Modul University Vienna


# Creating ooRexx Programs from Java Programs

## Applying the oRexx/Java Language Bindings

**Also see this year's tutorial about the ooRexx/Java bindings on Sunday afternoon entitled "Introduction to BSF4ooRexx"**

37th International Rexx Symposium
May 3rd – May 6th 2026, Barcelona



Prof. Rony G. Flatscher

# Overview

- An ooRexx class and a matching Java class
  - Use ooRexx class from Java, use Java class from Rexx
- Brief overview of the Java programming language
  - Some very important concepts
  - What can be easily mapped to ooRexx?
  - A simple sample
- Mapping Java to ooRexx
  - Iterating over Java collections with ooRexx means including Enum classes
  - Interface classes
  - A more complex example
- Roundup


Prof. Rony G. Flatscher

# The ooRexx Class RexxDimension, 1

- Demonstrate and use an ooRexx class RexxDimension
  - Consisting of two attributes, width and height
    - If no arguments for width and height are supplied at object creation time, then their values should be set to 0 by default
      - In ooRexx the constructor method routine is named INIT and will get invoked via the NEW method of an ooRexx class which will forward the received arguments, if any, in the same order to the INIT method routine
  - Showing the Rexx object with the SAY keyword instruction should indicate the name of the Rexx class and the values for the attributes width and height
    - Can be easily done in ooRexx by defining a method routine named makeString that returns a string


Prof. Rony G. Flatscher

# The ooRexx Class RexxDimension, 2

## ooRexx Program "RexxDimension.rex"

```
-- RexxDimension.rex
-- main program ("prolog"), compare to Java's static main
method
rd = .RexxDimension~new
say rd
rd~width = 123             -- assigning a new value to attribute
rd~height = 456            -- assigning a new value to attribute
say rd "(updated)"
say .RexxDimension~new(321,654)
say

::class "RexxDimension"    public  -- creates the Rexx class

      --- define instance attributes (fields) and methods ---
::attribute width   -- creates getter and setter method
::attribute height  -- creates getter and setter method

::method init       -- constructor method
  expose width height      -- establish direct access
  use strict arg width=0, height=0

::method makeString -- override default makeString
  expose width height      -- establish direct access
  return self~class~id"[width="width",height="height"]"
```

Output:

```
RexxDimension[width=0,height=0]
RexxDimension[width=123,height=456] (updated)
RexxDimension[width=321,height=654]
```

# The ooRexx Class JavaDimension, 1
## Java Program "JavaDimension.java"

```
// JavaDimension.java
public class JavaDimension
{
    // --- define static (class) fields and methods ---
    public static void main(String[] args)  // optional static main() method
    {
        JavaDimension jd=new JavaDimension();       // create instance
        System.out.println(jd.toString());   // show String representation
        jd.setWidth(123);                   // set width
        jd.setHeight(456);                  // set height
        System.out.println(jd + " (updated)");      // show String representation
        System.out.println(new JavaDimension(321,654));     // create instance
       System.out.println();                // output empty line
    }

    // --- define instance fields and methods ---
    int width = 0;            // define field
    int height = 0;           // define field

    public JavaDimension()           // default constructor method
    {}

    public JavaDimension(int w, int h)      // constructor method
    {
        width=w;
        height=h;
    }

    public int getWidth ()    // getter method
    {
        return width;
    }

    public void setWidth(int w)      // setter method
    {
        width=w;
    }

    public int getHeight ()          // getter method
    {
        return height;
    }

    public void setHeight(int h)     // setter method
    {
        height=h;
    }

    public String toString()  // override default toString()
    {
        return this.getClass().getName()+"[width="+width+",height="+height+"]";
    }
}
```

Output:

```
JavaDimension[width=0,height=0]
JavaDimension[width=123,height=456] (updated)
JavaDimension[width=321,height=654]
```


Prof. Rony G. Flatscher

# ooRexx Program Using Java Class, 1

- Demonstrate and use a Java class JavaDimension that is equivalent to RexxDimension
- Some notes
  - Compile with javac JavaDimension.java which creates the compiled file JavaDimension.**class**
    - Make sure that the environment variable CLASSPATH contains a dot (“.”, current directory) such that Java can find the class in the current directory as well , otherwise a *ClassNotFoundException* gets thrown by Java
      - Programs, including Rexx programs, that use this Java class need to run off tthe current directory where the file JavaDimension.class is located
    - Run the program with java JavaDimension (no file extension!)


