Automatisierung von Java Anwendungen (8)

Bean Scripting Framework (BSF), 2

Nutzung von Java-Klassen z.B. Java-GUI,
JSR-223/BSF 3.0

Prof. Dr. Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Augasse 2-6 ■ A-1090 Wien

Exkurs grafische Benutzeroberflächen unter Java

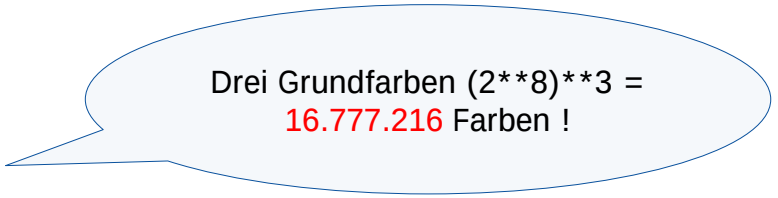
- GUI-Grundlagen unter Java
  - Komponenten
  - Ereignisse
  - Ereignisadapter
- BSF4Rexx-Beispiel
  - Abarbeitung von Ereignissen in Schleife
  - Nutzung von Java's awt von klassischem Rexx aus
  - Nutzung von Java's awt von Object Rexx aus

Grafische Benutzeroberflächen, 1

- Grafische Benutzerschnittstelle
  - Ausgabe
    - Grafikfähiger Bildschirm
    - Schwarz/weiß, Farbe
    - Sprache
  - Eingabe
    - Tastatur
    - Maus
    - Bildschirm
    - Stift
    - Sprache

Grafische Benutzeroberflächen, 2

- Bildschirmausgabe
  - Ansteuerung des Bildschirms
    - Jeder Bildpunkt ("pixel": von "picture element") einzeln
      - Zweidimensionales Koordinatenschema ("x", "y")
        - Auflösung z.B. 320x240, 640x480, 1024x768, 1280x1024, ...
      - Ursprung (Koordinate: "0,0")
        - Links oben (z.B. Windows)
        - Links unten (z.B. OS/2)
    - Farbe
      - Schwarz/weiß (1 Bit pro Bildpunkt)
      - Drei Grundfarben
        - Rot, grün, blau
        - Intensität von 0 bis 255
        - 1 Byte pro Grundfarbe (2**8)



