The Making of Jugs
(Entwurf)

Sven Eric Panitz
TFH Berlin
Version 5. März 2004

Dieses Papier beschreibt die Implementierung eines Javainterpreters. Es ist als XML-Dokument geschrieben und enthält den kompletten Quelltext des Programms. Per XQuery werden die Javadateien und das druckbare Papier erzeugt.

Derzeit ist das Papier in einem rohen Entwurfsstadiums. Es nicht abzusehen, wann es eine für jedermann gut aufgearbeitete Version geben wird. Ich habe mich frei nach einem Prinzip aus der Schule des *extreme programming* entschieden, so bald es etwas gibt, es öffentlich zu machen und zur Diskussion zu stellen; und somit den Lehrsatz: *der Feind des Guten ist das Bessere* Lügen zu strafen.

Inhaltsverzeichnis

1 Einführung

1.1 Kompilieren und Interpretieren

Programmiersprachen können in ihrem Ausführungsmodell in zwei grobe Klassen eingeteilt werden:

- **kompiliert** (C, Cobol, Fortran): in einem Übersetzungsschritt wird aus dem Quelltext direkt das ausführbare Programm erzeugt, das dann unabhängig von irgendwelchen Hilfen der Programmiersprache ausgeführt werden kann.
- **interpretiert** (Lisp, Scheme): der Programmtext wird nicht in eine ausführbare Datei übersetzt sondern durch einen Interpreter Stück für Stück anhand des Quelltextes ausgeführt. Hierzu muß stets der Interpreter zur Verfügung stehen, um das Programmm auszuführen. Interpretierte Programme sind langsamer in der Ausführung als übersetzte Programme.

Es gibt Programmiersprachen für die sowohl Interpreter als auch Übersetzer zur Verfügung stehen, z.B. Haskell. In diesem Fall wird der Interpreter gerne zur Programmentwicklung benutzt und der Übersetzer erst, wenn das Programm fertig entwickelt ist.

Java benutzt ein Ausführungsmodell, daß im Prinzip eine Mischform aus beiden obigen ist. Es gibt einen Übersetzer, der ein Programm in einen abstrakten Maschinencode kompiliert und es gibt einen Interpreter, der diesen abstrakten Maschinencode interpretiert und damit zur Ausführung bringt. Man spricht von byte-code, der interpretiert wird. Andere Programmiersprachen, die so verfahren, sind z.B. Caml oder auch Lisp.

Damit könnte man sagen, daß Java das beste aus beiden Welten anbietet. Effizienz wie bei einer kompilierten Sprache und Flexibilität wie bei einer interpretierten Sprache.

1.2 Lesen-Auswerten-Schleife

Ein klassischer Interpreter hat eine äußere Schleife, in der eine Eingabe gelesen wird, diese als Programm ausgewertet wird und das Ergebnis dann auf irgendeine Weise ausgegeben wird.

Die kleinste ladbare Einheit in Java sind Klassen. Daher werden wir eine Klasse benötigen, die für die Eingabe erzeugt wird, dann geladen und in der es eine Methode gibt, die schließlich ausgeführt wird. Wir sehen für diese Klasse eine allgemeine Schnittstelle vor.

MainJugsTestClassPrototype.java

```
1 package name.panitz.crempel.tool.jugs;
2 public interface MainJugsTestClassPrototype{
3   public void run() ;
4   static java.util.Map<String,Object> map
5     = new java.util.HashMap<String,Object>();
6 }
```

In der Methode `run` wird sich der auszuführende Code befinden. Die in einem statischen Feld enthaltene Abbildung soll ermöglichen, bestimmte Objekte unter einem Namen über mehrere Auswertungsdurchläufe abzuspeichern.

1.3 The Name of the Game

Der Name *Jugs* steht für *J*ava *U*m*g*ebungs *S*ystem. Er leitet sich von den Namen eines weit verbreiteten Interpreters für die Programmiersprache *Haskell* her: *Hugs*; wobei *Hugs* für *Haskell User Gofer System* steht. *Gofer* war ein Haskelldialekt mit dazugehörigen Interpreter.

Darüberhinaus bedeutet *to juggle* jonglieren.[1] Der Interpreter jongliert in gewisser mit Java und hält die Bälle in der Luft.

