Institut für Betriebswirtschaftslehre und Wirtschaftsinformatik

Abteilung für Wirtschaftsinformatik

Rexx Scripts Hosted and Evaluated/Executed by Java (javax.script)

The 2017 International Rexx Symposium

Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Welthandelsplatz 1 ■ A-1020 Wien

Agenda

- Java scripting framework javax.script, a.k.a. JSR-223
  - Features, ScriptContext
- The ooRexx JSR-223 scripting engine
  - Features, ScriptContext
- Examples
  - Java executing Rexx scripts
  - Rexx scripts fetching arguments from Java
  - Rexx scripts interfacing with the current ScriptContext
- Roundup

The Java Package javax.script, 1

- Package javax.script
  - Defined by the Java Specification Request 223 group (JSR-223)
  - Introduced with Java 1.6/6.0 (2006)
- Features
  - ScriptEngine supplies various eval(...) methods to execute ("evaluate") scripts
    - A script evaluation is always associated with a ScriptContext
    - A ScriptContext maintains at least two Bindings
      - ScriptContext.GLOBAL_SCOPE (constant, numeric value: 200)
      - ScriptContext.ENGINE_SCOPE (constant, numeric value: 100)

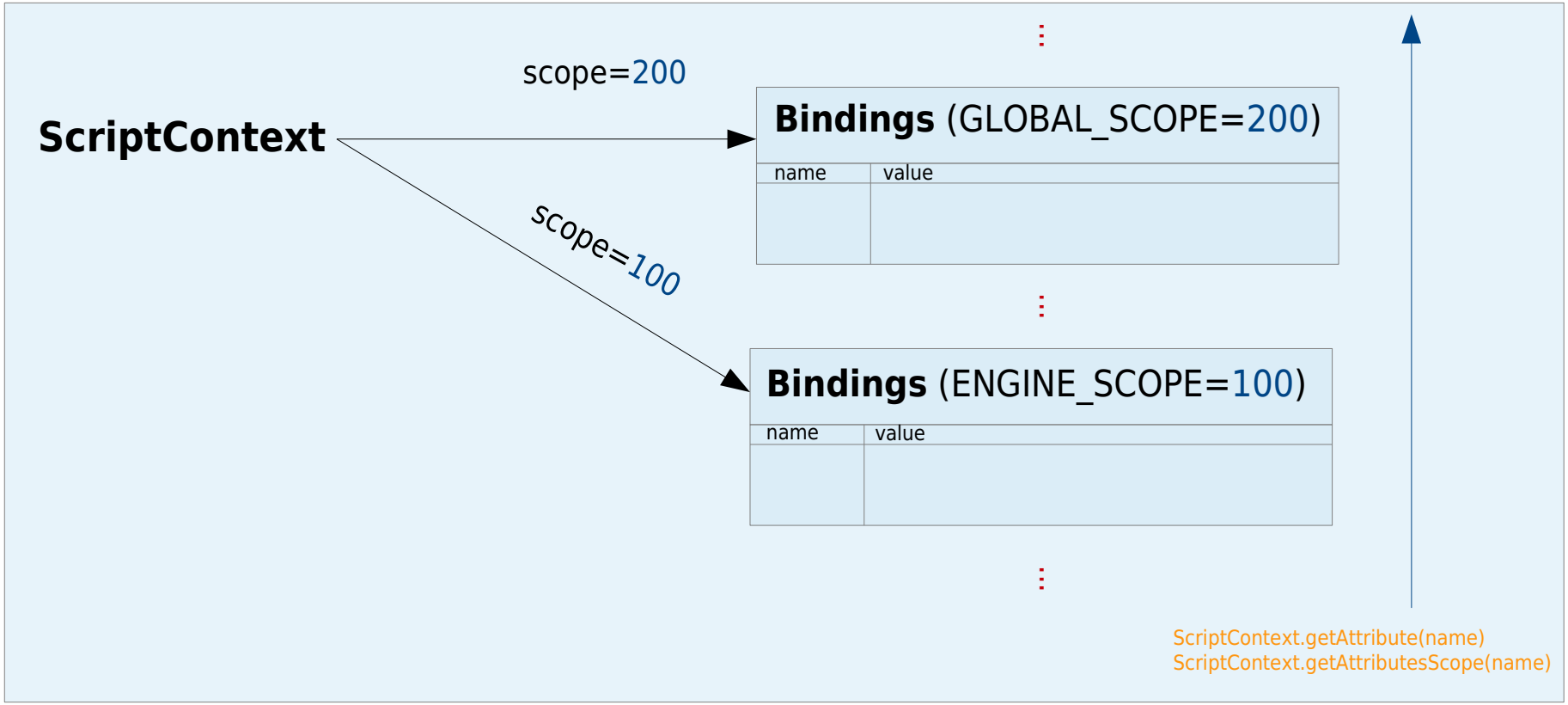
The Java Package javax.script, 2

- ScriptContext and "scopes"
  - Maintains Bindings stored with a specific scope number
  - Retrieving entries ("attributes") without giving a scope number
    - ScriptContext.getAttribute(String name)
      - Search starts in Bindings with lowest scope number to highest
      - The value of the first found entry will be returned, whatever Bindings
        - ScriptContext.getAttributesScope(String name): returns scope number
  - ScriptContext.GLOBAL_SCOPE Bindings
    - Scope value (a predefined constant of type int): 200
    - Visible for all ScriptEngine instances
    - Can be used for coupling and sharing data

The Java Package javax.script, 3

- ScriptContext and "scopes" (continued)
  - ScriptContext.ENGINE_SCOPE
    - Scope value (a predefined constant of type int): 100
    - Visible for a single ScriptEngine instance's invocation
    - Some important possible entries
      - ScriptEngine.ARGV (value: "javax.script.argv")
        - A Java array of type Object
        - Directly available to invoked Rexx scripts as Rexx arguments
