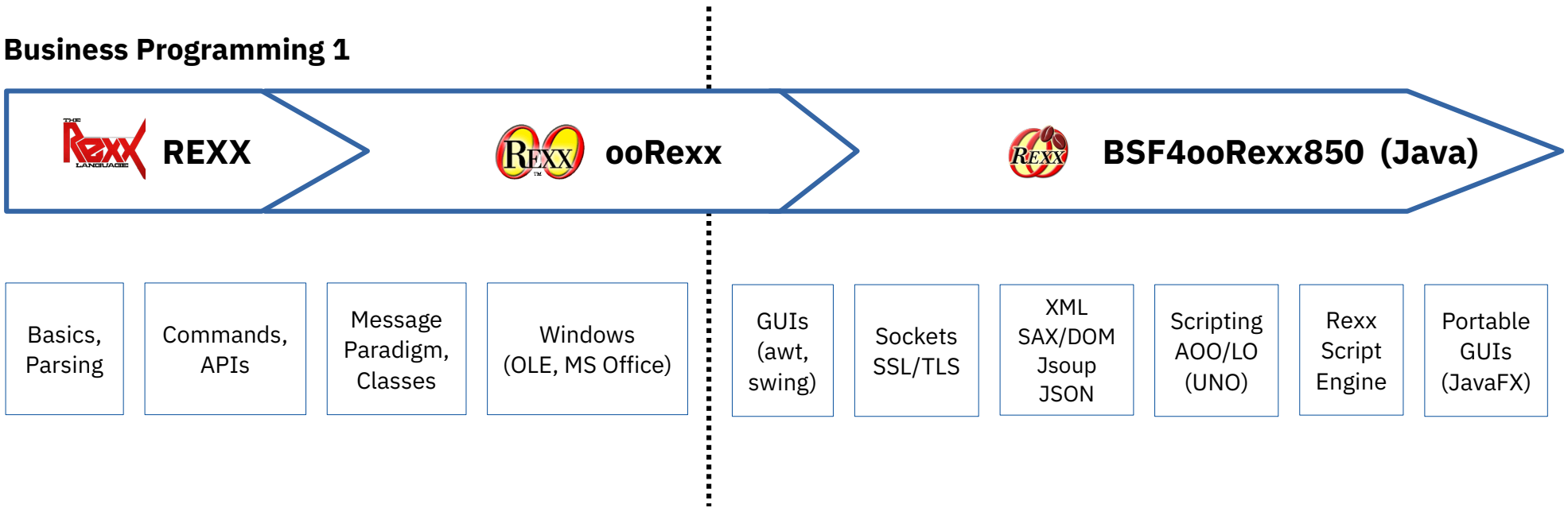
Department of Information Systems and Operations Management

"Business Programming"

Critical Factors from Zero to Portable GUI Programming in Four Hours

Workshop (CSEE & T 2023, Tokyo)



Rony G. Flatscher

Overview

- **Part 1**: overview, goals, organization of the course, critical success factors
- **Part 2**: critical success factor "programming language"
  - Rexx/ooRexx: concepts and nutshell example
  - BSF4ooRexx850: concepts and nutshell examples
- **Part 3**: critical lectures that determine the overall success of the course
- **Part 4**: hands-on installations, running nutshell examples
- Roundup
  - Links
  - Addendum (Rexx vis-à-vis Python)

Part 1
Overview
Goals
Critical Success Factors

Part 1

"Business Programming" – What Students Learn

- Teach novices in four hours in a single semester (four months)
  - First half of the semester (first two months)
    - Object-oriented programming
    - Programming of Windows OLE applications including MS Office
  - Second half of the semester (next two months)
    - Programming using Java classes and interacting with Java objects
      - Possible because Java gets camouflaged as the programming language used in the first half
    - Thanks to Java's platform independence all programs run *unchanged* on Windows, macOS and Linux
    - Using Java to learn creating 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)

Part 1

Background: WU (Business Administration University)

- Located in Vienna, Austria
  - WU (acronym from "*W*irtschafts*u*niversität")
  - Founded 1898 as a "World Trade High School" (celebrating 125 years in 2023)
- Appr. 20,000 students
  - One of the largest universities of its kind
  - Appr. 15,000 Bachelor, 4,200 Master, and 800 Doctoral/PhD
- Information Systems (IS) Department
  - One of eleven departments at WU
  - Currently seven institutes, in alphabetic order
    - "Data, Process and Knowledge Management", "Digital Ecosystems", "Distributed Ledgers and Token Economy", "Information Management and Control", "Information Systems and New Media", "Information Systems and Society", "Production Management"

Part 1

Background: Evolution of "Business Programming"