2 Klassen Laden

Das Kernstück des Interpreters wird ein eigener kleine Klassenlader darstellen. Immer wenn ein Ausdruck zum Auswerten oder Befehle zum Ausführen eingegeben werden, dann wird eine Java-Datei erzeugt. Diese wird vom Javaübersetzer übersetzt, so daß eine Klassendatei entsteht. Die erzeugte Klasse ist danach erneut zu laden. Jedesmal wird diese Klasse einen

[1] Während ein *jug* nach meinem Wörterbuch ein Kaffekännchen und wo Java doch Kaffee ist, scheint es ein treffender Name zu sein.

neuen Inhalt enthalten. Daher ist sie neu zu laden. Wir benötigen daher einen Klassenlader, der eine Klasse erneut lädt, obwohl sie bereits geladen wurde. Erfreulicher Weise läßt sich so etwas in Java realisieren. Hierzu erweitern wir die Klasse `URLClassLoader`.

JugsClassLoader.java

```
1  package name.panitz.crempel.tool.jugs;
2
3  import java.net.URL;
4  import java.net.URLClassLoader;
5
6  public class JugsClassLoader extends URLClassLoader {
7    public JugsClassLoader(URL []urls){super(urls);}
```

Wir überschreiben die Methode `loadClass`, so daß die Klasse direkt mit der Methode `findClass` der Oberklasse geladen wird. Damit umgehen wir, daß die Oberklasse darauf verzichtet, bereits geladene Klassen erneut zu laden. Lediglich Klassen aus Paken die mit `java` beginnen und Klassen aus dem eigenen Paket vun Jugs sind hiervon ausgenommen.

JugsClassLoader.java

```
8    public Class loadClass(String cl) throws ClassNotFoundException{
9      if (!(       cl.startsWith("java")
10                ||cl.startsWith("name.panitz.crempel.tool.jugs")))
11       return super.findClass(cl);
12     return super.loadClass(cl);
13   }
```

Die Methode `addUrl` überschreiben wir mit einer allgemeineren Sichtbarkeit.

JugsClassLoader.java

```
14   public void addURL(URL url) {super.addURL(url);}
15 }
```

Damit ist die wichtigste Komponente von Jugs bereits geschrieben.

3 Kommandozeilen Schnittstelle

In diesem Abschnitt schreiben wir zunächst eine einfache Kommandozeilen basierte Version von Jugs.

Jugs.java

```
1  package name.panitz.crempel.tool.jugs;
2
3  import java.io.*;
4
5  import java.net.URLClassLoader;
6  import java.net.URL;
7  import java.lang.reflect.Method;
8
9  import java.util.List;
```

```
10 import java.util.ArrayList;
11 import java.util.ResourceBundle;
12
13 public class Jugs {
```

Wir sehen einen festen Namen für die jeweils neu generiert und dann ausgeführte Klasse vor:

Jugs.java

```
14   static final public String MAIN_JUGS_CLASS="MainJugsTestClass";
```

Die generierte Klasse soll jeweils in das aktuelle Benutzerverzeichnis geschrieben werden, welches wir uns einmal vom System geben lassen.

Jugs.java

```
15   final public String USER_DIR
16    = System.getProperty("user.dir")+"/";
```

Sämtliche textuellen Ausgaben an den Benutzer sollen lokalisierbar sein und werden daher in einer Resource-Datei geschrieben, die zunächst geladen wird.

Jugs.java

```
17   static ResourceBundle labels = ResourceBundle
18   .getBundle("name.panitz.crempel.tool.jugs.JugsInfo");
```

In Java gibt es Ausdrücke, die einen Wert auf dem Stabel zurücklassen und Befühle. Für beide Programmteile sehen wir einen eigenen Modus in Jugs vor. Ein internes Flag markiert, in welchem Modus sich die aktuelle Jugs-Instanz gerade befindet. Der Standardmodus sei der zum Auswerten von Ausdrücken.

Jugs.java

```
19   boolean expressionMode = true;
```

3.1 Hauptschleife

Die eigene Hauptschleife zum Lesen und Auswerten von Ausdrücken respektive Befehlen läßt sich erfrischen einfach implementieren. Wir schreiben hierzu die Java `main`-Methode. Zunächst wird eine Instanz von `Jugs` erzeugt, eine Willkommensmeldung an den Benutzer ausgegeben und in die Hauptschleife eingestiegen. In dieser wird ein Prompt ausgegeben und eine Zeile von der Eingabekonsole gelesen. Wir sehen Steuerkommandos vor, die mit einem Doppelpunkt beginnen. Diese werden abgefangen, ansonsten wird die eingegebene Zeile zur Ausführung gebracht.