Prof. Rony G. Flatscher

# ooRexx Program Using Java Class, 2

## Use Java fields Like ooRexx Attributes

```
-- RexxJavaDimension1.rex
-- main program ("prolog"), compare to Java's static main method
jd = .bsf~new("JavaDimension")
say jd~toString      -- show the string representation
jd~width = 123       -- treat Java fields as Rexx attributes!
jd~height = 456      -- treat Java fields as Rexx attributes!
say jd~toString      -- show the string representation
say .bsf~new("JavaDimension",321,654)~toString

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

Output:

```
JavaDimension[width=0,height=0]
JavaDimension[width=123,height=456]
JavaDimension[width=321,height=654]
```


Prof. Rony G. Flatscher

# ooRexx Program Using Java Class, 3

## Use the Java Setter Methods

```
-- RexxJavaDimension2.rex
-- main program ("prolog"), compare to Java's static main method
jd = .bsf~new("JavaDimension")  -- create an instance
say jd~toString          -- show the string representation
jd~setWidth(123)         -- use Java setter method
jd~setHeight(456)        -- use Java setter method
say jd~toString          -- show the string representation
say .bsf~new("JavaDimension",321,654)~toString

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

Output:

```
JavaDimension[width=0,height=0]
JavaDimension[width=123,height=456]
JavaDimension[width=321,height=654]
```


Prof. Rony G. Flatscher

# Java Program Using ooRexx Class, 1

- Demonstrate a Java class that uses the Rexx class RexxDimension
  - Java program uses the Java scripting framework to load the Rexx engine
  - Java program executes a small Rexx program that calls RexxDimension.rex and which returns the Rexx class object for "RexxDimension"
    - Note # 1: will run its main program (*prolog*), hence output from Rexx will occur
    - Note # 2: using the Java scripting framework will cause a prefix to the standard monitor objects (`.input`, `.output`, `.error`, `.traceOutput`, `.debugInput`) to be prepended
  - Java program creates a RexxDimension instance without supplying arguments
    - It uses the attribute setters (the messages append an equal sign to the attribute name)
    - It uses makeString to have the Rexx object show its representing string value
  - Java program creates another RexxDimension instance this time supplying arguments
    - It uses makeString to have the Rexx object show its representing string value

# Java Program Using ooRexx Class, 2

# Java Program "JavaUseRexxDimension.java"

```
// JavaUseRexxDimension.java
import javax.script.ScriptEngineManager;
import javax.script.ScriptEngine;
import org.rexxla.bsf.engines.rexx.RexxProxy;

public class JavaUseRexxDimension
{
    public static void main(String[] args)    // optional static main() method
    {
        try {
            ScriptEngine engine = new ScriptEngineManager().getEngineByName("rexx");
            String code = "call RexxDimension.rex \n" +
                          "return .RexxDimension -- return ooRexx class object \n";
            RexxProxy rpClz = (RexxProxy) engine.eval(code);
               // create a RexxDimension object
            RexxProxy ro = (RexxProxy) rpClz.sendMessage0("NEW");
            System.out.println("Java: "+ro.sendMessage0("makeString"));
            ro.sendMessage1("WIDTH=", 666);     // set attribute
            ro.sendMessage1("HeIgHt=", 777);    // set attribute
            System.out.println("Java: "+ro.sendMessage0("makeString"));
               // create a RexxDimension option
            ro = (RexxProxy) rpClz.sendMessage2("NEW", 888, 999);
            System.out.println("Java: "+ro.sendMessage0("makeString"));
        }
        catch (Throwable t)
        {
            System.err.println("Java: error occurred: " + t);
        }
       System.exit(0);        // end Java
   }
}
```

Output:

```
REXXout>RexxDimension[width=0,height=0]
REXXout>RexxDimension[width=123,height=456] (updated)
REXXout>RexxDimension[width=321,height=654]
REXXout>
Java: RexxDimension[width=0,height=0]
Java: RexxDimension[width=666,height=777]
Java: RexxDimension[width=888,height=999]
```

# Java

- Programming language with the following notable features
  - Compiles to machine instructions (*"bytecode"*) of an *artificial processor*
  - Needs a "Java virtual machine (JVM)" to execute the bytecode
    - JVMs are available for all important operating systems and hardware architectures
    - *Hence, a Java class or a Java program, once compiled can be run everywhere!*
  - Distributed with a (huge) "Java runtime environment (JRE)"
    - A *huge Java class library* that offers everything that an application may possibly need
      - E.g. Socket classes for Internet programming, GUI classes for graphical user interfaces, ...
    - Uncountable third party Java class libraries, most available as open-source (e.g. ASF)
  - Most important programs get programmed with Java (even Android applications!)
  - Many professional applications that are not programmed in Java offer Java APIs
    - E.g. SAP, OpenOffice/LibreOffice, ...
- Hence Java is truly a programmer's "treasure trove" for all operating systems!

