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

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

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 abszuspeichern.

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

```java
package name.panitz.crempel.tool.jugs;

import java.net.URL;
import java.net.URLClassLoader;

public class JugsClassLoader extends URLClassLoader {
  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

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

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

JugsClassLoader.java

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

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

```java
package name.panitz.crempel.tool.jugs;

import java.io.*;

import java.net.URLClassLoader;
import java.net.URL;
import java.lang.reflect.Method;

import java.util.List;
```

```
import java.util.ArrayList;
import java.util.ResourceBundle;

public class Jugs {
```

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

Jugs.java
```
  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
```
  final public String USER_DIR
   = 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
```
  static ResourceBundle labels = ResourceBundle
  .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
```
  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
```
  public static void main(String [] args){
    Jugs jugs = new Jugs();
    System.out.println(labels.getString("WELCOME"));
    while (true){
      try {
```



```
        System.out.print(">\u0020");
        String input
         =  new BufferedReader(
                  new InputStreamReader(System.in)).readLine();

        if (input.startsWith(":")) handleCommand(input,jugs);
        else jugs.execute(input);
      }catch (IOException e){
        System.out.println(e);
      }
    }
  }
```

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
```
  private static void handleCommand(String input,Jugs jugs){
    if (input.startsWith(":q")) System.exit(0);
    else if (input.startsWith(":s"))
      jugs.expressionMode=false;
    else if (input.startsWith(":e"))
      jugs.expressionMode=true;
    else if (input.startsWith(":h") || input.startsWith(":?"))
      System.out.println(labels.getString("COMMAND_HELP"));
    else
     System.out.println(labels.getString("UNKNOWN_COMMAND")+input);
  }
```

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

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

Jugs.java

```
  public void execute (String input,List<String> imports){
    try{
      final String newClass = mkClass(input,imports);
      final FileWriter writeClass
        = new FileWriter(USER_DIR+MAIN_JUGS_CLASS+".java");
      writeClass.write(newClass,0,newClass.length());
      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

```
      String [] javacArg
        = {"-source","1.5"
          ,USER_DIR+MAIN_JUGS_CLASS+".java"};

      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

```
      if (erg==0){
        String classPath = System.getProperty("java.class.path");

        URL [] urls = { new URL("file://localhost/"+USER_DIR+"/")};

        JugsClassLoader loader = new JugsClassLoader(urls);

        java.util.StringTokenizer st
          = new java.util.StringTokenizer
              (classPath
              ,System.getProperty("path.separator"));

        while (st.hasMoreTokens()) {
          try {
            loader.addURL(new URL("file://localhost"+st.nextToken()));
          }catch (java.net.MalformedURLException _){}
        }
```

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

```
        Object o = loader.loadClass("MainJugsTestClass").newInstance();
        Class cl = o.getClass();

```



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

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

```
    }catch (java.lang.reflect.InvocationTargetException e){
      System.out.println(e);
    }catch (NoSuchMethodException e){
      System.out.println(e);
    }catch (ClassNotFoundException e){
      System.out.println(e);
    }catch (InstantiationException e){
      System.out.println(e);
    }catch (IllegalAccessException e){
      System.out.println(e);
    }catch (IOException e){
      System.out.println(e);
    }catch (ClassFormatError e){
      System.out.println(e);
    }
  }
```

3.4 Klassengenerierung

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

Jugs.java

```
  public String mkClass(String content,List<String> imports){
    String importsString = "";
    for (String imp:imports)
      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

```
    if (expressionMode)
     return importsString
     +"public class "
     + MAIN_JUGS_CLASS
     +" implements name.panitz.crempel.tool.jugs"
     +".MainJugsTestClassPrototype {\n"
     +"  public void run() {\n"
```

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

Befehle können direkt in den Methodenrumpf eingefügt werden.

Jugs.java

```java
     else return importsString
          +"public class "
          + MAIN_JUGS_CLASS
          +" implements name.panitz.crempel.tool.jugs."
                                 +"MainJugsTestClassPrototype {\n"
          +"  public void run(){try{"+content+"}\n"
          +"    catch(Exception e){System.out.println(e);}}}";
  }
```

Jugs.java

```java
  static public final String VERSION = "0.3.1";
}
```

3.5 Dialogtexte

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

JugsInfo.properties

```

WELCOME =\     __ __  __  ____   ___      ______________________________________
\     || ||  || ||  || ||__       Jugs: the interactive Java interpreter\n\
\     || ||__|| ||__||  __||      Copyright (c) 2003, 2004 Sven Eric Panitz\n\
\     ||         ___||            http://www.panitz.name/\n\
||   ||                          type ':´ for help\n\
\\\\__//  Version: February 2004  ________________________________________\n\
\

COMMAND_HELP =  \n\
\ Commands available from the prompt:\n\
\n\
\  <stmt>                       evaluate/run <stmt>\n\
\  :help,   :                  displays this list of command\n\
\  :quit                        exit jugs\n\
\  :statement                   switch to statement mode\n\
\  :expression                  switch to expression mode\n\
\n\
\commands can be abbreviated to :h, :q etc.\n

UNKNOWN_COMMAND = unknown command:\