Jugs.java

```
20   public static void main(String [] args){
21     Jugs jugs = new Jugs();
22     System.out.println(labels.getString("WELCOME"));
23     while (true){
24       try {
```

```
25        System.out.print(">\u0020");
26        String input
27         =  new BufferedReader(
28                  new InputStreamReader(System.in)).readLine();
29
30        if (input.startsWith(":")) handleCommand(input,jugs);
31        else jugs.execute(input);
32      }catch (IOException e){
33        System.out.println(e);
34      }
35    }
36  }
```

Eventuelle Fehlerfälle werden abgefangen und auf der Konsole bekannt gegeben.

3.2 Steuerkommandos

Ein paar rudimentäre Steuerbefahle, die alle mit einem Doppelpunkt beginnen, sind in Jugs vorgesehen. Die folgende Methode reagiert auf diese Befehle:

Jugs.java

```
37  private static void handleCommand(String input,Jugs jugs){
38    if (input.startsWith(":q")) System.exit(0);
39    else if (input.startsWith(":s"))
40      jugs.expressionMode=false;
41    else if (input.startsWith(":e"))
42      jugs.expressionMode=true;
43    else if (input.startsWith(":h") || input.startsWith(":ÿ"))
44      System.out.println(labels.getString("COMMAND_HELP"));
45    else
46     System.out.println(labels.getString("UNKNOWN_COMMAND")+input);
47  }
```

3.3 Ausführung

Das Herzstück von Jugs stellt natürlich die Auswertung dar. Die Methode `execute` erhält hierzu einen Strin, der den auszuführenden Javaquelltext enthält. Zusötzlich kann dieser Methode noch eine Liste von zu importierenden Klassen und Paketen mitgereicht werden. Als Standardwert wird hierfür die leere Liste verwendet:

Jugs.java

```
48  public void execute (String input){
49    execute(input,new ArrayList<String>());
50  }
```

Zum Ausführen des Codes müssen wir eine Datei generieren, die diesen Code in einer Methode `run` enthält.

Jugs.java
```
51  public void execute (String input,List<String> imports){
52    try{
53      final String newClass = mkClass(input,imports);
54      final FileWriter writeClass
55        = new FileWriter(USER_DIR+MAIN_JUGS_CLASS+".java");
56      writeClass.write(newClass,0,newClass.length());
57      writeClass.flush();
```

Dier dergestalt generierte Klasse ist schließlich durch den Javaübersetzer zu übersetzen. Das setzt jetzt voraus, daß wir die Datei `tools.jar` in unserem Klassenpfad aufgenommen haben!

Jugs.java
```
58      String [] javacArg
59        = {"-source","1.5"
60          ,USER_DIR+MAIN_JUGS_CLASS+".java"};
61
62      final int erg = com.sun.tools.javac.Main.compile(javacArg);
```

Damit haben wir im besten Falle eine Klassendatei generiert bekommen. Diese ist jetzt zu laden und auszuführen. Jetzt können wir unser eigenen Klassenlader endlich in Aktion setzen. Diesem initialisieren wir mit Einträgen des Klassenpfads.

Jugs.java
```
63      if (erg==0){
64        String classPath = System.getProperty("java.class.path");
65
66        URL [] urls = { new URL("file://localhost/"+USER_DIR+"/")};
67
68        JugsClassLoader loader = new JugsClassLoader(urls);
69
70        java.util.StringTokenizer st
71          = new java.util.StringTokenizer
72              (classPath
73              ,System.getProperty("path.separator"));
74
75        while (st.hasMoreTokens()) {
76          try {
77            loader.addURL(new URL("file://localhost"+st.nextToken()));
78          }catch (java.net.MalformedURLException _){}
79        }
```

Nun ist er bereit, die generierte Klasse zu laden, so daß wir von ihr eine Instanz erzeugen können. Über Reflektion wird die Methode `run` auf diese Instanz zur Ausführung gebracht.