# Things to Know About Java, 1

- Strictly typed language
  - Primitive types
    - boolean, byte, char, short, int, long, float, double
  - Object-oriented types
    - Any Java class, e.g.
      - `java.awt.Dimension`, `java.lang.String`, `java.lang.System`, `...`
    - Wrapper classes for primitive types
      - `java.lang.Boolean`, `java.lang.Byte`, `java.lang.Character`, `java.lang.Short`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Float`, `java.lang.Double`
      - "boxing": wraps up a primitive value into a wrapper object
      - "unboxing": retrieves a primitive value from its wrapper object

# Things to Know About Java, 2

- Case sensitive
  - Upper- and lowercase significant!
- Classes organized in packages
  - Package names may be compound
    - E.g. "java.lang"
  - Fully "qualified class name" includes package name
    - e.g. "java.lang.String"
  - "Unqualified class name"
    - e.g. "String"


Prof. Rony G. Flatscher

# Things to Know About Java, 3

- A Java class may consist of
  - Fields (comparable to ooRexx attributes) and
  - Methods (comparable to ooRexx methods)
- Fields and methods
  - Static fields and static methods
    - Sometimes dubbed "class fields" and "class methods"
    - Available to the class object *and* its instances
  - Otherwise "instance methods"
    - Only available to instances of a Java class

# BSF4ooRexx850

- External Rexx function package
  - Allows to interact with the Java runtime environment (JRE)
    - Exploit functionality of Java classes
    - Exploit functionality of Java objects
  - Java 8 or later, ooRexx 5.0 or later (hence “850”)
  - Package "BSF.CLS"
    - Camouflages Java as ooRexx (Java appears to be dynamic and message based)
    - Supplies class BSF and public routines
- "Everything that is available in Java becomes directly available to ooRexx !"
  - Java: "write once, run everywhere!"
    - Windows, MacOS, Linux, s/390 ...

# BSF.CLS: *Camouflages Java* as ooRexx

- ooRexx proxy class "**BSF**"
  - Allows to create Java objects
  - Requires the fully qualified Java class name
- Invoking Java methods
  - Just send the name of the method as a message to the Java object
    - Supply the arguments as documented, if any
      - Type conversions between ooRexx and Java are done automatically by BSF4ooRexx, if necessary
      - Return values are automatically converted by BSF4ooRexx, if necessary

# BSF.CLS: Creating Java Objects

- ooRexx proxy class "**BSF**"
  - Allows to create Java objects
  - Needs at least fully qualified Java class name
- Possible arguments for creating Java objects
  - Can be found by studying the "*Constructor*" section in the Javadocs
  - Supply the arguments as documented after the fully qualified Java class name argument
    - Type conversions ("marshalling") between ooRexx and Java are done automatically by BSF4ooRexx, if necessary

# BSF.CLS: *Camouflages Java* as ooRexx

- Allows to load any Java class
  - **bsf.loadClass(JavaClassName)**
    - Java class name
      - Use of the exact case is mandatory !
      - Java class name must be fully qualified !
- Allows accessing static (class) methods and fields (attributes)
  - Example uses java.lang.System's static getProperty() method to query the Java version from ooRexx


Prof. Rony G. Flatscher

# BSF.CLS: *Camouflages Java* as ooRexx

- Allows to import any Java class
  - **bsf.import(JavaClassName)**
    - Java class name
      - Use of the exact case is mandatory !
      - Java class name must be fully qualified !
- Imported Java class can be treated as if it were an ooRexx class
  - Allows to use the ooRexx "**new**"-method to create instances of the imported Java class
    - Possible arguments for creating Java objects can be found by studying the "Constructor" section in the Javadocs

# BSF.CLS: *Camouflages Java* as ooRexx

- Accessing, setting Java fields
  - ooRexx treats public fields as ooRexx attributes
  - Java "get" and "set" pattern methods for Java fields honored by BSF4ooRexx
    - Just use the field name following "get" and "set" only
  - Static fields can be accessed via the
    - Java class object or
    - Any of its instances

- About respecting case
  - Case of fully qualified Java class name
    - Always significant!
- Case of fields and method names insignificant!
  - Eases coding considerably

# BSF.CLS: Creating Java Arrays, 1

- Java arrays
  - Strictly typed
  - Fixed capacity
  - Indices start with value "**0**"
- Public routine "**bsf.createJavaArray(...)**"
  - Arguments
    - First argument gives the Java type
      - Fully qualified Java class name or Java class object
    - Each further argument is an integer value, denoting the maximum elements in that dimension

# BSF.CLS: Creating Java Arrays, 2

- Public routine "**bsf.createJavaArray(...)**"
  - Resulting Java array can be used as if it was an ooRexx array object!
    - Indices start at "**1**" as with ooRexx arrays!
    - Possesses the fundamental *ooRexx array methods* like "**AT**", "**[]**", "**PUT**", "**[]=**", "**supplier**", and "**makeArray**"
    - Can be therefore used in ooRexx "**DO ... OVER**" and "**DO WITH ... OVER**" loops