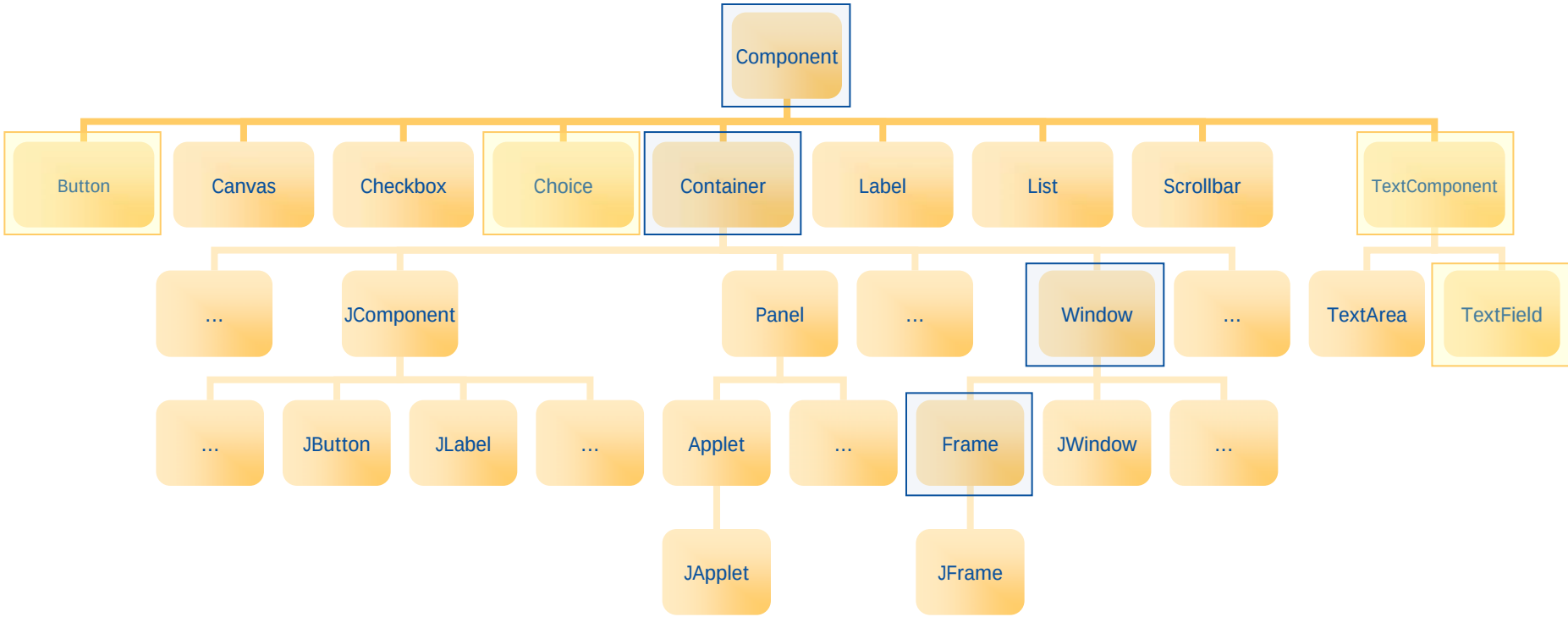
Grafische Benutzeroberflächen, 3

- Menge an Pixeln, Menge an Bytes
  - 640x480
    - 307.200 px = 300 Kpx
    - 38.400 Bytes (s/w) = 37,5 KB
    - 921.600 Bytes (Vollfarbe) = 900 KB
  - 1280x1024
    - 1.310.720 px = 1.280 Kpx
    - 163.840 Bytes (s/w) = 160 KB
    - 3.932.160 Bytes (Vollfarbe) = 3.840 KB = 3,75 MB
- Rechenaufwändig
  - ➔Aussehen einer jeden Komponente muss mit Pixeln ausprogrammiert werden!
    - Z.B. Farbpunkte, Rechtecke, Kreise, Rahmen, Schatten, Schriften, …
    - Aber auch Animationseffekte!

Grafische Benutzeroberflächen, 4

- Aufbau mit Elementen/Komponenten ("Component"s), z.B.
  - "Container"
    - Fenster ("Window")
      - Rahmen ("Frame")
    - "Panel"
  - "Button" (Druck-Knopf)
  - "Checkbox", "CheckboxGroup" (vergleichbar mit 'Radio-Buttons')
  - "Choice" (Auswahl-Feld)
  - "Image" (Bitmap-Feld)
  - Textfelder
    - "Label" (nur Ausgabe)
    - "TextField" (Ein- und Ausgabe-Feld)
    - "TextArea" (Ein- und Ausgabe-Feld, mehrzeilenfähig)
  - Listfelder ("List"), Bildlaufleisten ("Scrollbar"), Zeichenfelder ("Canvas"), ...

Grafische Benutzeroberflächen, 5



Grafische Benutzeroberflächen, 6

- "Component"
  - Können Ereignisse erzeugen, z.B. "ActionEvent", "KeyEvent", "MouseEvent", ...
  - Können "EventListener" entgegennehmen und ihnen die Ereignisse anzeigen, sobald sie anfallen, indem die entsprechenden Methoden der "EventListener"-Objekte aufgerufen werden
  - Können in "Container"n positioniert werden
- "Container"
  - Eine Graphik-Komponente vom Typ "Component"
  - Können andere Graphik-Komponenten aufnehmen
    - Aufgenommene "Component"s können z.B. auch vom Typ "Container" sein
  - Können enthaltene Graphik-Komponenten mit Hilfe von Layout-Managern positionieren und verwalten
- "Frame"
  - Spezialisierung von "Window" (eine "Container"-Klasse)
  - Fügt einen Rahmen und eine Titelleiste zu einem Fenster hinzu