      - ScriptEngine.FILENAME (value: "javax.script.filename")
        - The filename from which the script was retrieved, if any

The Java Package javax.script, 4



The Java Package javax.script, 5

- Features (continued)
  - ScriptEngine may optionally implement the interface classes
    - Compilable
      - Methods compile(String script) or compile(Reader script) allow for compilation of a script for reuse
    - Invocable
      - Allows to reuse public functions/routines of previously executed scripts
        - getInterface(Class InterfaceClass) returns an object which will invoke the InterfaceClass methods as public functions/routines in the script
        - invokeFunction(String name, Object … args)
      - Allows to use a script object's methods
        - getInterface(Object thiz, Class InterfaceClass) returns an object which will invoke the InterfaceClass methods in the returned script object
        - invokeMethod(Object thiz, String name, Object … args)

The ooRexx JSR-223 Scripting Engine, 1

- Features
  - Implements abstract class SimpleScriptEngine (a ScriptEngine)
    - org.rexxla.bsf.engines.rexx.jsr223.RexxScriptEngine
  - Implements the optional interfaces
    - Compilable
      - Rexx script gets tokenized
      - cf. org.rexxla.bsf.engines.rexx.jsr223.RexxCompiledScript
    - Invocable
  - The ooRexx RexxScriptEngine is fully featured!
    - E.g. deployable in JavaFX FXML (GUI) definitions

The ooRexx JSR-223 Scripting Engine, 2

- Each invocation of Rexx adds an
  - Additional slot argument as the last argument
  - A Rexx .directory object containing Java invocation related information
    - In the case of RexxScript invocations there is an entry named ScriptContext allowing direct access to this important Java object
      - Allows access to any Bindings maintained by the ScriptContext, e.g.
        - getBindings(scope): returns the Bindings of the given scope (a number)
        - getAttribute(name[,scope]): returns the value of the entry or .nil
        - getAttributesScope(name): returns the scope (a number) of the Bindings
        - removeAttribute(name,scope): removes an entry from the Bindings
        - setAttribute(name,value,scope): adds or changes a value to/in the given Bindings