Prof. Rony G. Flatscher

# BSF4ooRexx: BSFCreateRexxProxy, 1

- RexxProxy
  - A *Java object* that proxies an ooRexx object
  - Allows Java to send messages to ooRexx objects
  - Any method invocations on the Java object will be forwarded as an ooRexx message to the proxied ooRexx object
    - All arguments supplied to the Java method are forwarded in the same sequence with the ooRexx message
    - BSF4ooRexx always appends an additional argument, "**slotDir**" (an ooRexx directory object) to the ooRexx message, which will contain information about the Java method invocation

# BSF4ooRexx: BSFCreateRexxProxy, 2

- RexxProxy
  - **BSFCreateRexxProxy(rexxObj [, userData])**
    - Creates and returns a Java object that proxies "**rexxObj**"
    - If "**userData**" (any Rexx object) supplied, then it will be added to the "**slotDir**" directory
  - **BSFCreateRexxProxy(rexxObj [, [userData], jiClz[, ...]])**
    - "**jiClz**" can be one or more Java interface classes the returned RexxProxy can be used for!
  - **BSFCreateRexxProxy(rexxObj [, [userData], jaClz[, arg[,...]])**
    - "**jaClz**" is an abstract Java class, "**arg**" can be one or more arguments for creating an instance of it

# Hints for Mapping Java into ooRexx

- Java import statements
  - Meant for the Java compiler, reveal the fully qualified name of classes/packages
- Java can only define classes with fields and methods
  - The static method main(...) usually is the “main program”, if defined
  - Just map/transcribe that code
  - Replace dots (.) with tildes (~)
- The ooRexx program needs to require the BSF.CLS package
  - Establishes the bridge between ooRexx and Java
  - Camouflages Java classes and Java objects as ooRexx classes and ooRexx objects
    - One can just send ooRexx messages to Java (class) objects, which is easy :)
  - Adds support to simplify various Java-specific concepts like iterating over collections

# Examples Iterating Over Java Collections

- Java uses different means to iterate over Java collections
  - Array notation
  - Java Enum classes
  - The Enumeration, Iterable (Iterator), and Map interfaces, using for or while loops
    - Streams and lambda functions got introduced in addition with Java 8
- ooRexx allows for looping with the do ... over or loop ... over keyword statements over ooRexx collections
  - BSF4ooRexx extends this ability to Java collections and Java Enum classes

# Iterating Over Java Collections
## Example 1, Arrays

```java
import java.util.Arrays;

public class DemoDoOver_Array
{
    public static void main (String args[])
    {
        String[] stringArray = {"Apple", "Banana", "Cherry"};
        System.out.println("Index and value:");
        for (int i = 0; i < stringArray.length; i++)
        {
            System.out.println("idx="+i+" value: "+stringArray[i]);
        }

        System.out.println("\nOnly values:");
        for (String str : stringArray)
        {
            System.out.println(str);
        }

        System.out.println("\nonly values, using stream:");

        Arrays.stream(stringArray).forEach(System.out::println);
    }
}
```

```
clzString="java.lang.String"
stringArray=bsf.createJavaArrayOf(clzString, "Apple", "Banana", "Cherry")
say "Index and value:"
do i=1 to stringArray~items
   say "idx="i "value:" stringArray[i]
end

say; say "only values:"
do str over stringArray
   say str
end

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

Output (Java):

```
Index and value:
idx=0 value: Apple
idx=1 value: Banana
idx=2 value: Cherry

Only values:
Apple
Banana
Cherry

only values, using stream:
Apple
Banana
Cherry
```

Output (ooRexx):

```
Index and value:
index=1 value: Apple
index=2 value: Banana
index=3 value: Cherry

Only values:
Apple
Banana
Cherry
```

# Iterating Over Java Collections
## Example 2, Enum Class

```java
// DemoDoOver_Enum.java
import org.oorexx.misc.OldGreekAlphabetEnum;    // an Enum class

public class DemoDoOver_Enum
{
    public static void main (String args[])
    {
        for (OldGreekAlphabetEnum e : OldGreekAlphabetEnum.values())
        {
            System.out.println("ordinal: "+e.ordinal()+" name: "+e.name());
        }
    }
}
```

```
clzEnum = bsf.importClass("org.oorexx.misc.OldGreekAlphabetEnum")
do e over clzEnum~values
   say "ordinal:" e~ordinal "name:" e~name
end

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

```
-- demoDoOver_Enum.rxj
clzEnum=bsf.loadClass("org.oorexx.misc.OldGreekAlphabetEnum")

do with index o item n over clzEnum -- iterate over enum values
  say "ordinal:" o "name:" n~name
end

::requires "BSF.CLS" -- get Java support
```