"Hallo, liebe Welt", graphisch (Java)

```
import java.awt.*;

class HalloWelt
{
    public static void main (String args[])
    {
        Frame f = new Frame("Hallo, liebe Welt!");
        f.show();
    }
}
```


"Hallo, liebe Welt", graphisch (Rexx)

```
if rxFuncQuery("BSF") = 1 then   /* BSF() support not loaded yet ? */
do
   call rxFuncAdd "BsfLoadFuncs", "BSF4Rexx", "BsfLoadFuncs"
   call BsfLoadFuncs    /* register the BSF* external functions   */
   call BsfLoadJava     /* load Java, we need it!                 */
end

call BSF 'registerBean', 'win', 'java.awt.Frame', 'Hallo, liebe Welt - von Rexx aus.'
call BSF 'invoke', 'win', 'show'
call SysSleep 10
call BSF 'unregisterBean', 'win'
```

"Hallo, liebe Welt", graphisch (Object Rexx)

```
.bsf~new('java.awt.Frame', 'Hallo, liebe Welt - von Object Rexx aus.') ~show

call SysSleep 10

::requires BSF.CLS
```

"ShowCount.rex"
GUI mit Ereignisabarbeitung

```
call BSF 'registerBean',     'win', 'java.awt.Frame', 'Show count'
call BSF 'addEventListener', 'win', 'window', 'windowClosing', 'call BSF "exit"'

call BSF 'registerBean',     'but', 'java.awt.Button', 'Press me!'
call BSF 'addEventListener', 'but', 'action', '', 'call ShowSize'

call BSF 'registerBean',     'lab', 'java.awt.Label'
center=bsf('getStaticValue', 'java.awt.Label', 'CENTER')
call BSF 'invoke',           'lab', 'setAlignment', center

call BSF 'invoke', 'win', 'add', 'Center', 'lab'
call BSF 'invoke', 'win', 'add', 'South',  'but'
call BSF 'invoke', 'win', 'pack'
call BSF 'invoke', 'win', 'setSize', 300, 90
call BSF 'invoke', 'win', 'show'
call BSF 'invoke', 'win', 'toFront'
i=0             /* set counter to 0     */
do forever
   a = bsf("pollEventText")     /* wait for an eventText to be sent     */
   interpret a                  /* execute as a Rexx program    */
   if result= "SHUTDOWN, REXX !" then leave /* JVM will be shutdown in 0.1sec */
end
exit

ShowSize: /* show the actual number of times, you pressed the button      */
   i=i+1
   call BSF 'invoke', 'lab', 'setText', "Press #" i
   return
```

"ShowCount-oo.rex", Object Rexx GUI mit Ereignisabarbeitung

```
.bsf~bsf.import("java.awt.Frame", "awtFrame") --import Java class into ooRexx
win=.awtFrame~new("Show Count")  -- create an instance (a proxy for a Java object)
win~bsf.addEventListener('window', 'windowClosing', '.bsf~bsf.exit') -- add an event listener

but=.bsf~new("java.awt.Button", "Press me!")                -- use the proxy class "BSF"
but~bsf.addEventListener('action', '', 'call ShowSize')     -- add an event listener
center=.bsf~bsf.getStaticValue( 'java.awt.Label', 'CENTER' ) -- get value (1) for constant "CENTER"
lab=.awtLabel~new ~~setAlignment(center) -- create an instance of a Java class, set its alignment
   -- add the Java objects to the frame, pack its content, set its size, show it
win ~~add("Center", lab) ~~add("South", but) ~~pack ~~setSize(300,90) ~~show ~~toFront
i=0                            /* set counter to 0     */
do forever
   a = .bsf~bsf.pollEventText   /* retrive the string (if event occurred)     */
   interpret a                  /* execute received string as a Rexx program  */
   if result= "SHUTDOWN, REXX !" then leave /* JVM will be shutdown in 0.1sec */
end
exit

ShowSize:      /* show the actual number of times, you pressed the button     */
   i=i+1
   lab~setText("Press #" i)
   return

::requires BSF.CLS       -- get access to the Object Rexx support enhancement

   /* define an Object Rexx proxy class for the Java class "java.awt.Label" */
::class awtLabel subclass BSF  -- subclass the public class "BSF" (from file "BSF.CLS")

::method init           -- make sure this class refers to "java.awt.Label"
  self~init:super('java.awt.Label', ARG(1, 'A'))
```