The ooRexx JSR-223 Scripting Engine, 3

- Each invocation of Rexx uses
  - ScriptContext.getReader() as the standard input file ("stdin")
    - RexxScriptEngine will prepend any input with the string "REXXin>"
  - ScriptContext.getWriter() as the standard output file ("stdout")
    - RexxScriptEngine will prepend any output with the string "REXXout>"
  - ScriptContext.getErrorWriter() as the standard error file ("stderr")
    - RexxScriptEngine will prepend any output with the string "REXXerr>"

The ooRexx JSR-223 Scripting Engine, 4

- Each invocation of a Rexx script will
  - BSF.CLS will be always available to any Rexx script, therefore no need to code the directive ::requires "BSF.CLS" !
  - All public routines and all public classes in an executed Rexx script will be made available to all Rexx scripts that get invoked thereafter!
    - Resulting effect is as if a Rexx script would have issued a requires to each of the previously executed Rexx scripts!

Example 01: Test_01.java Evaluating/Executing the Rexx Script test_rexx_01.rex, 1

- Java program Test_01.java
  - Creates a RexxScriptEngine gets its default ScriptContext
  - Sets the filename of the Rexx script in the ENGINE_SCOPE
  - Does not (yet) define arguments for the Rexx script
  - Evaluates the Rexx script stored in the file test_rexx_01.rex
    - This evaluation will not supply any arguments to the Rexx script
  - Sets Java arguments for the Rexx script in the ENGINE_SCOPE
  - Fetches the current Rexx script and re-evaluates/re-executes it
    - This evaluation **will** supply arguments to the Rexx script!

Example 01: Test_01.java Evaluating/Executing the Rexx Script test_rexx_01.rex, 2

```
import javax.script.*;
import java.io.FileReader;
import org.rexxla.bsf.engines.rexx.jsr223.*;

public class Test_01    // demo evaluating a Rexx script
{
    public static void main (String args[])
    {
        ScriptEngineManager manager = new ScriptEngineManager();
        RexxScriptEngine rse=(RexxScriptEngine) manager.getEngineByName("Rexx");

        try
        {
            String filename="test_rexx_01.rex"; // define the filename
                // add the filename to the engine's SimpleBindings
            ScriptContext sc=rse.getContext();  // get the default ScriptContext
            sc.setAttribute(ScriptEngine.FILENAME,filename,ScriptContext.ENGINE_SCOPE);
            rse.eval(new FileReader(filename)); // now let us execute the Rexx script

            System.out.println("\n... about to reuse the last used Rexx script ...\n");
                // add arguments for the script to the ENGINE_SCOPE bindings
            sc.setAttribute(ScriptEngine.ARGV,
                        new Object[] {"one", null, java.util.Calendar.getInstance()},
                        ScriptContext.ENGINE_SCOPE);
                // the RexxScriptEngine always compiles the last script and
                // makes it available with the getCurrentScript() method
            rse.getCurrentScript().eval();          // now let us re-execute the Rexx script
        }
        catch (Exception exc)
        {
            System.err.println(exc);
            System.exit(-1);
        }
    }
}
```


Example 01: Test_01.java Evaluating/Executing the Rexx Script test_rexx_01.rex, 3

- The Rexx script program test_rexx_01.rex
  - Shows the information from PARSE SOURCE
  - Displays the received arguments
    - If an argument is a Rexx directory, then its content is shown

Example 01: Test_01.java Evaluating/Executing the Rexx Script test_rexx_01.rex, 4

```
parse source s
say "parse source: ["s"]"
say

say "received" arg() "arguments:"
do i=1 to arg()
   val=arg(i)      -- get value
   say "   arg #" i": ["val"]"
   if val~isA(.directory) then
   do
      say "   a directory with the following entries:"
      loop idx over val~allIndexes~sort
         say "       idx=["idx"] -> item=["val[idx]"]"
      end
   end
end
```