Output:

```
ordinal: 0 name: ALPHA
ordinal: 1 name: BETA
ordinal: 2 name: GAMMA
ordinal: 3 name: DELTA
ordinal: 4 name: EPSILON
ordinal: 5 name: ZETA
ordinal: 6 name: ETA
ordinal: 7 name: THETA
ordinal: 8 name: IOTA
ordinal: 9 name: KAPPA
ordinal: 10 name: LAMBDA
ordinal: 11 name: MY
ordinal: 12 name: NY
ordinal: 13 name: XI
ordinal: 14 name: OMIKRON
ordinal: 15 name: PI
ordinal: 16 name: RHO
ordinal: 17 name: SIGMA
ordinal: 18 name: TAU
ordinal: 19 name: YPSILON
ordinal: 20 name: PHI
ordinal: 21 name: CHI
ordinal: 22 name: PSI
ordinal: 23 name: OMEGA
```

# Iterating Over Java Collections
## Example 3, Enumeration Interface

```java
// DemoDoOver_Enumeration.java
import java.util.Enumeration;
import java.util.StringTokenizer;   // implements java.util.Enumeration

public class DemoDoOver_Enumeration
{
    public static void main (String args[])
    {
        String s = "Rexx, Regina, NetRexx, ooRexx, CREXX are great!";
        StringTokenizer st = new StringTokenizer(s);

        for ( ; st.hasMoreElements();)
        {
            System.out.println(st.nextElement());
        }
    }
}
```

```
 s = "Rexx, Regina, NetRexx, ooRexx, CREXX are great!"
 st = .bsf~new("java.util.StringTokenizer", s)
 do while st~hasMoreElements
    say st~nextElement
 end

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

```
-- demoDoOver_Enumeration.rxj
s = "Rexx, Regina, NetRexx, ooRexx, CREXX are great!";
tokenized=.BSF~new("java.util.StringTokenizer", s)
do v over tokenized
   say v
end

::requires "BSF.CLS" -- get Java support
```

Output:

```
Rexx,
Regina,
NetRexx,
ooRexx,
CREXX
are
great!
```

# Iterating Over Java Collections
## Example 4a, Iterable Interface

```java
// DemoDoOver_Iterator.java
import java.util.ArrayList; // implements Iterable interface
import java.util.Iterator;

public class DemoDoOver_Iterator
{
    public static void main (String args[])
    {
        ArrayList<String> al = new ArrayList<>();
        al.add("Rexx is great!");
        al.add("Regina is great!");
        al.add("NetRexx is great!");
        al.add("ooRexx is great!");
        al.add("BSF4ooRexx is great!");
        al.add("CREXX is great!");

        Iterator<String> iter = al.iterator();
        while (iter.hasNext())
        {
            System.out.println(iter.next());
        }
    }
}
```

```
al = .bsf~new("java.util.ArrayList")
al~add("Rexx is great!")
al~add("Regina is great!")
al~add("NetRexx is great!")
al~add("ooRexx is great!")
al~add("BSF4ooRexx is great!")
al~add("CREXX is great!")

iter = al~iterator
do while iter~hasNext
   say iter~next
end

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

Output:

```
Rexx is great!
Regina is great!
NetRexx is great!
ooRexx is great!
BSF4ooRexx is great!
CREXX is great!
```


Prof. Rony G. Flatscher

# Iterating Over Java Collections

## Example 4b, Iterable Interface

```java
// DemoDoOver_Iterator.java
import java.util.ArrayList; // implements Iterable interface
import java.util.Iterator;

public class DemoDoOver_Iterator
{
    public static void main (String args[])
    {
        ArrayList<String> al = new ArrayList<>();
        al.add("Rexx is great!");
        al.add("Regina is great!");
        al.add("NetRexx is great!");
        al.add("ooRexx is great!");
        al.add("BSF4ooRexx is great!");
        al.add("CREXX is great!");

        Iterator<String> iter = al.iterator();
        while (iter.hasNext())
        {
            System.out.println(iter.next());
        }
    }
}
```

```
-- demoDoOver_Iterator.rxj
al=.BSF~new("java.util.ArrayList")  -- create an ArrayList add
values to it
al~add("Rexx is great!")
al~add("Regina is great!")
al~add("NetRexx is great!")
al~add("ooRexx is great!")
al~add("BSF4ooRexx is great!")
al~add("CREXX is great!")

do v over al
   say v
end

::requires "BSF.CLS" -- get Java support
```

Output:

```
Rexx is great!
Regina is great!
NetRexx is great!
ooRexx is great!
BSF4ooRexx is great!
CREXX is great!
```