Ereignisse, 1

- Zahlreiche Ereignisse denkbar und möglich, z.B.
  - "ActionEvent"
    - Wichtig für Komponenten bei denen nur eine einzige Aktion vorgesehen ist, z.B. "Button"
  - "ComponentEvent"
    - "FocusEvent"
    - "InputEvent"
      - "KeyEvent"
      - "MouseEvent"
    - "WindowEvent"

Ereignisse, 2

- Event-Schnittstellen sind definiert in den Interfaces vom Typ "EventListener"
  - Online-Dokumentation für Paket "java.util"
  - Wichtige "EventListener" für graphische Benutzerschnittstellen...
    - Interface "ActionListener"
      ```
      void actionPerformed (ActionEvent e)
      ```
    - Interface "KeyListener"
      ```
      void keyPressed  (KeyEvent e)
      void keyReleased (KeyEvent e)
      void keyTyped    (KeyEvent e)
      ```

Ereignisse, 3

- Wichtige "EventListener" für graphische Benutzerschnittstellen...
  - Interface "MouseListener"

```
void mouseClicked ( MouseEvent e)
void mouseEntered ( MouseEvent e)
void mouseExited  ( MouseEvent e)
void mousePressed ( MouseEvent e)
void mouseReleased( MouseEvent e)
```

  - Interface "WindowListener"

```
void windowActivated  ( WindowEvent e)
void windowClosed     ( WindowEvent e)
void windowClosing    ( WindowEvent e)
void windowDeactivated( WindowEvent e)
void windowDeiconified( WindowEvent e)
void windowIconified  ( WindowEvent e)
void windowOpened     ( WindowEvent e)
```

Ereignisse und Komponenten

- Komponenten verarbeiten Ereignisse
- Komponenten kann man "Listener"-Objekte übertragen, die vom Eintritt von Ereignissen verständigt werden
  - Registrierung eines "Listener"-Objekts erfolgt durch eine

```
void add...Listener( ...Listener listener)
  z.B.:
void addKeyListener   (KeyListener kl)
void addMouseListener (MouseListener ml)
```

  - Verständigung erfolgt durch den Aufruf der im Interface definierten Ereignis-Methoden, z.B.

```
kl.keyPressed   (e);
ml.mouseClicked (e);
```