```


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(
> :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

```java
package name.panitz.crempel.tool.jugs;
import java.util.List;
import java.util.ArrayList;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

import java.io.*;

import name.panitz.crempel.tool.CrempelTool;

public class JugsGui extends JFrame
                     implements CrempelTool
{

  public String getDescription(){return "Jugs";}
```

```
public void startUp(){new JugsGui(new Jugs()).setVisible(true);};

private final Jugs jugs;

private ImportsDialog importsDialog;
private FontSelectDialog fontSelectDialog;

JTextArea inputArea = new JTextArea(15,80);
JTextArea outputArea = new JTextArea(15,80);

private JLabel imagePanel
  = new JLabel(new ImageIcon("images/jugs.jpg"));

private JPanel textPanel = new JPanel();

private JButton executeButton = new JButton("execute");
private JButton clearButton = new JButton("clear");

private JRadioButton expressionModeButton
  = new JRadioButton("expression mode");
private JRadioButton statementModeButton
  = new JRadioButton("statemente mode");
private ButtonGroup group = new ButtonGroup();

private JPanel buttonPanel = new JPanel();
private JPanel radioPanel = new JPanel();

private JPanel controlPanel = new JPanel();

private JMenuBar menuBar=new JMenuBar();
private JMenu fileMenu= new JMenu("File");
private JMenu optionsMenu= new JMenu("Options");
private JMenu helpMenu= new JMenu("Help");


private JMenuItem quitMenu
 = new JMenuItem("quit",KeyEvent.VK_Q);
private JMenuItem executeMenu
 = new JMenuItem("execute",KeyEvent.VK_X);
private JMenuItem clearMenu
 = new JMenuItem("clear",KeyEvent.VK_C);

private JMenuItem importsMenu
 = new JMenuItem("imports...",KeyEvent.VK_H);
private JMenuItem fontMenu
 = new JMenuItem("font...",KeyEvent.VK_F);


private JMenuItem aboutMenu= new JMenuItem("about");

