



Programmdokumentation

Teilprojekt Bühnenbildautomation

Universitäts-Rechenzentrum Freiburg

Autorin: Hannah Ullrich
Ausbildung: Applikationsentwickler, kaufmännische Systeme
Ausbildungsbetrieb: Universitäts-Rechenzentrum Freiburg
Hermann-Herderstr. 10
79104 Freiburg

Inhaltsverzeichnis

1. EINLEITUNG	3
2. FUNKTIONALITÄT	3
3. SYSTEMVORAUSSETZUNGEN	4
3.1. Software	4
3.2. Hardware	4
4. KONVENTIONEN	5
5. KLASSEN	6
5.1. RegieGui	6
5.1.1. Konstruktor	7
5.1.2. private void initComponents()	7
5.1.3. public void valueChanged(ListSelectionEvent e)	7
5.1.4. public void actionPerformed(ActionEvent actionevent)	7
5.2. MyPush	8
5.2.1. Konstruktoren	8
5.2.2. public void run()	8
5.2.3. public static BufferedImage toBufferedImage(Image Picture)	9
5.2.4. public static boolean hasAlpha(Image Picture)	9
5.2.5. private void showPicture(Graphics g)	9
5.2.6. Getter/Setter	10
6. ABBILDUNGSVERZEICHNIS	11
7. INDEX	11

1. Einleitung

Das Programm ist in der Programmiersprache Java geschrieben und verwendet den Sprachenstandard der Version 1.4.2.

Die Benutzung einer älteren Version hätte zur Folge gehabt, dass *'deprecated'* (verworfen) Funktionen verwendet worden wären. Das hätte dazu geführt, dass der Sourcecode bereits in nächster Zukunft angepasst werden müsste um lauffähig zu bleiben. Der Sourcecode ist mit der neuesten Java Version 1.4.2, auch als Java 2 bezeichnet, zu 100 Prozent kompatibel.

2. Funktionalität

Über die grafische Benutzeroberfläche kann der Benutzer verschiedene Bildfilter und Bildformate in Listfeldern auswählen.

Die Kameras, deren Bild angezeigt werden soll, werden über Buttons ausgewählt. Zur Zeit sind zwei Kameras implementiert, dies kann aber noch beliebig erweitert werden.

Es wird eine Verbindung über das Netzwerk zu dem Webserver der *'Vistabox'* hergestellt. Der Stream wird in einem eigenen Fenster angezeigt.

Wird das Fenster mit dem Stream geschlossen, wird der laufende *'Thread'* gestoppt.

Über die *'GUI'* können neue Filter, Formate oder Kameras ausgewählt werden. Mit dem *'Exit-Button'* kann die Applikation ganz beendet werden.

Besteht der Wunsch während des laufenden Streams einen Filter zu wechseln, kann dies jederzeit durch die Auswahl eines neuen im Listfeld geschehen.

Ein Bild kann über den *'pause-Button'* eingefroren werden, d.h. der *'Thread'* wird angehalten. Durch klicken des *'continue-Button'* wird wieder ein Stream angezeigt.

3. Systemvoraussetzungen

3.1. Software

Java besitzt gegenüber anderen Programmiersprachen den Vorteil, dass ein in Java geschriebenes Programm nicht neu kompiliert werden muss, um auf einem anderen Betriebssystem zu laufen. Diese Plattformunabhängigkeit wird erreicht, indem der Sourcecode nicht sofort in Maschinensprache übersetzt wird, die direkt den Prozessor steuert, sondern zunächst eine teilweise Übersetzung in Java-Bytecode stattfindet. Dateien mit Java-Bytecode enden mit .class. Um diese Dateien zu starten, ist eine Java 'Virtual Machine (JVM)'¹ notwendig, die den Bytecode zu Laufzeit interpretiert und in Maschinensprache übersetzt. Die 'Java Virtual Machine' ist bereits für alle gängigen Betriebssysteme verfügbar. Dies gewährleistet, dass ein Java Programm, das einmal programmiert und kompiliert wurde, dort ohne Adaptierungen laufen kann.