Jugs.java
```
80        Object o = loader.loadClass("MainJugsTestClass").newInstance();
81        Class cl = o.getClass();
82
```

```
83          final Class [] emptyC = {};
84          Method m = cl.getMethod("run",emptyC);
85          final Object [] empty = {};
86          m.invoke(o,empty);
87        }
```

Bei der Vielzahl von Techniken, die wir verwendet haben, können viele unterschiedliche Ausnahmen auftreten. Wir gönnen uns den Luxus diese alle einzaln aufzulisten:

Jugs.java

```
88      }catch (java.lang.reflect.InvocationTargetException e){
89        System.out.println(e);
90      }catch (NoSuchMethodException e){
91        System.out.println(e);
92      }catch (ClassNotFoundException e){
93        System.out.println(e);
94      }catch (InstantiationException e){
95        System.out.println(e);
96      }catch (IllegalAccessException e){
97        System.out.println(e);
98      }catch (IOException e){
99        System.out.println(e);
100     }catch (ClassFormatError e){
101       System.out.println(e);
102     }
103   }
```

3.4 Klassengenerierung

Diesem Abschnitt ist schließlich zu entnehmen, was für eine Klasse für den auszuführenden Code erzeugt wird.

Jugs.java

```
104   public String mkClass(String content,List<String> imports){
105     String importsString = "";
106     for (String imp:imports)
107       importsString = importsString+"import "+imp+";\n";
```

Für die zwei verschiedenen Modi von Jugs werden recht unterschiedliche Klassen generiert. Für Ausdrücke wird der Ausdruck der Methode `println` übergeben.

Jugs.java

```
108     if (expressionMode)
109      return importsString
110      +"public class "
111      + MAIN_JUGS_CLASS
112      +" implements name.panitz.crempel.tool.jugs"
113      +".MainJugsTestClassPrototype {\n"
114      +"  public void run() {\n"
```

```
115       +"    try{System.out.println("+content+");}\n"
116       +"    catch(Exception e){System.out.println(e);}}}";
```

Befehle können direkt in den Methodenrumpf eingefügt werden.

Jugs.java

```
117     else return importsString
118          +"public class "
119          + MAIN_JUGS_CLASS
120          +" implements name.panitz.crempel.tool.jugs."
121                                  +"MainJugsTestClassPrototype {\n"
122          +"  public void run(){try{"+content+"}\n"
123          +"    catch(Exception e){System.out.println(e);}}}";
124   }
```

Jugs.java

```
125   static public final String VERSION = "0.3.1";
126 }
```

3.5 Dialogtexte

Damit können wir jetzt interaktiv Javafragmente austesten. Die folgende Session gibt ein kleines Beispiel hierfür.

JugsInfo.properties

```
1
2  WELCOME =\     __ __  __  _____   ___     ____________________________________
3  \     || ||  || ||  || ||__      Jugs: the interactive Java interpreter\n\
4  \     || ||__|| ||__||  __||     Copyright (c) 2003, 2004 Sven Eric Panitz\n\
5  \     ||         ___||           http://www.panitz.name/\n\
6  ||   ||                         type ':' for help\n\
7  \\\\__//  Version: February 2004  ___________________________________________\n\
8  \
9
10 COMMAND_HELP =  \n\
11 \ Commands available from the prompt:\n\
12 \n\
13 \  <stmt>                    evaluate/run <stmt>\n\
14 \  :help,   :               displays this list of command\n\
15 \  :quit                     exit jugs\n\
16 \  :statement                switch to statement mode\n\
17 \  :expression               switch to expression mode\n\
18 \n\
19 \commands can be abbreviated to :h, :q etc.\n
20
21 UNKNOWN_COMMAND = unknown command:\
22
23
```

3.6 Beispielsession

```
sep@linux:~/fh/jugs/examples> java -classpath classes/:src/:$CLASSPATH name.panitz.crempel.tool.jugs.Jugs
     -- --  --  ----   ---      ----------------------------------------
     || ||  || ||  || ||__      Jugs: the interactive Java interpreter
     || ||__|| ||__||  __||     Copyright (c) 2003, 2004 Sven Eric Panitz
     ||         ___||           http://www.panitz.name/
||   ||                         type ':?' for help
\\__//  Version: February 2004  ----------------------------------------

> 2*21
42
> "Burgstrasse".toUpperCase().substring(5)
TRASSE
> :s
> :?

 Commands available from the prompt:

  <stmt>                    evaluate/run <stmt>
  :help,   :?               displays this list of command
  :quit                     exit jugs
  :statement                switch to statement mode
  :expression               switch to expression mode

commands can be abbreviated to :h, :q etc.

> map.put("fenster",new javax.swing.JFrame("Hallo Welt"));
> javax.swing.JFrame f=(javax.swing.JFrame)map.get("fenster");f.pack();f.setVisible(true);
> javax.swing.JFrame f=(javax.swing.JFrame)map.get("fenster");f.getContentPane().add(new javax.swing.JButton("K
> :q
sep@linux:~/fh/jugs/examples>
```