```



```

class TextAreaOutputStream extends java.io.OutputStream{
  JTextArea area;
  TextAreaOutputStream(JTextArea area){
    this.area=area;
  }

  public void write(int b) {
    area.append(new Character((char)b).toString() );
  }
}

public JugsGui (){this(new Jugs());}

public JugsGui (final Jugs jugs){
  super("Jugs: the interactive Java environment");
  this.jugs=jugs;

  Font [] fonts =  GraphicsEnvironment
                    .getLocalGraphicsEnvironment()
                    .getAllFonts();

  for (int i = 0;i<fonts.length;i++){
    Font f = fonts[i];
    if(f.getFontName().equals("Courier")
       && f.getStyle()==Font.PLAIN){
      f=f.deriveFont((float)18);
      inputArea.setFont(f);
      outputArea.setFont(f);
      break;
    }
  }

  if (jugs.expressionMode) expressionModeButton.setSelected(true);
  else statementModeButton.setSelected(true);

  group.add(expressionModeButton);
  group.add(statementModeButton);

  expressionModeButton.addActionListener(
    new ActionListener(){
      public void actionPerformed(ActionEvent e){
    jugs.expressionMode = true;
  }
    }
  );


  statementModeButton.addActionListener(
    new ActionListener(){
```

```
        public void actionPerformed(ActionEvent e){
      jugs.expressionMode = false;
        }
      }
    );

    executeButton.addActionListener(new ExecuteActionListener());

    clearButton.addActionListener(new ClearActionListener());

    Container panel = getContentPane();

    buttonPanel.setLayout(new GridLayout(2,1));
    buttonPanel.add(executeButton);
    buttonPanel.add(clearButton);

    radioPanel.setLayout(new GridLayout(2,1));
    radioPanel.add(expressionModeButton);
    radioPanel.add(statementModeButton);

    controlPanel.setLayout(new BorderLayout());
    controlPanel.setBackground(java.awt.Color.RED);
    controlPanel.add(buttonPanel,BorderLayout.NORTH);
    controlPanel.add(imagePanel,BorderLayout.CENTER);
    controlPanel.add(radioPanel,BorderLayout.SOUTH);

    textPanel.setLayout(new BorderLayout());

    Component inputPane = new JScrollPane(inputArea);
    textPanel.add(inputPane,BorderLayout.NORTH);
    textPanel.add(new JPanel(),BorderLayout.CENTER);

    Component outputPane = new JScrollPane(outputArea);
    textPanel.add(outputPane,BorderLayout.SOUTH);
    outputArea.setEditable(false);

    panel.setLayout(new BorderLayout());
    panel.add(controlPanel,BorderLayout.WEST);
    panel.add(textPanel,BorderLayout.CENTER);



    importsDialog = new ImportsDialog(this);
    fontSelectDialog = new FontSelectDialog(this,inputArea.getFont().getSize());

    setJMenuBar(menuBar);

    optionsMenu.setMnemonic(KeyEvent.VK_O);
    optionsMenu
      .getAccessibleContext()
```



```
      .setAccessibleDescription("to set options");

    fileMenu.setMnemonic(KeyEvent.VK_F);
    fileMenu.getAccessibleContext()
            .setAccessibleDescription("quit etc...");

    importsMenu
      .getAccessibleContext()
      .setAccessibleDescription("to add items to the import list");
    optionsMenu.add(importsMenu);

    importsMenu.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          importsDialog.setVisible(true);
        }
      }
    );

    fontMenu
      .getAccessibleContext()
      .setAccessibleDescription("to set the font size");
    optionsMenu.add(fontMenu);

    fontMenu.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          fontSelectDialog.setVisible(true);
        }
      }
    );


    quitMenu
      .getAccessibleContext()
      .setAccessibleDescription("quit jugs");
    fileMenu.add(executeMenu);
    fileMenu.add(clearMenu);
    fileMenu.add(quitMenu);

    clearMenu.addActionListener(new ClearActionListener());
    executeMenu.addActionListener(new ExecuteActionListener());

    quitMenu.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          System.exit(0);
        }
      }
    );
```

```java


    helpMenu.add(aboutMenu);

    aboutMenu.addActionListener(new AboutActionListener(this));

    menuBar.add(fileMenu);
    menuBar.add(optionsMenu);
    menuBar.add(helpMenu);

    pack();
  }

  class ClearActionListener implements ActionListener{
     public void actionPerformed(ActionEvent e){
       inputArea.setText("");
     }
  }


  class ExecuteActionListener implements ActionListener{
     public void actionPerformed(ActionEvent e){
       outputArea.setText("");
       System.setOut(new PrintStream(new TextAreaOutputStream(outputArea)));
       System.setErr(new PrintStream(new TextAreaOutputStream(outputArea)));
       jugs.execute(inputArea.getText(),importsList());
       System.setOut(System.out);
       System.setErr(System.err);
     }

     List<String> importsList(){
       List<String> result=new ArrayList<String>();
       java.util.Enumeration<String> imps = importsDialog.listModel.elements();
       while (imps.hasMoreElements())
         result.add(imps.nextElement());
       return result;
     }
  }

  class AboutActionListener implements ActionListener{
    Component c;
    AboutActionListener(Component c){this.c=c;}
    public void actionPerformed(ActionEvent e){
      JOptionPane.showMessageDialog(c, ABOUT_STRING);
    }
  }

  static String ABOUT_STRING
    =   "Jugs: Java UmGebungs System v."+Jugs.VERSION
     +"\n2003 Sven Eric Panitz";
```



```java

  public static void main(String [] _){
    JugsGui gui = new JugsGui(new Jugs());
    gui.setVisible(true);
  }
}