- Personal challenge of more than 35 years
  - 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?"
  - Evolved over appr. 120 lectures (two lectures 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)
  - As of 2023
    - 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 programming via COM/OLE
      - Platform independent programming via Java: GUIs, client/server, OpenOffice/LibreOffice, ...
  - Key success factors: programming language ooRexx and ooRexx-Java bridge BSF4ooRexx850

Part 1

Goals (First Semester Half, "Business Programming 1")

- Statement, comment, symbol, variable, block, comparison, branch, loop
- Routine, label, scope, function, associative arrays
- 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, and MS Word as ooRexx
- Windows: MS InternetExplorer (DHTML, fundamentals of HTML)

Part 1

Goals (Second Semester Half, "Business Programming 2")

- Introduction to Java and BSF4ooRexx850 (camouflages Java as ooRexx)
- 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

Part 1
Critical Success Factors (CSF)

- CSF # 1: *Use an easy to learn programming language!*
  - Saves precious lecture time for explaining and digesting/understanding/applying
  - Experimented with various languages (VBS/VBA, Java, Rexx/ooRexx)
  - 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 non-profit special interest group RexxLA.org
      - Professional and powerful programming language
    - Open-source and free
    - Available for all major platforms (Windows, Linux, macOS)

Part 1

CSF # 1, Example

```
 /* a loop                    */
do i=1 to 99                 -- loop 99 times
   if i=1 then iterate       -- next loop, skip remaining body
   if i=4 then leave         -- leave loop prematurely
   say "hello #" i           -- says "hello # 2" and "hello # 3"
end

   /* demo arithmetics         */
say "i="i "1/i="1/i          -- says "i=4 1/i=0.25"

   /* demo message             */
say "hi, there!"~reverse     -- says "!ereht ,ih"

   /* demo use of a class      */
say ".demo:" .demo           -- says ".demo: The DEMO class"
o=.demo~new                  -- create an instance/object/value
say "o:" o                   -- says "o: a DEMO"
o~name="IEEE 2023"           -- assign value to attribute
say "o~name:"    o~name      -- says "o~name: IEEE 2023"
say "o~reverse:" o~reverse   -- says "o~reverse: 3202 EEEI"

   /* demo defining a class    */
::class demo                 -- defines the class DEMO
::attribute name             -- defines an attribute NAME
::method reverse             -- defines a  method REVERSE
  return self~name~reverse   -- returns the name reversed
```

Output:

```
hello # 2
hello # 3
i=4 1/i=0.25
!ereht ,ih
.demo: The DEMO class
o: a DEMO
o~name: IEEE 2023
o~reverse: 3202 EEEI
```


Rony G. Flatscher

Part 1
Critical Success Factors (CSF)

- 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 ! :-)

Part 1
Critical Success Factors (CSF)

- 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, add new slides because students are interested
    - Rework or add new slides, remove complex slides and improve nutshell examples
    - Retest the new and updated slides and nutshell examples in the next semester's course
  - Allows to gradually improve the course and its materials over time

Part 1

Critical Success Factors (CSF)

- 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

Part 1
Critical Success Factors (CSF)

- 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

Part 1

Critical Success Factors (CSF)

- 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"

Part 1

Critical Success Factors (CSF)

- 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

Part 1
Critical Success Factors (CSF)

- CSF # 8: *Concluding project assignment*
  - Students suggest three projects combining with ooRexx
    - Three Windows programs ("Business Programming 1")
      - At the end of the first half of the semester
    - Three Java jar class libraries ("Business Programming 2")
      - 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
BSF4ooRexx850

Part 2: Critical Success Factor # 1 "Programming Language ooRexx"

- "Business Programming 1 (BP1)" (1st half of semester, 2 months)
  - Rexx: concepts and nutshell example
  - ooRexx: concepts and nutshell examples, camouflaging Windows
    - ooRexx class OLEObject, example: MS Excel
- "Business Programming 2 (BP2)" (2nd half of semester, 2 months)
  - BSF4ooRexx850: concepts and nutshell examples, camouflaging Java
    - Portable ooRexx programs that run unchanged on Windows, macOS, Linux
    - ooRexx class BSF, examples: swing GUI and JDOR (**J**ava2**D** for **o**o**R**exx)

Part 2

Concepts Taught with REXX

- Statement, comment, symbol
- Variable, block, comparison, branch, loops, commands
- Routine, label, function, scopes, associative arrays
- Condition (exception), condition handler