Prof. Rony G. Flatscher

# Iterating Over Java Collections
## Example 5, Map Interface

```java
// DemoDoOver_Map.java
import java.util.HashMap;

public class DemoDoOver_Map
{
    public static void main (String args[])
    {
        HashMap<String,String> hm = new HashMap<>();
        hm.put("key4", "ooRexx is great!");
        hm.put("key5", "CREXX is great!");

        // via entrySet()
        System.out.println("using entrySet():");
        for (java.util.Map.Entry<String,String> entry : hm.entrySet())
        {
            System.out.println("key: "+entry.getKey()+" value: "+
                    entry.getValue());
        }
        // via keySet()
        System.out.println("\nusing keySet():");
        for (String key : hm.keySet())
        {
            System.out.println("key: "+key+" value: "+hm.get(key));
        }
        // lambda expression
        System.out.println("\n(Java) using forEach (lambda):");
        hm.forEach( (k, v)  -> System.out.println("key: "+k+" value: "+v));
    }
}
```

```
hm = .bsf~new("java.util.HashMap")
hm~put("key4", "ooRexx is great!")
hm~put("key5", "CREXX is great!")

say "using entrySet():"
do entry over hm~entrySet
   say "key:" entry~getKey "value:" entry~getValue
end

say; say "using keySet():"
do key over hm~keySet
   say "key:" key "value:" hm~get(key)
end

say; say "(ooRexx) using do with...over:"
do with index key item value over hm
   say "key:" key "value:" value
end

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

Output:

```
using entrySet():
key: key5 value: CREXX is great!
key: key4 value: ooRexx is great!

using keySet():
key: key5 value: CREXX is great!
key: key4 value: ooRexx is great!

Java: using forEach (lambda):   |  ooRexx: using do with...over:
key: key5 value: CREXX is great!
key: key4 value: ooRexx is great!
```

# Implementing Java Interface Classes

- Java defines interface classes that define
  - Constant fields (static fields)
  - Abstract methods that define the signature, but are not implemented
- Java classes can implement interface classes
  - They must impement all defined methods, the Java compiler checks that
  - Interface classes can be used as Java datatypes
    - Java objects from a class that implements such interface classes can be used as arguments to methods or as their return types of that interface class
  - BSF4ooRexx allows one to implement the abstract Java methods in an ooRexx class in form of normal ooRexx methods
    - Using BSFCreateRexxProxy() allows to define which Java interface classes an ooRexx object implements

# Implementing Java Interface Classes
## Example: Functional Interface Classes, 1

- Java 8 introduced functional interface classes
  - One function method is abstract and needs to be implementd
- Example implements two functional interface classes
  - java.util.function.Predicate, abstract method: boolean test(T t)
    - Returns true if string contains `'k'` or `'K'`, false else
  - java.util.function.Consumer, abstract method: void accept(T t)
    - Outputs the string, embedding it in three angle brackets
- Example uses `java.util.Collection`'s `stream()` method to apply these implemented functions to the elements of a string array


Prof. Rony G. Flatscher

# Implementing Java Interface Classes

## Example: Functional Interface Classes (Java), 2

```java
import java.util.Arrays;
import java.util.function.Predicate;
import java.util.function.Consumer;

public class DemoFunctionalInterface
{
    public static void main(String[] args)
    {
        String ws = "Just a bunch of words to test for killer items containing a k";
        String[] arrWords = ws.split(" ");
        DemoFunctionalInterfaceWorker dfiw = new DemoFunctionalInterfaceWorker();
        String[] arrFiltered = Arrays.stream(arrWords)
                                     .filter(dfiw)
                                     .toArray(String[]::new);
        for (String w : arrFiltered)
        {
            System.out.println(w);
        }
        System.out.println("-----------------------");
        Arrays.stream(arrWords).forEach(dfiw);
    }
}

… continued …
```

```java
… continued …
    // implements two functional methods
class DemoFunctionalInterfaceWorker implements Consumer<String>,
                                               Predicate<String>
{
    public void accept(String t)    // java.util.function.Consumer
    {
        System.out.println(">>" + t + "<<");
    }

    public boolean test(String t)   //
java.util.function.Predicate
    {
        return t.toLowerCase().indexOf('k') != -1;
    }
}
```

Output:

```
killer
k
-----------------------
>>Just<<
>>a<<
>>bunch<<
>>of<<
>>words<<
>>to<<
>>test<<
>>for<<
>>killer<<
>>items<<
>>containing<<
>>a<<
>>k<<
```

# Implementing Java Interface Classes

## Example: Functional Interface Classes (ooRexx), 3