Wie man sieht, steht nichts dem entgegen, GUI-Elemente in Jugs zu öffnen oder ähnliches zu machen.

4 Graphische Benutzerschnittstelle

JugsGui.java

```
1  package name.panitz.crempel.tool.jugs;
2  import java.util.List;
3  import java.util.ArrayList;
4  import javax.swing.*;
5  import java.awt.event.*;
6  import java.awt.*;
7
8  import java.io.*;
9
10 import name.panitz.crempel.tool.CrempelTool;
11
12 public class JugsGui extends JFrame
13                       implements CrempelTool
14 {
15
16   public String getDescription(){return "Jugs";}
```

```
17  public void startUp(){new JugsGui(new Jugs()).setVisible(true);};
18
19  private final Jugs jugs;
20
21  private ImportsDialog importsDialog;
22  private FontSelectDialog fontSelectDialog;
23
24  JTextArea inputArea = new JTextArea(15,80);
25  JTextArea outputArea = new JTextArea(15,80);
26
27  private JLabel imagePanel
28    = new JLabel(new ImageIcon("images/jugs.jpg"));
29
30  private JPanel textPanel = new JPanel();
31
32  private JButton executeButton = new JButton("execute");
33  private JButton clearButton = new JButton("clear");
34
35  private JRadioButton expressionModeButton
36    = new JRadioButton("expression mode");
37  private JRadioButton statementModeButton
38    = new JRadioButton("statemente mode");
39  private ButtonGroup group = new ButtonGroup();
40
41  private JPanel buttonPanel = new JPanel();
42  private JPanel radioPanel = new JPanel();
43
44  private JPanel controlPanel = new JPanel();
45
46  private JMenuBar menuBar=new JMenuBar();
47  private JMenu fileMenu= new JMenu("File");
48  private JMenu optionsMenu= new JMenu("Options");
49  private JMenu helpMenu= new JMenu("Help");
50
51
52  private JMenuItem quitMenu
53   = new JMenuItem("quit",KeyEvent.VK_Q);
54  private JMenuItem executeMenu
55   = new JMenuItem("execute",KeyEvent.VK_X);
56  private JMenuItem clearMenu
57   = new JMenuItem("clear",KeyEvent.VK_C);
58
59  private JMenuItem importsMenu
60   = new JMenuItem("imports...",KeyEvent.VK_H);
61  private JMenuItem fontMenu
62   = new JMenuItem("font...",KeyEvent.VK_F);
63
64
65  private JMenuItem aboutMenu= new JMenuItem("about");
66
```

```
67
68    class TextAreaOutputStream extends java.io.OutputStream{
69      JTextArea area;
70      TextAreaOutputStream(JTextArea area){
71        this.area=area;
72      }
73
74      public void write(int b) {
75        area.append(new Character((char)b).toString() );
76      }
77    }
78
79    public JugsGui (){this(new Jugs());}
80
81    public JugsGui (final Jugs jugs){
82      super("Jugs: the interactive Java environment");
83      this.jugs=jugs;
84
85      Font [] fonts =  GraphicsEnvironment
86                        .getLocalGraphicsEnvironment()
87                        .getAllFonts();
88
89      for (int i = 0;i<fonts.length;i++){
90        Font f = fonts[i];
91        if(f.getFontName().equals("Courier")
92           && f.getStyle()==Font.PLAIN){
93          f=f.deriveFont((float)18);
94          inputArea.setFont(f);
95          outputArea.setFont(f);
96          break;
97        }
98      }
99
100     if (jugs.expressionMode) expressionModeButton.setSelected(true);
101     else statementModeButton.setSelected(true);
102
103     group.add(expressionModeButton);
104     group.add(statementModeButton);
105
106     expressionModeButton.addActionListener(
107       new ActionListener(){
108         public void actionPerformed(ActionEvent e){
109       jugs.expressionMode = true;
110     }
111       }
112     );
113
114
115     statementModeButton.addActionListener(
116       new ActionListener(){
```

```
117         public void actionPerformed(ActionEvent e){
118       jugs.expressionMode = false;
119         }
120       }
121     );
122 
123     executeButton.addActionListener(new ExecuteActionListener());
124 
125     clearButton.addActionListener(new ClearActionListener());
126 
127     Container panel = getContentPane();
128 
129     buttonPanel.setLayout(new GridLayout(2,1));
130     buttonPanel.add(executeButton);
131     buttonPanel.add(clearButton);
132 
133     radioPanel.setLayout(new GridLayout(2,1));
134     radioPanel.add(expressionModeButton);
135     radioPanel.add(statementModeButton);
136 
137     controlPanel.setLayout(new BorderLayout());
138     controlPanel.setBackground(java.awt.Color.RED);