Part 2
REXX, 1

- Mike F. Cowlishaw (IBM)
- IBM released REXX 1979 as a product (IBM product name in all uppercase)
  - Became IBM's SAA strategic procedural language in the 80's
- Design of REXX
  - Explicitly human oriented as opposed to the cryptic EXEC 2 it should replace
    - Goal: easy to learn and easy to maintain
  - Principles
    - Dynamic language, interpreted
    - Typeless language (everything is a string, including numbers)
    - Caseless (everything outside of quotes will be uppercased before processing)
    - No reserved keywords
    - Whitespace can be freely used for formatting instructions for better legibility and better comprehension

Part 2
Rexx, 2

- Only three instruction types
  - Assignment instruction
    - Variable name, followed by the equal sign (**=**), followed by an expression
  - Keyword instruction
    - Keywords are English words that convey their meaning
      - Makes Rexx programs look like pseudo code
    - Starts with one of the defined keyword instructions like call, if, do, ...
  - Command instruction
    - Anything else (an expression evaluating to a string)
      - Or explicitly using the address keyword instruction which allows one to target the environment the command should get sent to
    - By default the command gets sent to the operating system for execution and the command's ***r***eturn ***c***ode is made directly available to Rexx via the variable named **RC**

Rony G. Flatscher

Part 2
Rexx, 3

- A Rexx program demonstrating the three instruction types

```
a="Hello, world"     /* assignment          */

do i=1 to 3          /* a loop              */
   say "... round #" i":" a
End

   /* command, will have a return code     */
"copy file1.txt file1.txt.bkp"
if rc<>0 then  /* variable RC set by REXX */
   SAY "Command's return code:" rc
```

Assuming that the file *file1.txt* does not exist such that the copy command will issue the error message "The system cannot find the file specified." in the command line window

Output:

```
... round # 1: Hello, world
... round # 2: Hello, world
... round # 3: Hello, world
The system cannot find the file specified.
Command's return code: 1
```

Part 2
Concepts Taught with ooRexx

- Directives (routine, requires, class, attribute, method, resource, constant)
- Arguments by reference
- Messages
- OO: Abstract datatype (ADT), class/type, attribute, method routine, creating objects/instances/values, constructor, destructor
- Unknown message handling
- Class hierarchy, (multiple) inheritance, collection classes and iteration
- Windows
  - COM, OLE, Windows registry, ooRexx class OLEObject to camouflage Windows, MS Excel, MS Word as ooRexx
  - MS Internet Explorer: introduction to HTML, MSIE's DHTML

Part 2
ooRexx, 1

- Object-oriented successor for REXX developed by IBM
  - IBM released "Object REXX" 1994 with the operating system "OS/2 Warp"
  - 2004 source code handed over to the non-profit SIG "Rexx Language Assoc."
  - RexxLA.org released "open object Rexx (ooRexx) version 3.0" in 2005
- Design of ooRexx
  - Goals
    - Keep human oriented design principle
    - Run Rexx programs unchanged
  - Influenced by SmallTalk
    - Message paradigm (the tilde character **~** is an explicit message operator in ooRexx)
      - Alan Kay (cf. Wikipedia): ***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".***
      - Simplifies programming as object's implementation is encapsulated (and becomes irrelevant)

Rony G. Flatscher

Part 2
ooRexx, 2

- The message paradigm abstracts from the implementation
  - A programmer conceptually communicates with an object (as if it was a living thing) by sending it a message
    - No need to have any knowledge about the implementation of a method routine
  - The object will search for a method routine by the name of the received message, invokes it (passing any arguments received with the message) and returns any result to the caller
    - If a method routine is not found in the object's class it will search its superclass up to the root class (thereby realizing inheritance): the first found method routine will be executed by the object
    - Should the object not be able to find the method routine an error condition with the message "Object does not understand message" gets raised
- Introduces the *directive* instruction type
  - Placed at the end of a program, led in with two colons **::** followed by one of
    - ANNOTATE, ATTRIBUTE, CLASS, CONSTANT, METHOD, OPTIONS, REQUIRES, RESOURCE, ROUTINE
  - Directive instructions get carried out by the interpreter after the *syntax checking phase*, thereby setting up the program's environment (*setup phase*) before it gets executed (*execution phase*)

Part 2
ooRexx, 3 (Directives and Messages)

```
p1=.person~new("Albert Einstein", 45000)  -- create a new person:  person1
say "p1:" p1~name p1~salary               -- show person1's attribute values

p2=.person~new("Mary Withanyname", 35000) -- create a new person:  person2
say "p2:" p2~name p2~salary               -- show person2's attribute values

p1~increaseSalary(10000)                  -- increase salary of person1
say "p1:" p1~name p1~salary               -- show person1's attribute values

p2~name="Mary Withaspecificname"          -- change the name of person2
p2~salary=45500                           -- change the salary of person2
say "p2:" p2~name p2~salary               -- show person2's attribute values
say "total of salaries:" p1~salary + p2~salary

::class Person              -- define name of class
::attribute name            -- define attribute "name"
::attribute salary          -- define attribute "salary"

::method init               -- define constructor (a method routine)
  expose name salary        -- establish direct access to attributes
  use arg name, salary      -- fetch and store arguments in attributes

::method increaseSalary     -- define method routine
  expose salary             -- establish direct access to attribute "salary"
  use arg increase          -- fetch argument
  salary=salary+increase    -- increase value of salary attribute
```