Example 01: Test_01.java Evaluating/Executing the Rexx Script test_rexx_01.rex, 5

```
REXXout>parse source: [WindowsNT SUBROUTINE test_rexx_01.rex]
REXXout>
REXXout>received 1 arguments:
REXXout>   arg # 1: [a Directory]
REXXout>   a directory with the following entries:
REXXout>       idx=[SCRIPTCONTEXT] -> item=[javax.script.SimpleScriptContext@117d9a3]
```

```
... about to reuse the last used Rexx script ...
```

```
REXXout>parse source: [WindowsNT SUBROUTINE test_rexx_01.rex]
REXXout>
REXXout>received 4 arguments:
REXXout>   arg # 1: [one]
REXXout>   arg # 2: [The NIL object]
REXXout>   arg # 3: [java.util.GregorianCalendar@ed1f14]
REXXout>   arg # 4: [a Directory]
REXXout>   a directory with the following entries:
REXXout>       idx=[SCRIPTCONTEXT] -> item=[javax.script.SimpleScriptContext@117d9a3]
```

Example 02: Test_02.java Evaluating/Executing the Rexx Script test_rexx_02.rex, 1

- Java program Test_02.java
  - Creates a RexxScriptEngine gets its default ScriptContext
  - Sets the filename of the Rexx script in the ENGINE_SCOPE
  - Does not (yet) define arguments for the Rexx script
  - Evaluates the Rexx script stored in the file test_rexx_02.rex
    - This evaluation will not supply any arguments to the Rexx script
  - Sets Java arguments for the Rexx script in the ENGINE_SCOPE
  - Fetches the current Rexx script and re-evaluates/re-executes it
    - This evaluation **will** supply arguments to the Rexx script!

Example 02: Test_02.java Evaluating/Executing the Rexx Script test_rexx_02.rex, 2

```
import javax.script.*;
import java.io.FileReader;
import org.rexxla.bsf.engines.rexx.jsr223.*;

public class Test_02    // demo evaluating a Rexx script
{
    public static void main (String args[])
    {
        ScriptEngineManager manager = new ScriptEngineManager();
        RexxScriptEngine rse=(RexxScriptEngine) manager.getEngineByName("Rexx");

        try
        {
            String filename="test_rexx_02.rex";      // define the filename
                // add the filename to the engine's SimpleBindings
            ScriptContext sc=rse.getContext();  // get the default ScriptContext
            sc.setAttribute(ScriptEngine.FILENAME,filename,ScriptContext.ENGINE_SCOPE);
            rse.eval(new FileReader(filename), sc);  // now let us execute the Rexx script

            System.out.println("\n... about to reuse the last used Rexx script ...\n");
                // add arguments for the script to the ENGINE_SCOPE bindings
            sc.setAttribute(ScriptEngine.ARGV,
                        new Object[] {"one", null, java.util.Calendar.getInstance()},
                        ScriptContext.ENGINE_SCOPE);
                // the RexxScriptEngine always compiles the last script and i
                // makes it available with the getCurrentScript() method
            rse.getCurrentScript().eval();         // now let us re-execute the Rexx script
        }
        catch (Exception exc)
        {
            System.err.println(exc);
            System.exit(-1);
        }
    }
}
```

Example 02: Test_02.java Evaluating/Executing the Rexx Script test_rexx_02.rex, 3

- The Rexx script program test_rexx_02.rex
  - Shows the information from PARSE SOURCE
  - Fetches the ScriptContext from the slotDir argument
    - The slotDir argument is always appended, hence the last argument
    - Demonstrates how to interacts with the ScriptContext
  - Displays the received arguments
    - If an argument is a Java object (an instance of the Rexx proxy class named BSF) then its toString method is employed
    - If an argument is a Rexx directory, then its content is shown