3.2. Hardware

- Computer
 - ✓ Prozessorleistung 2500 MHz oder höher
 - ✓ Arbeitsspeicher mind. 512 MB RAM
- Vistabox
- Web-Cam
- 10 Mbit Twisted-Pair-Anschluss
- Cross-Connect-Kabel für den direkten Anschluss an den PC

¹ Freier download unter: <http://java.sun.com>

4. Konventionen

Durch das gesamte Programm wurde versucht eine konsistente Bezeichnungsart zu verwenden.

Besteht ein Bezeichner (Identifier) aus mehr als einem Wort, so beginnt jedes »innere« Wort mit einem Großbuchstaben.

Klassennamen fangen immer mit einen Großbuchstaben an und enthalten Substantive, wie MyPush, RegieGui Klasse.

Klassenmethoden bzw. Variable beginnen immer mit einem Kleinbuchstaben, wie z.B. setFilter(), getFormat(), exitForm() etc.

Für die Steuerelemente innerhalb der GUI wurden folgende Präfixe verwendet:

- panel – Container für die Komponenten (z.B. panelHeader1)
- btn – für die Buttons(z.B. btnExit)
- lbl – für ein Bezeichnungsfeld (z.B. lblCam1)
- list – für die Listfelder (z.B. listFilter)
- scroll – für die Bildlaufleiste (z.B. scrollFilter)

Alle Klassen, Methoden und Variablen haben, so weit wie möglich, einen aussagekräftigen Namen. Die einzige Ausnahme sind Zählvariablen, wie z.B. in Schleifenkonstrukten. Hier werden meistens Variable wie i, j, k, l, usw. verwendet.

Oder die Variable gilt nur in einem sehr kleinen Bereich. Hier sollte es sofort ersichtlich sein, wofür die Variable steht.

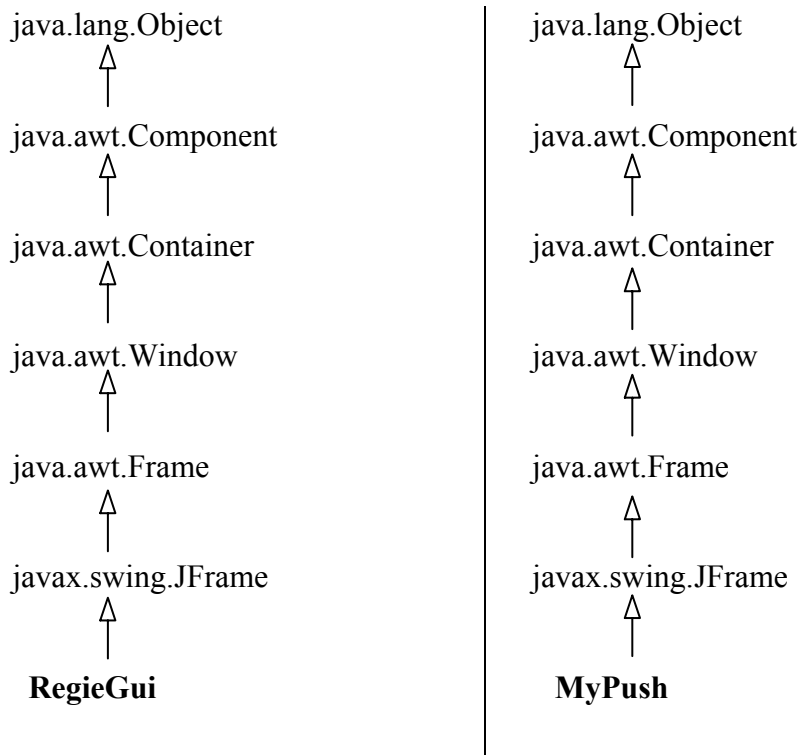
Bei den geschweiften Klammern wird folgende Konvention verwendet:

```
if ( bimage != null ) {  
    showPicture(g);  
}
```

Die öffnende Klammer steht immer in der gleichen Zeile wie der Bedingungsausdruck, und die schließende Klammer befindet sich auf der gleichen Höhe wie die Bedingung.

5. Klassen

Die folgende Grafik zeigt die Hierarchie der verwendeten Klassen an.