Output:

```
p1: Albert Einstein 45000
p2: Mary Withanyname 35000
p1: Albert Einstein 55000
p2: Mary Withaspecificname 45500
total of salaries: 100500
```

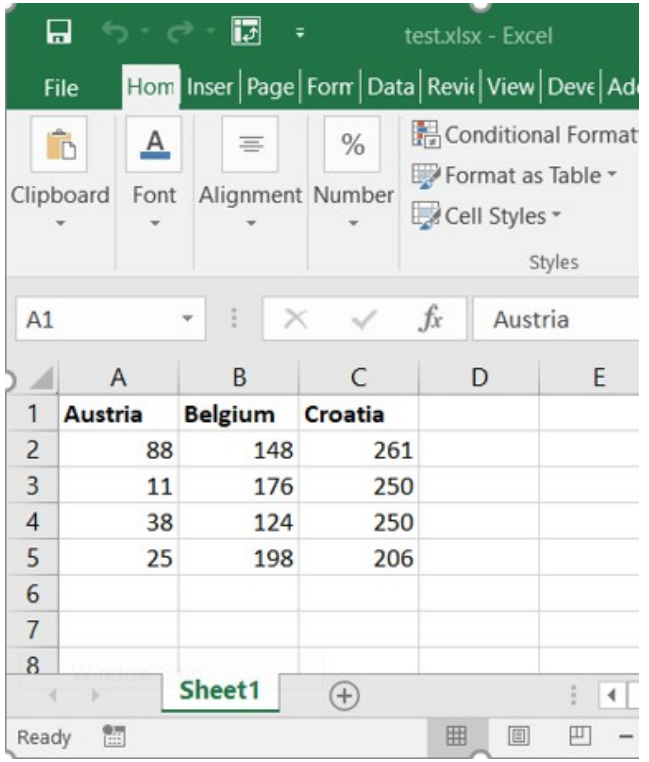
Part 2
Windows Support: ooRexx Proxy Class OLEObject

- Windows version of ooRexx comes with the ooRexx proxy class OLEObject
  - Camouflages Windows OLE (COM) objects as ooRexx objects
  - Easy to create or fetch an instance of a Windows OLE class
  - Allows to send messages to Windows via OLE
  - Allows for callbacks from Windows OLE object events to OLEObject
  - Intercepts the unknown message condition
    - Messages that are not known in ooRexx get forwarded to Windows
      - OLEObject marshals any Windows arguments
      - OLEObject marshals any Windows return value (if a Windows OLE object gets returned then it gets proxied as an OLEObject)
- Makes it easy for students to interact with any Windows OLE object!

Part 2
ooRexx, 4: Messages to MS Excel (Windows)

```
excApp = .OLEObject~new("Excel.Application")
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("Austria", "Belgium", "Croatia")
titleRange~font~bold = .true          -- use bold font for titles
sheet~range("A2:C5")~value = createRows(4)   -- create and assign array
excApp~displayAlerts = .false         -- no alerts (should file exists 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    -- create two-dimensional array with arbitrary data
  use arg items=5                 -- fetch argument, default, if omitted: 5
  arr=.array~new                  -- create Rexx array
  do i=1 to items                 -- create random(min,max) numbers
     arr[i,1] = random( 0 ,100 )     -- Austria
     arr[i,2] = random(101,200 )     -- Belgium
     arr[i,3] = random(201,300)      -- Croatia
  end
  return arr              -- return two-dimensional Rexx array
```

Output:




```
fileName: C:\Users\rony\test.xlsx
```

Part 2
Concepts Taught with BSF4ooRexx850

- Introduction to Java and BSF4ooRexx850 (camouflages Java as ooRexx)
- GUI concepts (awt/swing) 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 (callbacks)