139     controlPanel.add(buttonPanel,BorderLayout.NORTH);
140     controlPanel.add(imagePanel,BorderLayout.CENTER);
141     controlPanel.add(radioPanel,BorderLayout.SOUTH);
142 
143     textPanel.setLayout(new BorderLayout());
144 
145     Component inputPane = new JScrollPane(inputArea);
146     textPanel.add(inputPane,BorderLayout.NORTH);
147     textPanel.add(new JPanel(),BorderLayout.CENTER);
148 
149     Component outputPane = new JScrollPane(outputArea);
150     textPanel.add(outputPane,BorderLayout.SOUTH);
151     outputArea.setEditable(false);
152 
153     panel.setLayout(new BorderLayout());
154     panel.add(controlPanel,BorderLayout.WEST);
155     panel.add(textPanel,BorderLayout.CENTER);
156 
157 
158 
159     importsDialog = new ImportsDialog(this);
160     fontSelectDialog = new FontSelectDialog(this,inputArea.getFont().getSize()
161 
162     setJMenuBar(menuBar);
163 
164     optionsMenu.setMnemonic(KeyEvent.VK_O);
165     optionsMenu
166       .getAccessibleContext()
```

```
167       .setAccessibleDescription("to set options");
168
169     fileMenu.setMnemonic(KeyEvent.VK_F);
170     fileMenu.getAccessibleContext()
171             .setAccessibleDescription("quit etc...");
172
173     importsMenu
174       .getAccessibleContext()
175       .setAccessibleDescription("to add items to the import list");
176     optionsMenu.add(importsMenu);
177
178     importsMenu.addActionListener(
179       new ActionListener(){
180         public void actionPerformed(ActionEvent e){
181           importsDialog.setVisible(true);
182         }
183       }
184     );
185
186     fontMenu
187       .getAccessibleContext()
188       .setAccessibleDescription("to set the font size");
189     optionsMenu.add(fontMenu);
190
191     fontMenu.addActionListener(
192       new ActionListener(){
193         public void actionPerformed(ActionEvent e){
194           fontSelectDialog.setVisible(true);
195         }
196       }
197     );
198
199
200     quitMenu
201       .getAccessibleContext()
202       .setAccessibleDescription("quit jugs");
203     fileMenu.add(executeMenu);
204     fileMenu.add(clearMenu);
205     fileMenu.add(quitMenu);
206
207     clearMenu.addActionListener(new ClearActionListener());
208     executeMenu.addActionListener(new ExecuteActionListener());
209
210     quitMenu.addActionListener(
211       new ActionListener(){
212         public void actionPerformed(ActionEvent e){
213           System.exit(0);
214         }
215       }
216     );
```

```
217
218
219       helpMenu.add(aboutMenu);
220
221       aboutMenu.addActionListener(new AboutActionListener(this));
222
223       menuBar.add(fileMenu);
224       menuBar.add(optionsMenu);
225       menuBar.add(helpMenu);
226
227       pack();
228     }
229
230     class ClearActionListener implements ActionListener{
231        public void actionPerformed(ActionEvent e){
232          inputArea.setText("");
233        }
234     }
235
236
237     class ExecuteActionListener implements ActionListener{
238        public void actionPerformed(ActionEvent e){
239          outputArea.setText("");
240          System.setOut(new PrintStream(new TextAreaOutputStream(outputArea)));
241          System.setErr(new PrintStream(new TextAreaOutputStream(outputArea)));
242          jugs.execute(inputArea.getText(),importsList());
243          System.setOut(System.out);
244          System.setErr(System.err);
245        }
246
247        List<String> importsList(){
248          List<String> result=new ArrayList<String>();
249          java.util.Enumeration<String> imps = importsDialog.listModel.elements()
250          while (imps.hasMoreElements())
251            result.add(imps.nextElement());
252          return result;
253        }
254     }
255
256     class AboutActionListener implements ActionListener{
257       Component c;
258       AboutActionListener(Component c){this.c=c;}
259       public void actionPerformed(ActionEvent e){
260         JOptionPane.showMessageDialog(c, ABOUT_STRING);
261       }
262     }
263
264     static String ABOUT_STRING
265       =   "Jugs: Java UmGebungs System v."+Jugs.VERSION
266        +"\n2003 Sven Eric Panitz";
```

```
267
268   public static void main(String [] _){
269     JugsGui gui = new JugsGui(new Jugs());
270     gui.setVisible(true);
271   }
272 }
273
```