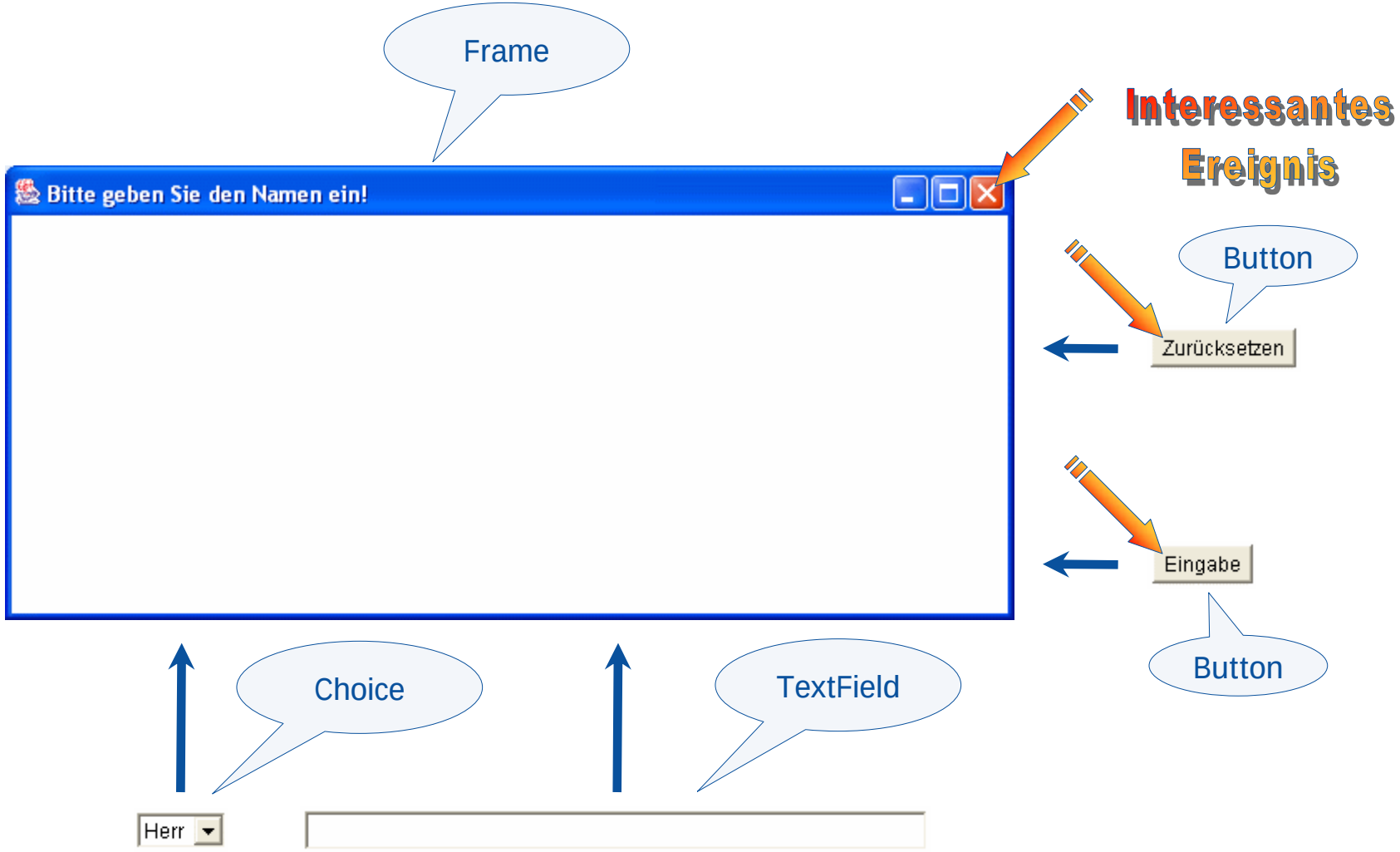
Beispiel "Eingabe", 1

- "TextField"
  - Eingabefeld, um einen Namen eingeben zu können
- "Choice"
  - Auswahlfeld für die Anrede: "**Herr**" bzw. "**Frau**"
- "Button": "Zurücksetzen"
  - Setzt die Eingabe zurück
- "Button": "Eingabe"
  - Akzeptiert die Eingabe
  - Anrede und Eingabefeld werden ausgelesen und auf "System.out" ausgegeben

Beispiel "Eingabe", 2

- Überlegungen
  - Welche awt-Klassen?
    - "Frame", "Choice", "TextField", "Button"
  - Welche Ereignisse?
    - Fenster schließen
      - Ereignismethode "windowClosing" aus "WindowListener"
      - Benutzung einer Adapterklasse
        - Sonst müssten wir alle sieben Ereignismethoden implementieren!
    - Drücken des jeweiligen "Button"s
      - Ereignismethode "actionPerformed" aus "ActionListener"
    - Alle anderen Ereignisse sind für diese Aufgabenstellung vollkommen uninteressant und werden daher von uns ignoriert!