```
ws = "Just a bunch of words to test for killer items containing a k"
refWordString=.bsf~new("java.lang.String", ws)
arrWords=refWordString~split(" ")   -- get Java array of words

clzArrays=bsf.loadClass("java.util.Arrays")
-- create Rexx worker object, wrap it up as a Java object that can serve
-- the two functional interface classes Predicate and Consumer
jWorker=BsfCreateRexxProxy(.worker~new, , "java.util.function.Predicate", -
                                          "java.util.function.Consumer")
sa=clzArrays~stream(arrWords) ~filter(jWorker)~toArray
loop w over sa
  say w
end
say "----------------------"
clzArrays~stream(arrWords) ~foreach(jWorker)

::requires BSF.CLS   -- get Java-support

::class Worker    -- implements two functional methods

   -- implements the interface java.util.function.Predicate's test method
::method test     -- will return .true for strings containing 'k', .false else
  use arg s       -- fetch the argument (from Java)
  return s~caselessPos('k')>0 -- return result

   -- implements the interface java.util.function.Consumer's accept method
::method accept   -- will show each string
  use arg s       -- fetch the argument (from Java)
  say ">>"s"<<"   -- enclose value in angle brackets
```

Output:

```
killer
k
----------------------
>>Just<<
>>a<<
>>bunch<<
>>of<<
>>words<<
>>to<<
>>test<<
>>for<<
>>killer<<
>>items<<
>>containing<<
>>a<<
>>k<<
```

# A More Complex Example

- Apache POI

  https://poi.apache.org/

  Allows for processing Microsoft Office files like Excel's .xls and .xlsx files

  - Can be exploited on non-Windows patforms thanks to Java

- Example from

  https://wi.wu.ac.at/rgf/rexx/tmp/poi_oorexx/

  - `readme.txt` contains all related information including links to the Java tutorials

    https://howtodoinjava.com/java/library/readingwriting-excel-files-in-java-poi-tutorial/

# A More Complex Example
## Creating an Excel File (Java), 1

```java
// WriteExcelDemo.java (edited)
import org.apache.poi.hssf.usermodel.HSSFWorkbook;  // .xls (older)
import org.apache.poi.xssf.usermodel.XSSFWorkbook;  // .xlsx (newer)
// import org.apache.poi.xssf.streaming.SXSSFWorkbook;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Cell;
import java.io.File;
import java.io.FileOutputStream;
import java.util.Map;
import java.util.TreeMap;
import java.util.Set;

public class WriteExcelDemo {
    public static void main (String args[])
    {
        // HSSFWorkbook workbook = new HSSFWorkbook();  // for .xsl (older) format
        XSSFWorkbook workbook = new XSSFWorkbook();     // for .xslx (newer) format
        //Create a blank sheet
        Sheet sheet = workbook.createSheet("Employee Data");
        //Prepare data to be written as an Object[]
        Map<String, Object[]> data = new TreeMap<String, Object[]>();
        data.put("1", new Object[] {"ID", "NAME", "LASTNAME"});
        data.put("2", new Object[] {1, "Amit", "Shukla"});
        data.put("3", new Object[] {2, "Lokesh", "Gupta"});
        data.put("4", new Object[] {3, "John", "Adwards"});
        data.put("5", new Object[] {4, "Brian", "Schultz"});