```

4.1 Options Dialoge

4.1.1 Fontggröße

FontSelectDialog.java

```java
package name.panitz.crempel.tool.jugs;
import javax.swing.*;
import java.awt.Frame;
import java.awt.Container;
import java.awt.Component;
import java.awt.event.*;
import java.awt.*;
import java.util.Arrays;

 class FontSelectDialog extends JDialog{
  final JTextArea fontSizeArea = new  JTextArea(1,2);
  final JButton okButton = new JButton("OK");
  final JButton cancelButton = new JButton("Cancel");
  final JPanel buttonPanel = new JPanel();
  final JPanel sizePanel = new JPanel();
  final JLabel sizeLabel = new JLabel("size");

  final JRadioButton plain = new JRadioButton("plain");
  final JRadioButton bold = new JRadioButton("bold");
  final JRadioButton italic = new JRadioButton("italic");
  final ButtonGroup  styleButtonGroup = new ButtonGroup();
  final JPanel  styleButtonPanel  = new JPanel();
  JList fontList;

  int style  = Font.PLAIN;
  int size;
  final JugsGui jugsGui;

  private void createButtonGroup(){
    styleButtonGroup.add(plain);
    styleButtonGroup.add(bold);
    styleButtonGroup.add(italic);
    plain.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          style=Font.PLAIN;
```

```
        }
      }
    );
    bold.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
        style=Font.BOLD;
        }
      }
    );

    italic.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
        style=Font.ITALIC;
        }
      }
    );

    styleButtonPanel.setLayout(new GridLayout(3,1));
    styleButtonPanel.add(plain);
    styleButtonPanel.add(bold);
    styleButtonPanel.add(italic);
  }

  FontSelectDialog(JugsGui frame,int size) {
    super(frame,"Font Size");
    this.jugsGui = frame;
    this.size=size;

    createButtonGroup();

    fontList = new JList(
                      GraphicsEnvironment
                      .getLocalGraphicsEnvironment()
                      .getAvailableFontFamilyNames()
                     );

    Container contentPane = getContentPane();

    JPanel p = new JPanel();
    p.setLayout(new BorderLayout());
    p.add(fontList,BorderLayout.WEST);
    p.add(styleButtonPanel,BorderLayout.CENTER);

    fontSizeArea.setText(""+size);

    sizePanel.add(sizeLabel);
    sizePanel.add(fontSizeArea);
    p.add(sizePanel,BorderLayout.EAST);
```



```

    okButton.addActionListener(new OkListener(this));
    cancelButton.addActionListener(new CancelListener(this));

    buttonPanel.add(cancelButton);
    buttonPanel.add(okButton);
    p.add(buttonPanel,BorderLayout.SOUTH);
    contentPane.add(p);
    pack();
  }

  class OkListener implements ActionListener{
    Component c;
    OkListener(Component c){this.c = c;}

    public void actionPerformed(ActionEvent e) {
      size = new Integer(fontSizeArea.getText()).intValue();
      final java.awt.Font newFont
          = new Font((String)fontList.getSelectedValue()
                    ,style
                    ,size);
      jugsGui.inputArea.setFont(newFont);
      jugsGui.outputArea.setFont(newFont);
      c.setVisible(false);
    }
  };

  class CancelListener implements ActionListener{
    Component c;
    CancelListener(Component c){this.c = c;}

    public void actionPerformed(ActionEvent e) {
      fontSizeArea.setText(""+size);
      c.setVisible(false);
    }
  };

}
```

4.1.2 Import-Anweisungen

ImportsDialog.java

```
package name.panitz.crempel.tool.jugs;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
```

```

public class ImportsDialog
  extends JDialog
  implements ListSelectionListener{

  DefaultListModel listModel = new DefaultListModel();;
  private JList list=new JList(listModel);

  final JButton okButton = new JButton("OK");
  final JButton addButton = new JButton("Add");
  final JButton deleteButton = new JButton("Delete");
  private JTextField importName = new JTextField(30);

  ImportsDialog(Frame frame) {
    super(frame, "Imports", true);

    list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    list.addListSelectionListener(this);

    JScrollPane listScrollPane = new JScrollPane(list);
    importName.addActionListener(new AddListener());

    okButton.addActionListener(new OkListener(this));

    addButton.addActionListener(new AddListener());

    deleteButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        int index = list.getSelectedIndex();
        listModel.remove(index);

    int size = listModel.getSize();

    //Nobody's left, disable firing
    if (size == 0) {
          deleteButton.setEnabled(false);

        //Adjust the selection
        } else {
          //removed item in last position
          if (index == listModel.getSize())
            index--;
          //otherwise select same index
          list.setSelectedIndex(index);
        }
      }
    });

    JPanel buttonPane = new JPanel();
    buttonPane.add(importName);
```



```
    buttonPane.add(deleteButton);
    buttonPane.add(addButton);
    buttonPane.add(okButton);

    Container contentPane = getContentPane();
    contentPane.add(listScrollPane, BorderLayout.CENTER);
    contentPane.add(buttonPane, BorderLayout.SOUTH);

    getRootPane().setDefaultButton(okButton);
    pack();
  }

  class AddListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
      //User didn't type in a name...
      if (importName.getText().equals("")) {
        Toolkit.getDefaultToolkit().beep();
        return;
      }

      int index = list.getSelectedIndex();
      int size = listModel.getSize();

      //If no selection or if item in last position is selected,
      //add the new hire to end of list, and select new hire
      if (index == -1 || (index+1 == size)) {
        listModel.addElement(importName.getText());
        list.setSelectedIndex(size);

      //Otherwise insert the new hire after the current selection,
      //and select new hire
      } else {
        listModel.insertElementAt(importName.getText(), index+1);
        list.setSelectedIndex(index+1);
      }
    }
  };

  class OkListener implements ActionListener{
    Component c;
    OkListener(Component c){this.c = c;}

    public void actionPerformed(ActionEvent e) {
      c.setVisible(false);
    }
  };

  public void valueChanged(ListSelectionEvent e) {
    if (!e.getValueIsAdjusting()) {

```

```
      if (list.getSelectedIndex() != -1) {
        addButton.setEnabled(true);
        String name = list.getSelectedValue().toString();
        importName.setText(name);
      }
    }
  }

}
```