Beispiel "Eingabe", 3



"Eingabe.java", anonyme Java-Klasse

```
import java.awt.*; import java.awt.event.*;

class Eingabe
{
    public static void main (String args[])
    {
        Frame     f  = new Frame("Bitte geben Sie den Namen ein!");
        f.addWindowListener( new WindowAdapter()
            { public void windowClosing( WindowEvent e) { System.exit(0); } } );
        f.setLayout(new FlowLayout());  // Layout Manager
        final Choice cf = new Choice(); cf.add("Herr"); cf.add("Frau");
        f.add(cf);           // Komponente wird dem Container hinzugefügt
        final TextField tf = new TextField("", 50);   // 50 Zeichen Platz
        f.add(tf);           // Komponente wird dem Container hinzugefügt
        Button    bNeu = new Button("Zurücksetzen");
        f.add(bNeu);         // Komponente wird dem Container hinzugefügt
        bNeu.addActionListener( new ActionListener ()
            { public void actionPerformed(ActionEvent e) { tf.setText(""); } } );
        Button    bOK  = new Button("Eingabe");
        f.add(bOK);         // Komponente wird dem Container hinzugefügt
        bOK.addActionListener( new ActionListener ()
            { public void actionPerformed(ActionEvent e) {
                System.out.println(cf.getSelectedItem()+" "+tf.getText());
                System.exit(0); }
            } );
        f.pack(); f.show();
    }
}
```

"Eingabe-oo.rex", Object Rexx


```
f=.bsf~new("java.awt.Frame", "Bitte geben Sie den Namen ein!")        -- create the frame object
f~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"') -- add event listener
f~setLayout( .bsf~new("java.awt.FlowLayout") ) -- create FlowLayout object and assign it
cf=.BSF~new("java.awt.Choice")    -- create Choice object
cf ~~add("Herr") ~~add("Frau")    -- add options
f~add(cf)         -- add Choice object to Frame object
tf=.bsf~new("java.awt.TextField", "", 50)  -- create TextField, show 50 chars
f~add(tf)         -- add TextField object to Frame object
but=.bsf~new('java.awt.Button', 'Zurücksetzen') -- create Button object
f~add(but)        -- add Button object to Frame object
but~bsf.addEventListener('action', '', ' tf~setText("") ') -- add event listener
but=.bsf~new('java.awt.Button', 'Eingabe') -- create another Button object
f~add(but)        -- add Button object to Frame object
but~bsf.addEventListener( 'action', '', 'call done cf, tf') -- add event listener
f ~~pack ~~show ~~toFront  -- layout the Frame object, show it, make sure it is in front
do forever
   INTERPRET .bsf~bsf.pollEventText  -- get eventText, interpret it as Rexx code
   if result="SHUTDOWN, REXX !" then leave  -- Java will be exited, leave Rexx
end
exit

   /* called, if the "done" button is pressed and the according eventText gets sent */
done: procedure
   use arg cf, tf
   say cf~getSelectedItem tf~getText
   return .bsf~bsf.exit      /* shutdown JVM in .1sec, in case this program was started via Java  */

::requires BSF.CLS    -- load Object Rexx BSF support
```

JSR-223, 1

- JCP ("Java Community Process") Arbeitsgruppe
- JSR ("Java Specification Request") von Sun verwaltet
- JSR-223, "Java Scripting Specification"
  - Entwicklung eines Java-Standards für die Anbindung von Skripten
  - Ursprünglich von Sun nur gedacht für
    - Anwendungsschwerpunkt WWW-Seiten
    - Einbindung vor allem von PHP
  - Homepage: http://www.jcp.org/en/jsr/detail?id=223
- BSF als Modell und Grundlage für die Spezifikation
  - Mitarbeit als "technischer Experte", u.a. wegen "BSF4Rexx"
  - Wirtschaftsuniversität Wien offiziell im JSR-223-Standard angeführt!

JSR-223, 2

- Seit November 2006 Bestandteil von Sun's Java Version 6 (1.6) !
  - Java-Paket **javax.script**
- Apache Software Foundation (ASF), Jakarta-Projekt
  - BSF-Projekt
    - BSF 2.4 offiziell im Herbst 2007 veröffentlicht
      - Vorher offiziell im "Beta"-Status, aber bereits weit verbreitet/benutzt
  - Opensource-Implementierung von JSR-223,
    - "BSF 3.0" offiziell in Beta seit April 2007
    - Seit Winter 2007/08 im Apache "Harmony"-Projekt benutzt
      - Opensource Java 1.5, Implementierung von Java 6 fast fertig