Part 2
BSF4ooRexx850, 1

- Bidirectional bridge between ooRexx and Java
  - In development since 2000, latest version: BSF4ooRexx850
    - Minimum Java version: 8, minimum ooRexx version: 5.0
- Design of BSF4ooRexx850
  - Goals
    - Keep Rexx' human oriented design principle
      - ooRexx programmers need not know implementation details
        - Camouflage Java objects as ooRexx objects that understand messages
      - Allow Java programmers to send ooRexx objects messages from Java
    - Make all Java functionality available to ooRexx in a platform independent manner
    - Includes a Rexx command handler for Java2D named "JDOR" (**J**ava2**D** for **o**o**R**exx)
      - Simplifies using Java2D considerably using Rexx commands

Part 2
BSF4ooRexx850, 2

- Prerequisites for Windows, macOS and Linux
  - Installation (download the version for your operating system)
    - Java 8 or later (Oracle) or OpenJDK 8 (open-source version of Java) or later
      - Hint: use the installation packages with the JavaFX GUI modules ("*FX*" or "*full*" in name)
    - ooRexx 5.0 or later
    - BSF4ooRexx850 or later
  - ooRexx programs that wish to exploit Java classes and objects
    - Get the camouflaging support by requiring (a directive instruction) the ooRexx package named BSF.CLS which defines public classes and public routines

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

    - Use its public proxy class BSF to create and interact with Java objects

Part 2
Java Support: BSF4Rexx850 Proxy Class BSF

- BSF.CLS defines public routines and public classes to support (and ease) interaction with Java, among them the ooRexx proxy class BSF which
  - Camouflages Java objects as ooRexx objects
  - Easies loading or creating an instance of a Java class
  - Allows to send messages to Java
  - Allows for callbacks from Java object events to BSF
  - Intercepts the unknown message condition
    - Messages that are not known in ooRexx get forwarded to Java
      - BSF marshals any Java arguments
      - BSF marshals any Java return value (if a Java object gets returned then it gets proxied as a BSF object)
- Makes it easy for students to interact with any Java (class) object!

Part 2
BSF4ooRexx850, 3 (Messages to Java Objects)

ooRexx:

```
d = .bsf~new("java.awt.Dimension",100,200)
say "d="d~toString
d~width =300
d~height=400
say "d="d~toString

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

Output:

```
d=java.awt.Dimension[width=100,height=200]
d=java.awt.Dimension[width=300,height=400]
```

Java:

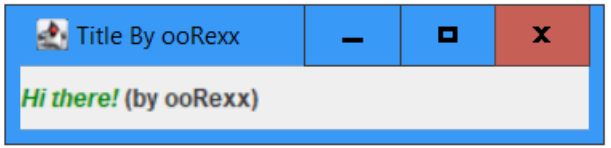
```
import java.awt.Dimension;

class TestDimension  // saved in file "TestDimension.java"
{
    public static void main (String args[])
    {
        Dimension d = new Dimension(100,200);
        System.out.println("d="+d);
        d.width =300;
        d.height=400;
        System.out.println("d="+d);
    }
}
```

Part 2
BSF4ooRexx850, 4 (Messages to Java Objects)

```
jf = .bsf~new("javax.swing.JFrame", "Title By ooRexx")  -- create JFrame
lblText = '<html><em style="color: green;">Hi there!</em> (by ooRexx)</html>'
lbl= .bsf~new("javax.swing.JLabel", lblText)    -- create JLabel
jf~add(lbl)                -- add label
jf~setSize(300,70)         -- set size
jf~setLocation(50,200)     -- set location
jf~visible=.true           -- make visible
jf~toFront                 -- place frame in front of all windows
say 'Hit <enter> to proceed (end) ...'
parse pull data            -- wait until user presses <enter> on the keyboard

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

Output:

```
E:\rony\Vortraege\2023\isecon23\work>rexx code_4_ooRexx_1.rex
Hit <enter> to proceed (end) ...
```



Part 2
BSF4ooRexx850, 5 (JDOR Rexx Command Handler)

- Java2D
  - Powerful 2D graphics
  - Used for drawing light-weight javax.swing classes
  - Used for Java games and business graphics of any kind, …
- Example
  - Java code to create a Java2D graphic from
    - Chuan, H.C. (2008). Java Game Programming 2D Graphics, Java2D and Images, as of 2023-07-29: https://www3.ntu.edu.sg/home/ehchua/programming/java/J8b_Game_2DGraphics.html#zz-2.2
  - Equivalent ooRexx program that creates the same Java2D graphic
    - BSF4ooRexx850 comes with the JDOR (Java2D for ooRexx) Rexx command handler
    - Rexx JDOR commands are *plain strings* that ease this task considerably
      - WU students immediately take advantage of it after the first Java related lecture!