4.1 Options Dialoge

4.1.1 Fontggröße

FontSelectDialog.java

```
1  package name.panitz.crempel.tool.jugs;
2  import javax.swing.*;
3  import java.awt.Frame;
4  import java.awt.Container;
5  import java.awt.Component;
6  import java.awt.event.*;
7  import java.awt.*;
8  import java.util.Arrays;
9
10  class FontSelectDialog extends JDialog{
11   final JTextArea fontSizeArea = new  JTextArea(1,2);
12   final JButton okButton = new JButton("OK");
13   final JButton cancelButton = new JButton("Cancel");
14   final JPanel buttonPanel = new JPanel();
15   final JPanel sizePanel = new JPanel();
16   final JLabel sizeLabel = new JLabel("size");
17
18   final JRadioButton plain = new JRadioButton("plain");
19   final JRadioButton bold = new JRadioButton("bold");
20   final JRadioButton italic = new JRadioButton("italic");
21   final ButtonGroup  styleButtonGroup = new ButtonGroup();
22   final JPanel  styleButtonPanel  = new JPanel();
23   JList fontList;
24
25   int style  = Font.PLAIN;
26   int size;
27   final JugsGui jugsGui;
28
29   private void createButtonGroup(){
30     styleButtonGroup.add(plain);
31     styleButtonGroup.add(bold);
32     styleButtonGroup.add(italic);
33     plain.addActionListener(
34       new ActionListener(){
35         public void actionPerformed(ActionEvent e){
36           style=Font.PLAIN;
```

```
37          }
38        }
39      );
40      bold.addActionListener(
41        new ActionListener(){
42          public void actionPerformed(ActionEvent e){
43          style=Font.BOLD;
44          }
45        }
46      );
47
48      italic.addActionListener(
49        new ActionListener(){
50          public void actionPerformed(ActionEvent e){
51          style=Font.ITALIC;
52          }
53        }
54      );
55
56      styleButtonPanel.setLayout(new GridLayout(3,1));
57      styleButtonPanel.add(plain);
58      styleButtonPanel.add(bold);
59      styleButtonPanel.add(italic);
60    }
61
62    FontSelectDialog(JugsGui frame,int size) {
63      super(frame,"Font Size");
64      this.jugsGui = frame;
65      this.size=size;
66
67      createButtonGroup();
68
69      fontList = new JList(
70                       GraphicsEnvironment
71                       .getLocalGraphicsEnvironment()
72                       .getAvailableFontFamilyNames()
73                      );
74
75      Container contentPane = getContentPane();
76
77      JPanel p = new JPanel();
78      p.setLayout(new BorderLayout());
79      p.add(fontList,BorderLayout.WEST);
80      p.add(styleButtonPanel,BorderLayout.CENTER);
81
82      fontSizeArea.setText(""+size);
83
84      sizePanel.add(sizeLabel);
85      sizePanel.add(fontSizeArea);
86      p.add(sizePanel,BorderLayout.EAST);
```

```
87
88      okButton.addActionListener(new OkListener(this));
89      cancelButton.addActionListener(new CancelListener(this));
90
91      buttonPanel.add(cancelButton);
92      buttonPanel.add(okButton);
93      p.add(buttonPanel,BorderLayout.SOUTH);
94      contentPane.add(p);
95      pack();
96    }
97
98    class OkListener implements ActionListener{
99      Component c;
100     OkListener(Component c){this.c = c;}
101
102     public void actionPerformed(ActionEvent e) {
103       size = new Integer(fontSizeArea.getText()).intValue();
104       final java.awt.Font newFont
105           = new Font((String)fontList.getSelectedValue()
106                     ,style
107                     ,size);
108       jugsGui.inputArea.setFont(newFont);
109       jugsGui.outputArea.setFont(newFont);
110       c.setVisible(false);
111     }
112   };
113
114   class CancelListener implements ActionListener{
115     Component c;
116     CancelListener(Component c){this.c = c;}
117
118     public void actionPerformed(ActionEvent e) {
119       fontSizeArea.setText(""+size);
120       c.setVisible(false);
121     }
122   };
123
124 }
```