… continued …
```

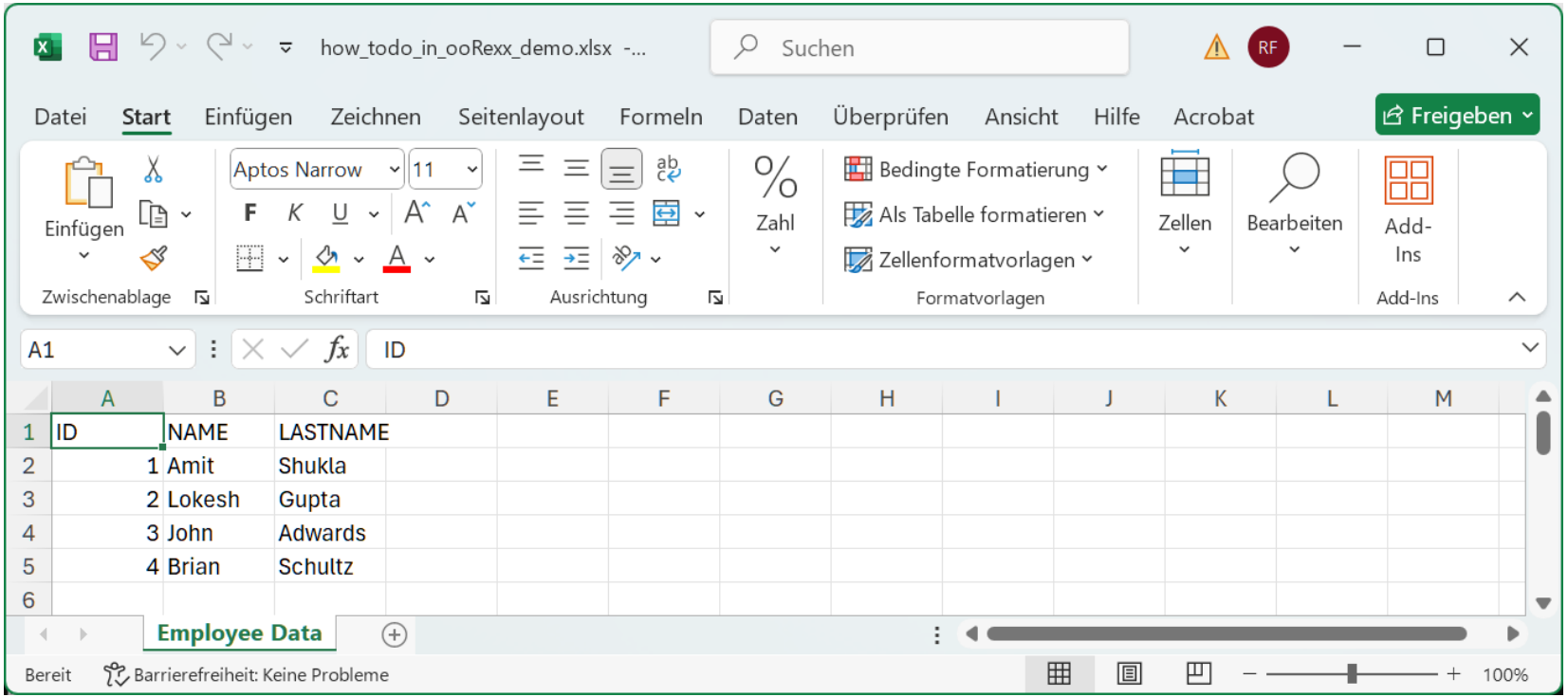
```java
… continued …

//Iterate over data and write to sheet
       Set<String> keyset = data.keySet();
       int rownum = 0;
       for (String key : keyset) {
         Row row = sheet.createRow(rownum++);
         Object [] objArr = data.get(key);
         int cellnum = 0;
         for (Object obj : objArr)
         {
            Cell cell = row.createCell(cellnum++);
            if(obj instanceof String)
                 cell.setCellValue((String)obj);
             else if(obj instanceof Integer)
                 cell.setCellValue((Integer)obj);
         }
       }
       //Write the workbook in file system
       try {
         // String ftype = ".xls";  // org.apache.poi.hssf.usermodel.HSSFWorkbook
         String ftype = ".xlsx";    // org.apache.poi.xssf.usermodel.XSSFWorkbook
         FileOutputStream out = new FileOutputStream(new File("howtodoinjava_demo"+ftype));
         workbook.write(out);
         out.close();
         System.out.println("howtodoinjava_demo"+ftype+" written successfully on disk.");
       }
       catch (Exception e) { e.printStackTrace();  }
    }
}
```

# A More Complex Example
## Creating an Excel File (ooRexx), 2

```
-- WriteExcelDemo.rxj
-- define data to be written to Excel
data = ( ("ID", "NAME",    "LASTNAME"), -
         (1   , "Amit",    "Shukla"  ), -
         (2   , "Lokesh",  "Gupta"   ), -
         (3   , "John",    "Adwards" ), -
         (4   , "Brian",   "Schultz" ) )
do ftype over ".xls", ".xlsx"
   -- create workbook
   if ftype=".xls" then
      workbook = .bsf~new("org.apache.poi.hssf.usermodel.HSSFWorkbook") -- .xsl
   else
      workbook = .bsf~new("org.apache.poi.xssf.usermodel.XSSFWorkbook") -- .xslx
   -- create a named sheet
   sheet = workbook~createSheet("Employee Data");
   -- iterate over data and write to sheet
   rownum = 0
   do objArr over data
      row = sheet~createRow(rownum)
      cellnum = 0
      do val over objArr
         cell = row~createCell(cellNum)
         if datatype(val,"number") then cell~setCellValue(box("double",val))
                                   else cell~setCellValue(val)
         cellNum+=1   -- next column
      end
      rowNum+=1    -- next row
   end
  -- save the workbook
   fname = "how_todo_in_ooRexx_demo"ftype
   out = .bsf~new("java.io.FileOutputStream", .bsf~new("java.io.File", fname))
   workbook~write(out)
   out~close
   say fname "written successfully on disk"
end
::requires "BSF.CLS"    -- get ooRexx-Java bridge
```


Prof. Rony G. Flatscher

# A More Complex Example
## Creating an Excel File, 3



# Roundup

- BSF4ooRexx850
  - Allows to use ooRexx instead of Java
  - Java programs can be mapped to ooRexx in a straightforward manner
- ooRexx can employ any Java class and interact with any Java object
- Questions?