Example 02: Test_02.java Evaluating/Executing the Rexx Script test_rexx_02.rex, 4

```
parse source s
say "parse source: ["s"]"
say
-- demonstrate how to access and use the ScriptContext
slotDir=arg(arg())        -- last argument is a directory containing "SCRIPTCONTEXT"
sc=slotDir~scriptContext   -- fetch the ScriptContext object
say "ScriptContext field: ENGINE_SCOPE:" pp(sc~engine_scope)
say "ScriptContext field: GLOBAL_SCOPE:" pp(sc~global_scope)
say
-- import the Java class that defines some Constants like FILENAME, ARGV ...
seClz=bsf.importClass("javax.script.ScriptEngine")
say "ScriptEngine field: FILENAME="pp(seClz~filename)
say "ScriptEngine field: ARGV    ="pp(seClz~argv)
say
key=seClz~FILENAME      -- get string value for FILENAME entry in Bindings
say "value of ScriptEngine static field named ""FILENAME"":" pp(key)
say "  fetch filename from ScriptContext                :" pp(sc~getAttribute(key))
say "  fetch scope (engine or global) from Scriptcontext:" pp(sc~getAttributesScope(key))
say
key=seClz~ARGV          -- get string value for ARGV entry in Bindings
say "value of ScriptEngine static field named ""ARGV""    :" pp(key)
say "  fetch filename from ScriptContext                :" pp(sc~getAttribute(key))
say "  fetch scope (engine or global) from Scriptcontext:" pp(sc~getAttributesScope(key))
say "---"
say "received" arg() "arguments:"
do i=1 to arg()
   val=arg(i)
   str="    arg("i")=["
   if val~isA(.bsf) then str=str || val~toString"]"
                    else str=str || val"]"
   say str
   if val~isA(.directory) then
   do
      say "    a directory with the following entries:"
      loop idx over val~allIndexes~sort
         say "        idx=["idx"] -> item=["val[idx]"]"
      end
   end
end
```

Example 02: Test_02.java Evaluating/Executing the Rexx Script test_rexx_02.rex, 5

```
RE REXXout>parse source: [WindowsNT SUBROUTINE test_rexx_02.rex]
REXXout>
REXXout>ScriptContext field: ENGINE_SCOPE: [100]
REXXout>ScriptContext field: GLOBAL_SCOPE: [200]
REXXout>
REXXout>ScriptEngine field: FILENAME=[javax.script.filename]
REXXout>ScriptEngine field: ARGV    =[javax.script.argv]
REXXout>
REXXout>value of ScriptEngine static field named "FILENAME": [javax.script.filename]
REXXout>  fetch filename from ScriptContext                 : [test_rexx_02.rex]
REXXout>  fetch scope (engine or global) from Scriptcontext: [100]
REXXout>
REXXout>value of ScriptEngine static field named "ARGV"     : [javax.script.argv]
REXXout>  fetch filename from ScriptContext                 : [The NIL object]
REXXout>  fetch scope (engine or global) from Scriptcontext: [-1]
REXXout>---
REXXout>received 1 arguments:
REXXout>    arg(1)=[a Directory]
REXXout>    a directory with the following entries:
REXXout>        idx=[SCRIPTCONTEXT] -> item=[javax.script.SimpleScriptContext@117d9a3]

... about to reuse the last used Rexx script ...
```

[output continued on next page ...]

Example 02: Test_02.java Evaluating/Executing the Rexx Script test_rexx_02.rex, 6