Part 2
BSF4ooRexx850, 6 – Java2D (Java *AffineTransformDemo*)

```
import java.awt.*;
import java.awt.geom.AffineTransform;
import javax.swing.*;

/** Test applying affine transform on vector graphics */
@SuppressWarnings("serial")
public class AffineTransformDemo extends JPanel {
   // Named-constants for dimensions
   public static final int CANVAS_WIDTH = 640;
   public static final int CANVAS_HEIGHT = 480;
   public static final String TITLE = "Affine Transform Demo";

   // Define an arrow shape using a polygon centered at (0, 0)
   int[] polygonXs = { -20, 0, +20, 0};
   int[] polygonYs = { 20, 10, 20, -20};
   Shape shape = new Polygon(polygonXs, polygonYs, polygonXs.length);
   double x = 50.0, y = 50.0;  // (x, y) position of this Shape

   /** Constructor to set up the GUI components */
   public AffineTransformDemo() {
      setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT));
   }

   /** Custom painting codes on this JPanel */
   @Override
   public void paintComponent(Graphics g) {
      super.paintComponent(g);    // paint background
      setBackground(Color.WHITE);
      Graphics2D g2d = (Graphics2D)g;

      // Save the current transform of the graphics contexts.
      AffineTransform saveTransform = g2d.getTransform();
      // Create a identity affine transform, and apply to the Graphics2D context
      AffineTransform identity = new AffineTransform();
      g2d.setTransform(identity);
// … continued …
```

```
// … continued …
      // Paint Shape (with identity transform), centered at (0, 0) as defined.
      g2d.setColor(Color.GREEN);
      g2d.fill(shape);
      // Translate to the initial (x, y) position, scale, and paint
      g2d.translate(x, y);
      g2d.scale(1.2, 1.2);
      g2d.fill(shape);

      // Try more transforms
      for (int i = 0; i < 5; ++i) {
         g2d.translate(50.0, 5.0);  // translates by (50, 5)
         g2d.setColor(Color.BLUE);
         g2d.fill(shape);
         g2d.rotate(Math.toRadians(15.0)); // rotates about transformed origin
         g2d.setColor(Color.RED);
         g2d.fill(shape);
      }
      // Restore original transform before returning
      g2d.setTransform(saveTransform);
   }

   /** The Entry main method */
   public static void main(String[] args) {
      // Run the GUI codes on the Event-Dispatching thread for thread safety
      SwingUtilities.invokeLater(new Runnable() {
         @Override
         public void run() {
            JFrame frame = new JFrame(TITLE);
            frame.setContentPane(new AffineTransformDemo());
            frame.pack();
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setLocationRelativeTo(null); // center the application window
            frame.setVisible(true);
         }
      });
   }
}
```

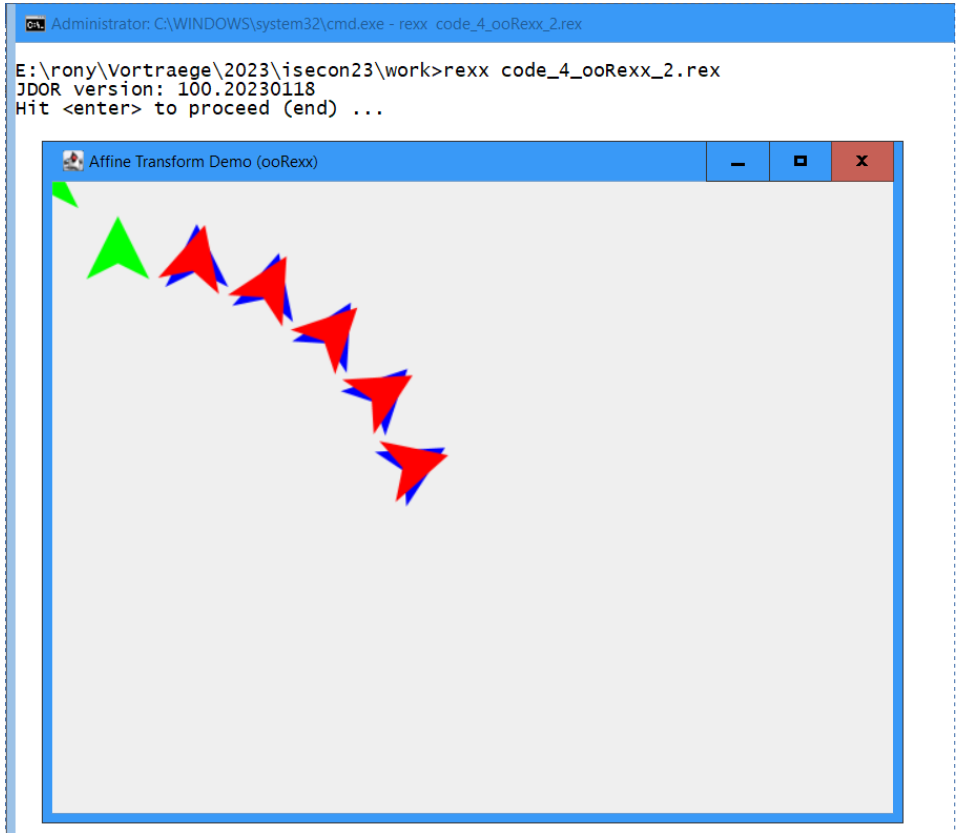
Part 2
JDOR (Java2D for ooRexx) Rexx Command Handler