5.1. RegieGui

```
public class RegieGui
extends javax.swing.JFrame
implements java.awt.event.ActionListener, javax.swing.event.ListSelectionListener
```

Dies ist die Klasse, in der die Grafische Benutzeroberfläche aufgebaut wird, sie enthält die *'main'* – Methode zum Ausführen.

In ihr werden die Ereignisse (Events) behandelt und Objekte der Klasse **MyPush** erzeugt. Danach wird **MyPush** als Thread gestartet.

5.1.1. Konstruktor

Aufruf der Methode *'initComponents'*

5.1.2. `private void initComponents()`

Innerhalb dieser Methode werden alle Komponenten für den Aufbau der grafischen Benutzeroberfläche initialisiert.

Das Layout wird angelegt, die Listfelder werden gefüllt, Farben und Größen festgelegt.

5.1.3. `public void valueChanged(ListSelectionEvent e)`

Musste implementiert werden, da die Schnittstelle *'ListSelectionListener'* implementiert wurde.

Sobald eine Veränderung der Auswahl innerhalb des Listfeldes stattfindet, wird sie aufgerufen. Überprüft mit Hilfe *'getFilter()'* von *MyPush* welcher Filter ausgewählt war, setzt diesen wieder auf *'false'* und setzt über *'setFilter()'* den neuen Filter.

5.1.4. `public void actionPerformed(ActionEvent actionevent)`

Ist in erster Linie relevant, wenn die *GUI* das erste Mal gestartet wird.

Die Zustände der Schaltflächen werden überprüft und über die verschiedenen *set*-Methoden von *MyPush* auf *'true'* oder *'false'* gesetzt.

Der entsprechende Konstruktor der Folgeklasse wird aufgerufen.

Die Klasse wird als *'Thread'* gestartet.

5.2. MyPush

```
public class MyPush
```

```
extends javax.swing.JFrame implements java.lang.Runnable
```

5.2.1. Konstruktoren

- **public MyPush()**

hier wird die Bildschirmgröße über

'*GraphicsEnvironment.getDefaultScreenDevice()*' ermittelt, damit das Bild in 'fullsize-modus' angezeigt werden kann

- **public MyPush(String str)**

diesem Konstruktor wird ein Parameter übergeben, über den die Größe des Frames festgelegt wird, in den das Bild gezeichnet wird (800x600 oder 512x512 Format).

```
Dimension screensize = Toolkit.getDefaultToolkit().getScreenSize();  
setBounds(0, 0, 800, 600);
```

oder

```
setBounds(0, 0, 512, 512);
```

5.2.2. public void run()

Unter Verwendung der beiden Klassen '*URL*' und '*URLConnection*' wird eine Verbindung zum Web-Server der '*Vistabox*' hergestellt.

Die notwendige Information, welche Web-Cam angesprochen werden soll, wird über eine Variable, deren Zustand von der RegieGui-Klasse übergeben wird, übermittelt.

Von der bestehenden Verbindung wird gelesen und die Größe jedes einzelnen JPG's ermittelt, dies in ein Array geschrieben und daraus ein Bild erzeugt.

```
→ Picture = Toolkit.getDefaultToolkit().createImage(byteArray);
```

Es wird der Frame in der gewünschten Größe (fullsize, 800x600 oder 12x512) in das das Bild gezeichnet wird angelegt, ein gepuffertes Image erzeugt und über eine Methode das Bild auch gezeichnet.

Das Berechnen der Bilder wird in einer *'do-while'* Schleife durchgeführt, die durch das Umsetzen einer Variablen von false auf true beendet werden kann. Ist dies der Fall, wird das Array auf 'null' gesetzt und die Verbindung zum Web-Server wieder geschlossen.

5.2.3. public static BufferedImage toBufferedImage(Image Picture)

Aus dem Image, das über die Verbindung kommt, wird ein *'BufferedImage'* unter Berücksichtigung der Kompatibilität zum Bildschirm erstellt.

Über den Aufruf der Methode *'hasAlpha()'* wird überprüft, ob das Bild transparente Pixel enthält.

Liefert ein *'BufferedImage'* zurück.

