th Convention, engineering education track. https://research.wu.ac.at/files/64505159/mipro24_9194_Flatscher_Winkler_EmployingMessageParadigm_final-4.pdf

Literature: Cognitive Load Theory

- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive science, 12(2), 257-285.
- Sweller, J., & Van Merriënboer, J. J. G. (2005). Cognitive load theory and complex learning: Recent developments and future directions. Educational Psychology Review, 53(3), 147-177
- Garner, S. (2002). Reducing the cognitive load on novice programmers (pp. 578-583). Association for the Advancement of Computing in Education (AACE).
- Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. Educational psychologist, 38(1), 1-4.

Overview of BP1-Slides, 1

- **010_ooRexx**, *installment 1*, 90': history, symbols, comparisons, blocks, loops, commands
- **020_ooRexx**, installment 2, 90': labels, internal routines, functions, search order, scopes, associative arrays, parsing strings, parsing keyboard input, parsing arguments
- **030_ooRexx**, installment 2, 90': exceptions (conditions, includes a brief lecture on stdin, stdout, stderr and redirection), references, directives (::routine, ::requires)
- **040_ooRexx**, *installment 3*, 90': abstract datatype (ADT), classes, methods, attributes, messages, class hierarchy, inheritance, inter and intra object multithreading
- **050_ooRexx**, *installment 3*, 90': *repetition* of abstract datatype (ADT), classes, methods, attributes, messages, class hierarchy, inheritance, add details compared to *040_ooRexx*
- **051_ooRexx**, installment 4, 90': ordered and unordered collections, iterating over collections
- **340_JSON**, installment 4, 15': JSON, json.cls, reading and creating JSON data
- **060_ooRexx_commands**, installment 4, 30': process, environment variables, standard files (stdin, stdout, stderr), redirection, pipes, commands, curl
- **350_LLM_curl**, installment 4, 45': generative AI's, interface with with chatgpt, grok

Overview of BP1-Slides, 2

- **110_AutoWin**, *installment 5*, 90': COM, OLE, proxy class OLEObject, explaining key nutshell programs coming with the Windows version of **ooRexx** (%ProgramFiles%\ooRexx\samples\ole), like MS Excel, AD (active directory services), WMI (windows management instrumentation) and more
- **120_AutoWin_markup**, installment 5, 90': introduction to HTML, XML (learned concepts will be reused in BP2's SAX, DOM and Jsoup lectures!), nutshell sample for MS InternetExplorer (still accessible via OLE), allows retrieving and analyzing text from webservers
- Installment 6: students present project ideas, then the following slides
  - **130_AutoWin_oleinfo**, 30' : utility to query all registered COM classes and to generate HTML documentations of the published OLE interfaces of any Windows OLE program
  - **140_AutoWin_vba**, 30': overview of **VBA**, **VBA** macros, how to map **VBA** to **ooRexx** and vice versa
  - **060_ooRexx_commands**, 60': on processes, standard files, redirection of Rexx commands, **cURL**
  - **070_ooRexx_trace_debug**, 30': optional, turn on (off) trace to learn how statements execute exactly and how one can debug interactively at program runtime
  - **080_ooRexx_environment_symbols**, 15': optional, **ooRexx** runtime environment and resolution of environment symbols
- Installment 7: presentation and demonstration of each assigned student project

Overview of BP1-Slides, 3

- **210_AutoJava_BSF4ooRexx**, *installment 8*, 180': overview of **Java**, static language, strictly typed, case sensitive, qualified and unqualified class names, **Java** arrays (strictly typed, fixed size), mapping of classes/methods/fields to ooRexx classes/methods/attributes, JavaDocs on the Internet; **ooRexx** external function package BSF4ooRexx850, ooRexx program BSF.CLS defining the proxy class BSF for camouflaging **Java** objects as ooRexx, BsfCreateRexxProxy() function to create a Java RexxProxy (a Java object) to allow interaction from **Java** with embedded **ooRexx** objects
  - ***Exploiting Java has the effect that all ooRexx programs run unchanged on Windows, macOS and Linux!***
- **220_AutoJava_gui**, instalment 9, 90': introduction to GUIs, event thread, events (**Java** event callbacks to ooRexx), awt nutshell sample
- **230_AutoJava_Sockets**, installment 9, 90': switchboard and sockets, IP addresses, client/server, java.net.Socket, java.net.ServerSocket, data encrypted client/server with SSL/TLS (javax.net.ssl)
- **240_AutoJava_AOO_LO**, installment 10, 180': AOO/LO via their **Java** APIs, history, programming model, UNO framework, UNO classes, nutshells for the modules swriter, scalc, simpress
  – or –
  350_Weka_ML, installment 10, 180': machine learning (ML), concepts, Weka structure, Weka GUI, Weka commandline, Weka Java APIs, nutshell examples for supervised and unsupervised learning, plotting

Overview of BP1-Slides, 4

- **250_AutoJava_XML_SAX**, installment 11, 90': SAX programming model, callbacks to ooRexx, nutshells that extract text, element names, element hierarchy
- **252_AutoJava_XML_DOM**, installment 11, 90': DOM programming model, walk trees recursively, nutshells that extract text, element names, element hierarchy; in addition xhtml and xslt
- **254_AutoJava_jsoup**, installment 11, 30': Jsoup programming model, nutshells
- **260_AutoJava_RexxScript**, installment 12, 90': **Java** scripting framework (javax.script), features, application, RexxScript scripting engine implementaion, nutshells
- **270_AutoJava_JavaFX**, installment 12, 90': history, concepts, SceneBuilder, FXML and exploiting Java scripting framework, nutshells
- Installment 13: students present project ideas, then the following slides
  - **280_AutoJava_Environment**, 30': **Java** environment, CLASSPATH, **Java** modules
  - **320_Codepages**, 30': ASCII, 8-bit codepages (Windows cp1252), Unicode, UTF-8, nutshells
  - **330_Paths**, 15': source, current home, temporary directory, environment variables, java.lang.System
  - **340_JSON**, 30': concepts, nutshells
- Installment 14: presentation and demonstration of each assigned student project

Rony G. Flatscher