[... continued from previous page ...]

```
REXXout>parse source: [WindowsNT SUBROUTINE test_rexx_02.rex]
REXXout>
REXXout>ScriptContext field: ENGINE_SCOPE: [100]
REXXout>ScriptContext field: GLOBAL_SCOPE: [200]
REXXout>
REXXout>ScriptEngine field: FILENAME=[javax.script.filename]
REXXout>ScriptEngine field: ARGV    =[javax.script.argv]
REXXout>
REXXout>value of ScriptEngine static field named "FILENAME": [javax.script.filename]
REXXout>  fetch filename from ScriptContext                 : [test_rexx_02.rex]
REXXout>  fetch scope (engine or global) from Scriptcontext: [100]
REXXout>
REXXout>value of ScriptEngine static field named "ARGV"     : [javax.script.argv]
REXXout>  fetch filename from ScriptContext                 : [[Ljava.lang.Object;@107d329]
REXXout>  fetch scope (engine or global) from Scriptcontext: [100]
REXXout>---
REXXout>received 4 arguments:
REXXout>    arg(1)=[one]
REXXout>    arg(2)=[The NIL object]
REXXout>
arg(3)=[java.util.GregorianCalendar[time=1486496302996,areFieldsSet=true,areAllFieldsSet=true,lenient=true
,zone=sun.util.calendar.ZoneInfo[id="Europe/Berlin",offset=3600000,dstSavings=3600000,useDaylight=true,tra
nsitions=143,lastRule=java.util.SimpleTimeZone[id=Europe/Berlin,offset=3600000,dstSavings=3600000,useDayli
ght=true,startYear=0,startMode=2,startMonth=2,startDay=-
1,startDayOfWeek=1,startTime=3600000,startTimeMode=2,endMode=2,endMonth=9,endDay=-
1,endDayOfWeek=1,endTime=3600000,endTimeMode=2]],firstDayOfWeek=2,minimalDaysInFirstWeek=4,ERA=1,YEAR=2017
,MONTH=1,WEEK_OF_YEAR=6,WEEK_OF_MONTH=2,DAY_OF_MONTH=7,DAY_OF_YEAR=38,DAY_OF_WEEK=3,DAY_OF_WEEK_IN_MONTH=1
,AM_PM=1,HOUR=8,HOUR_OF_DAY=20,MINUTE=38,SECOND=22,MILLISECOND=996,ZONE_OFFSET=3600000,DST_OFFSET=0]]
REXXout>    arg(4)=[a Directory]
REXXout>    a directory with the following entries:
REXXout>        idx=[SCRIPTCONTEXT] -> item=[javax.script.SimpleScriptContext@117d9a3]
```


Rexx Script Annotations, 1

- Rexx block comments with a symbol starting with @
  - Must end in the same line to be processed!
- Makes it easy to get attributes from Bindings
  - Will be immediately made available as local Rexx variables
  - /*@get(name1 name2...)*/
- Makes it easy to set attributes in Bindings
  - Will be immediately set from local Rexx variables
  - Attribute entry must exist in one of the available Bindings
  - /*@set(name1 name2...)*/

Rexx Script Annotations, 2

- For debugging the following Rexx script annotation is defined
  - /*@showsource*/
  - RexxScriptEngine displays the Rexx script source that gets evaluated/executed
  - If Rexx script @get and @set annotations are present, the edited Rexx script source will be shown in addition

Example 03: Test_03.java Evaluating/Executing the Rexx Script test_rexx_03.rex, 1

- Java program Test_03.java
  - Creates a ScriptEngine gets its default ScriptContext
  - Sets the filename of the Rexx script in the ENGINE_SCOPE
  - Defines the attributes d1, d2 and sum in GLOBAL_SCOPE
  - Evaluates the Rexx script stored in the file test_rexx_03.rex
    - Upon return the public routines ONE, TWO and PP are available to Rexx scripts that are evaluated later
  - Calls the public Rexx routine ONE, shows attributes afterwards
  - Calls the public Rexx routine TWO, shows attributes afterwards
    - The attributes d1, d2 and sum will reflect the Rexx values!

Example 03: Test_03.java Evaluating/Executing the Rexx Script test_rexx_03.rex, 2

```
import javax.script.*;
import java.io.FileReader;
import org.rexxla.bsf.engines.rexx.jsr223.*;
public class Test_03    // demo evaluating a Rexx script
{
    public static void main (String args[])
    {
        ScriptEngineManager manager = new ScriptEngineManager();
        ScriptEngine            rse = manager.getEngineByName("Rexx");
        try
        {
            String filename="test_rexx_03.rex";      // define the filename
                // add the filename to the engine's SimpleBindings
            ScriptContext sc=rse.getContext();  // get the default ScriptContext
            sc.setAttribute(ScriptEngine.FILENAME,filename,ScriptContext.ENGINE_SCOPE);
            sc.setAttribute("d1",  "1", ScriptContext.GLOBAL_SCOPE);
            sc.setAttribute("d2",  "2", ScriptContext.GLOBAL_SCOPE);
            sc.setAttribute("sum", "3", ScriptContext.GLOBAL_SCOPE);
            showAttributes(sc);
            rse.eval(new FileReader(filename), sc);  // now let us execute the Rexx script
            System.out.println("\n... about to call public Rexx routine 'one':");
            // now let us execute a global Rexx routine, forward slotDir argument!
            rse.eval("call one arg(arg())");
            showAttributes(sc);

            System.out.println("\n... about to call public Rexx routine 'two':");
            // now let us execute a global Rexx routine, forward slotDir argument!
            rse.eval("call two arg(arg())");
            showAttributes(sc);
        }
        catch (Exception exc) { System.err.println(exc); System.exit(-1); }
    }
    public static void showAttributes(ScriptContext sc)
    {
        System.out.println("... d1=["+sc.getAttribute("d1")+"]"
                           + ", d2=["+sc.getAttribute("d2")+"]"
                          + ", sum=["+sc.getAttribute("sum")+"] ...");
    }
}
```