4.1.2 Import-Anweisungen

ImportsDialog.java

```
1  package name.panitz.crempel.tool.jugs;
2
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.util.*;
```

```
8
9  public class ImportsDialog
10   extends JDialog
11   implements ListSelectionListener{
12
13   DefaultListModel listModel = new DefaultListModel();;
14   private JList list=new JList(listModel);
15
16   final JButton okButton = new JButton("OK");
17   final JButton addButton = new JButton("Add");
18   final JButton deleteButton = new JButton("Delete");
19   private JTextField importName = new JTextField(30);
20
21   ImportsDialog(Frame frame) {
22     super(frame, "Imports", true);
23
24     list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
25     list.addListSelectionListener(this);
26
27     JScrollPane listScrollPane = new JScrollPane(list);
28     importName.addActionListener(new AddListener());
29
30     okButton.addActionListener(new OkListener(this));
31
32     addButton.addActionListener(new AddListener());
33
34     deleteButton.addActionListener(new ActionListener() {
35       public void actionPerformed(ActionEvent e) {
36         int index = list.getSelectedIndex();
37         listModel.remove(index);
38
39     int size = listModel.getSize();
40
41     //Nobody's left, disable firing
42     if (size == 0) {
43           deleteButton.setEnabled(false);
44
45         //Adjust the selection
46         } else {
47           //removed item in last position
48           if (index == listModel.getSize())
49             index--;
50           //otherwise select same index
51           list.setSelectedIndex(index);
52         }
53       }
54     });
55
56     JPanel buttonPane = new JPanel();
57     buttonPane.add(importName);
```

```
58      buttonPane.add(deleteButton);
59      buttonPane.add(addButton);
60      buttonPane.add(okButton);
61
62      Container contentPane = getContentPane();
63      contentPane.add(listScrollPane, BorderLayout.CENTER);
64      contentPane.add(buttonPane, BorderLayout.SOUTH);
65
66      getRootPane().setDefaultButton(okButton);
67      pack();
68    }
69
70    class AddListener implements ActionListener{
71      public void actionPerformed(ActionEvent e) {
72        //User didn't type in a name...
73        if (importName.getText().equals("")) {
74          Toolkit.getDefaultToolkit().beep();
75          return;
76        }
77
78        int index = list.getSelectedIndex();
79        int size = listModel.getSize();
80
81        //If no selection or if item in last position is selected,
82        //add the new hire to end of list, and select new hire
83        if (index == -1 || (index+1 == size)) {
84          listModel.addElement(importName.getText());
85          list.setSelectedIndex(size);
86
87        //Otherwise insert the new hire after the current selection,
88        //and select new hire
89        } else {
90          listModel.insertElementAt(importName.getText(), index+1);
91          list.setSelectedIndex(index+1);
92        }
93      }
94    };
95
96    class OkListener implements ActionListener{
97      Component c;
98      OkListener(Component c){this.c = c;}
99
100     public void actionPerformed(ActionEvent e) {
101       c.setVisible(false);
102     }
103   };
104
105   public void valueChanged(ListSelectionEvent e) {
106     if (!e.getValueIsAdjusting()) {
107
```

```
108         if (list.getSelectedIndex() != -1) {
109           addButton.setEnabled(true);
110           String name = list.getSelectedValue().toString();
111           importName.setText(name);
112         }
113       }
114     }
115
116   }
```