- BSF4ooRexx850 allows for implementing Rexx command handlers in Java (in addition to assembler, C or C++)
- JDOR started as a proof-of-concept implementation and in the meantime has been fully implemented
  - All Java2D features available via Rexx commands that are pure strings
  - Nutshell samples in BSF4ooRexx850/samples
  - The JDOR Rexx commands reference can be found at BSF4ooRexx850/information/jdor/jdor_doc.html or temporarily at https://wi.wu.ac.at/rgf/rexx/misc/jdor_doc.tmp/jdor_doc.html
- Usage: a JDOR command handler needs to be created and Rexx instructed that Rexx commands are to be addressed to the JDOR command handler

Part 2
BSF4ooRexx850, 7 – Java2D (JDOR Rexx Commands)

```
   -- create a JDOR Rexx command handler
jdh=.bsf~new("org.oorexx.handlers.jdor.JavaDrawingHandler")
say "JDOR version:" jdh~version     -- show version
call BsfCommandHandler "add", "jdor", jdh -- add as a Rexx command handler
address jdor -- set default environment from operating system to JDOR

newImage 640 480           -- create new image
winShow                    -- show image in a window
winTitle "Affine Transform Demo (ooRexx)" -- set window's title

   -- could use Rexx variables denoting the respective Java arrays instead
polygonXs="(-20,0,+20,0)"  -- define four x coordinates
polygonYs="(20,10,20,-20)" -- define four y coordinates
shape myP polygon polygonXs polygonYs 4   -- create polygon shape
color green                -- set color to green
fillShape myP              -- fill (and show) the polygon shape
translate 50 50            -- move origin (x=x+50, y=y+50)
scale 1.2 1.2              -- increase x and y by 20%
fillShape myP              -- fill (and show) the polygon shape

do 5                       -- repeat five times
   translate 50 5          -- move origin (x=x+50, y=y+5)
   color blue              -- set color to blue
   fillShape myP           -- fill (and show) the polygone shape
   rotate 15               -- rotate by 15°
   color red               -- set color to red
   fillShape myP           -- fill (and show) the polygone shape
end

say 'Hit <enter> to proceed (end) ...'
parse pull data         -- wait until user presses <enter> on the keyboard

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

Output (AffineTransformDemo: Java and ooRexx):



Part 3
Critical Lectures that Determine the Success of the Course

Overview of the Slides

1st Installment (Critical): Onboarding
3rd Installment (Critical): Messaging and Object-Orientation
5th Installment (Critical): COM/OLE for Windows Applications
8th Installment (Critical): Java and BSF4ooRexx850

Part 3
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 on the following slides

Part 3
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

Rony G. Flatscher

Part 3
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 (as of Windows 10 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

Part 3
Overview of BP2-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
- **250_AutoJava_XML_SAX**, installment 11, 90': SAX programming model, callbacks to ooRexx, nutshells that extract text, element names, element hierarchy

Part 3
Overview of BP2-Slides, 4

- **250_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

Part 3
1st Installment (Critical) – 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

Part 3
1st Installment (Critical) – 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!

Part 3
3rd Installment (Critical) – Messages & OO, 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

Part 3
3rd Installment (Critical) – Messages & OO, 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

Part 3
3rd Installment (Critical) – Messages & OO, 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

Part 3
5th Installment (Critical) – Windows COM & OLE, 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!

Part 3
5th Installment (Critical) – Windows COM & OLE, 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

Part 3
8th Installment (Critical) – Java and BSF4ooRexx850, 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 default (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

Rony G. Flatscher

Part 3
8th Installment (Critical) – Java and BSF4ooRexx850, 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 ASF)
    - **8**: Java version 8 or later
    - **50**: ooRexx version 5.0 or later
    - Developed for more than 20 years by Rony G. Flatscher

Part 3
8th Installment (Critical) – Java and BSF4ooRexx850, 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

Part 3
8th Installment (Critical) – Java and BSF4ooRexx850, 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: Install Software and Run Nutshell Examples

– ooRexx 5 or Higher
– Java 8 or Higher (*with JavaFX!*)
– BSF4ooRexx850
– Nutshell Examples
(ooRexx/samples, ooRexx/samples/ole/{adsi|appls|methinfo|wmi)
(BSF4ooRexx850/samples/index.html)

Part 4
Hands-on: 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
  - 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`

Part 4
Hands-on: Installation, 2

- ooRexx
  - URL (as of 2023-08-04)
    - Recommended: https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0beta/