Example 03: Test_03.java Evaluating/Executing the Rexx Script test_rexx_03.rex, 3

- The Rexx script program test_rexx_03.rex
  - Fetches the attributes d1, d2 and sum from GLOBAL_SCOPE and displays them
  - Defines the public routines ONE, TWO and PP
    - Rexx script annotations are used to fetch the attributes
    - The local variables d1 and d2 will get random values, the local variable sum will contain the result of adding d1 and d2
    - Routine TWO will overwrite the attribute values

Example 03: Test_03.java Evaluating/Executing the Rexx Script test_rexx_03.rex, 4

```
scriptContext=arg(arg())~scriptContext -- get ScriptContext
d1=scriptContext~getAttribute("d1")    -- get attribute
d2=scriptContext~getAttribute("d2")    -- get attribute
sum=scriptContext~getAttribute("sum")  -- get attribute
say "d1="d1", d2="d2", sum="sum

/* access attributes with Rexx script annotations  */
::routine one public
/*@get(d1 d2 sum)*/                    -- get attributes
say "d1="pp(d1)", d2="pp(d2)", sum="pp(sum)
d1=random()
d2=random()
sum=d1+d2
say "d1="pp(d1)", d2="pp(d2)", sum="pp(sum)

/* access attributes with Rexx script annotations  */
::routine two public
/*@get(d1 d2 sum)*/                    -- get attributes
say "d1="pp(d1)", d2="pp(d2)", sum="pp(sum)
d1=random()
d2=random()
sum=d1+d2
say "d1="pp(d1)", d2="pp(d2)", sum="pp(sum)
say "--> ---> now updating Bindings from Rexx! <--- <--"
/*@set(d1 d2 sum)*/  -- replace the values in the Bindings

::routine pp   -- "pretty-print": enclose argument in brackets
  return "["arg(1)"]"
```

Example 03: Test_03.java Evaluating/Executing the Rexx Script test_rexx_03.rex, 5

```
... d1=[1], d2=[2], sum=[3] ...
REXXout>d1=1, d2=2, sum=3
```

```
... about to call public Rexx routine 'one':
REXXout>d1=[1], d2=[2], sum=[3]
REXXout>d1=[618], d2=[62], sum=[680]
... d1=[1], d2=[2], sum=[3] ...
```

```
... about to call public Rexx routine 'two':
REXXout>d1=[1], d2=[2], sum=[3]
REXXout>d1=[248], d2=[64], sum=[312]
REXXout>--> ---> now updating Bindings from Rexx! <--- <--
... d1=[248], d2=[64], sum=[312] ...
```

Roundup and Outlook

- Roundup
  - The JSR-223 support for the ooRexx interpreter allows any Rexx program to be run/evaluated/executed by Java
  - Rexx scripts are able to fetch and interact with the ScriptContext
  - Rexx script annotations ease fetching and setting attribute entries in any of the Bindings
  - Each RexxScriptEngine object will cause the creation of a new ooRexx interpreter instance
    - All Rexx scripts executed within a Rexx interpreter instance have
      - All public routines and public classes defined in earlier executed Rexx scripts directly available to them!