5.2.4. public static boolean hasAlpha(Image Picture)

Der *'PixelGrabber'* wird verwendet, um das Farbmodell (*'ColorModel'*) des Bildes zu ermitteln.

Liefert 'true', wenn das Bild transparente Pixel enthält.

5.2.5. private void showPicture(Graphics g)

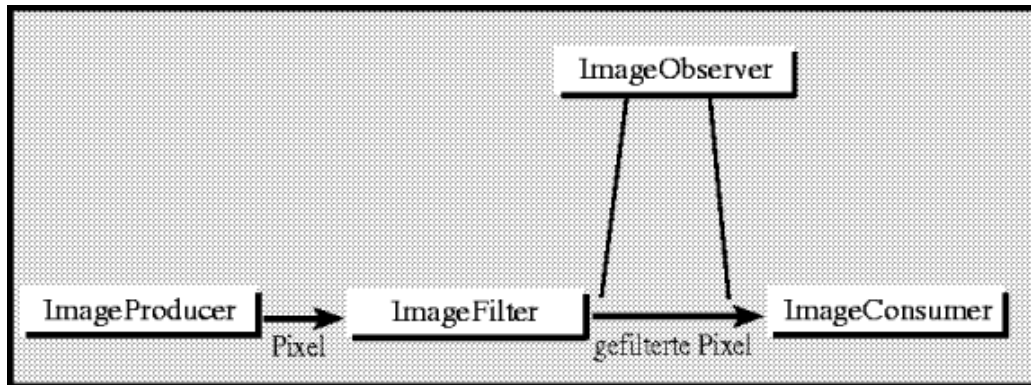
wird innerhalb der run()-Methode aufgerufen, wenn das Bild geladen ist und gezeichnet werden soll.

Es wird ein *'double buffered image'* angelegt, evtl. Filter angewendet und dann das Bild gezeichnet.

Bei den meisten Bildfiltern wird die Klasse *'Kernel'* verwendet, mit der eine *'Matrix'* von Pixel definiert wird, über die das Originalbild zum gefilterten Bild verrechnet wird. Dem Konstruktor von *'Kernel'* werden als Parameter die Ausmaße der Matrix und die Matrix selbst übergeben.

Ein Filter schaltet sich zwischen 'ImageProducer' und 'ImageConsumer'. Er empfängt die Bilddaten, die der 'ImageProducer' liefert, verändert sie nach bestimmten Regeln und gibt sie anschließend an den 'ImageConsumer' weiter.

Abbildung 1 Arbeitsweise der Image Filter²



5.2.6. Getter/Setter

Diese Methoden werden durch die *RegieGui* Klasse aufgerufen, um zu überprüfen, welche Komponenten innerhalb der *GUI* ausgewählt wurden.

Folgende Methoden stehen zur Verfügung:

- `public void setFilter(String type, boolean select);`
- `public String getFilter();`
- `public setFormat(String type, boolean select);`
- `public void setCam(String type, boolean select);`

² http://www.dpunkt.de/java/Programmieren_mit_Java/Grafikprogrammierung/17.html

6. Abbildungsverzeichnis

Abbildung 1 Arbeitsweise der Image Filter 10

7. Index

B		K	
<i>BufferedImage</i>	9	<i>Kernel</i>	9
C		L	
<i>ColorModel</i>	9	<i>ListSelectionListener</i>	7
<i>continue-Button</i>	3	M	
D		<i>main</i>	6
<i>deprecated</i>	3	<i>Matrix</i>	9
<i>double buffered image</i>	9	P	
<i>do-while</i>	9	<i>pause-Button</i>	3
E		<i>PixelGrabber</i>	9
<i>Exit-Button</i>	3	S	
G		<i>setFilter()</i>	7
<i>getFilter()</i>	7	T	
<i>GraphicsEnvironment</i>		<i>Thread</i>	3
<i>getDefaultScreenDevice()</i>	8	U	
<i>GUI</i>	3	<i>URL</i>	8
H		<i>URLConnection</i>	8
<i>hasAlpha()</i>	9	V	
I		<i>Virtual Machine (JVM)</i>	4
<i>ImageConsumer</i>	10	<i>Vistabox</i>	3
<i>ImageProducer</i>	10		