    - Released version: https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0/
    - "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

Part 4
Hands-on: 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, long term support)
    - Last non-modular Java version (released 2014), supported at least until December 2030!
    - 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

Rony G. Flatscher

Part 4
Hands-on: Installation, 4

- BSF4ooRexx850
  - Prerequisite: ooRexx 5+ and Java 8+ need to be installed/available via PATH
  - URL (as of 2023-08-04)
    - https://sourceforge.net/projects/bsf4oorexx/files/beta/20221004/
      - Formally in beta, however of release quality, 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: 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 2023-08-04)
      - https://sourceforge.net/projects/bsf4oorexx/files/beta/20221004/MacOS_Universal_BSF4ooRexx850_beta-20230621-with-ooRexx51beta.zip
      - Unzip and run installer

Rony G. Flatscher

Part 4
Hands-on: Nutshell Examples, 1

- ooRexx (as of 2023-08-04), 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

Part 4
Hands-on: Nutshell Examples, 2

- BSFooRexx850 (as of 2023-08-04), 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)

Roundup

- "Business Programming"
  - Four weekly contact hours for one semester (four months)
  - 8 ECTS points, total net teaching load 200 hours
  - *Novices* get empowered by being able to learn programming
    - At the middle of the semester (after two months), after seven installments
      - Fundamental programming concepts, programming Windows (COM/OLE) and MS Office, AOO/LO, cURL
    - At the end of the semester (after four months), after another seven installments
      - Programming exploiting all of Java camouflaged as ooRexx
        - GUI (awt, swing, JavaFX)
        - Client/server socket programming including SSL/TLS
        - Interacting with web servers (curl, Jsoup)
        - Using Java APIs: Apache OpenOffice (AOO)/LibreOffice (LO)
- Critical success factor "programming language"
  - ooRexx with BSF4ooRexx850 (making all of Java/OpenJDK available, camouflaged as ooRexx)
  - All needed software is free and open-source

Links (As of 2023-08-04), 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) 2023: http://wi.wu.ac.at/rgf/wu/lehre/autowin/2023sBP1/BP1-autowin-2023s-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) 2023: http://wi.wu.ac.at/rgf/wu/lehre/autojava/2023sBP2/BP2-autojava-2023s-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#zulu
  - **BSF4ooRexx850**: https://sourceforge.net/projects/bsf4oorexx/files/beta/20221004/
- 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 2023-08-04), 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/2.2.0/
- **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 articles in this context:**
  - 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

Addendum (Rexx vis-à-vis Python)

- Rexx and Python programs
  - Instructions
  - Block, selections, multiple selections
  - Parsing strings
- Possible assessment question
  - What concepts need to be explained and understood by ***novices*** for the Rexx solution and for the feature equivalent Python solution, how much time needs therefore to be planned for each programming language?

Rexx and Python, 1 (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 Windows command (could be typed into a command line window)    */
"dir a.txt"       /* command: list the file a.txt                      */
   /* variable RC contains the command's return code, 0 means success   */
if rc=0 then say "found!"
        else say "some problem occurred, rc="rc /* show return code     */
```

```
# 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("dir a.txt", shell=True) # run command
rc=completedProcess.returncode  # fetch return code, an int
if rc==0:
    print("found!") # indentation mandatory (forcing a block)
else:
    print("some problem occurred, rc="+str(rc)) # turn rc into a string
```

Rexx and Python, 2
(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 of the above program will be:
   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
    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
        print("-> a==max")          # nested block # 2
        print("-> last round!")     # nested block # 2
        print("-> loop will end")   # nested block # 2



""" output of the above program will be:
    1: first round
    2: second round
    3: third round
    (a=4)
    (a=5)
    -> a==max
    -> last round!
    -> loop will end
"""
```

Rexx and Python, 3 (Parsing Strings)

```rexx
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 of the above program will be:
   first name: John, last name: Doe, city: Vienna
   first name: Mary, last name: Doe, city: Tokyo
*/
```

```python
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 of the above program will be:
    first name: John, last name: Doe, city: Vienna
    first name: Mary, last name: Doe, city: Tokyo
"""
```

Acknowledgement

- Many thanks for feedback and proof reading go to

  DI Walter Pachl, Vienna, Austria !