# Jeliot

Peter Blanchebarbe

"Animation of User Algorithms on the Web"

J. Haajanen, M. Pesonius, E. Sutinen, J. Tarhio,
T. Teräsvirta, P. Vanninen
University of Helsinki

# Überblick

- Einleitung
- Aus der Sicht des Benutzers
- Implementierung
- EJava
- Fazit
- Vorführung

# EINLEITUNG

- Programmiersprachen- & Algorithmenanimation

  **PA:** einfacher in Darstellung
  arbeitet nah an dem zugrunde liegenden Code

  **AA:** arbeitet auf höherer Ebene
  ist in ihrer Darstellung aufwendiger

  $\Rightarrow$ Unterschied in der Abstraktion

- Entwickelt an der University of Helsinki nach Ideen von Eliot

- Jeliot als interaktive Lernsoftware

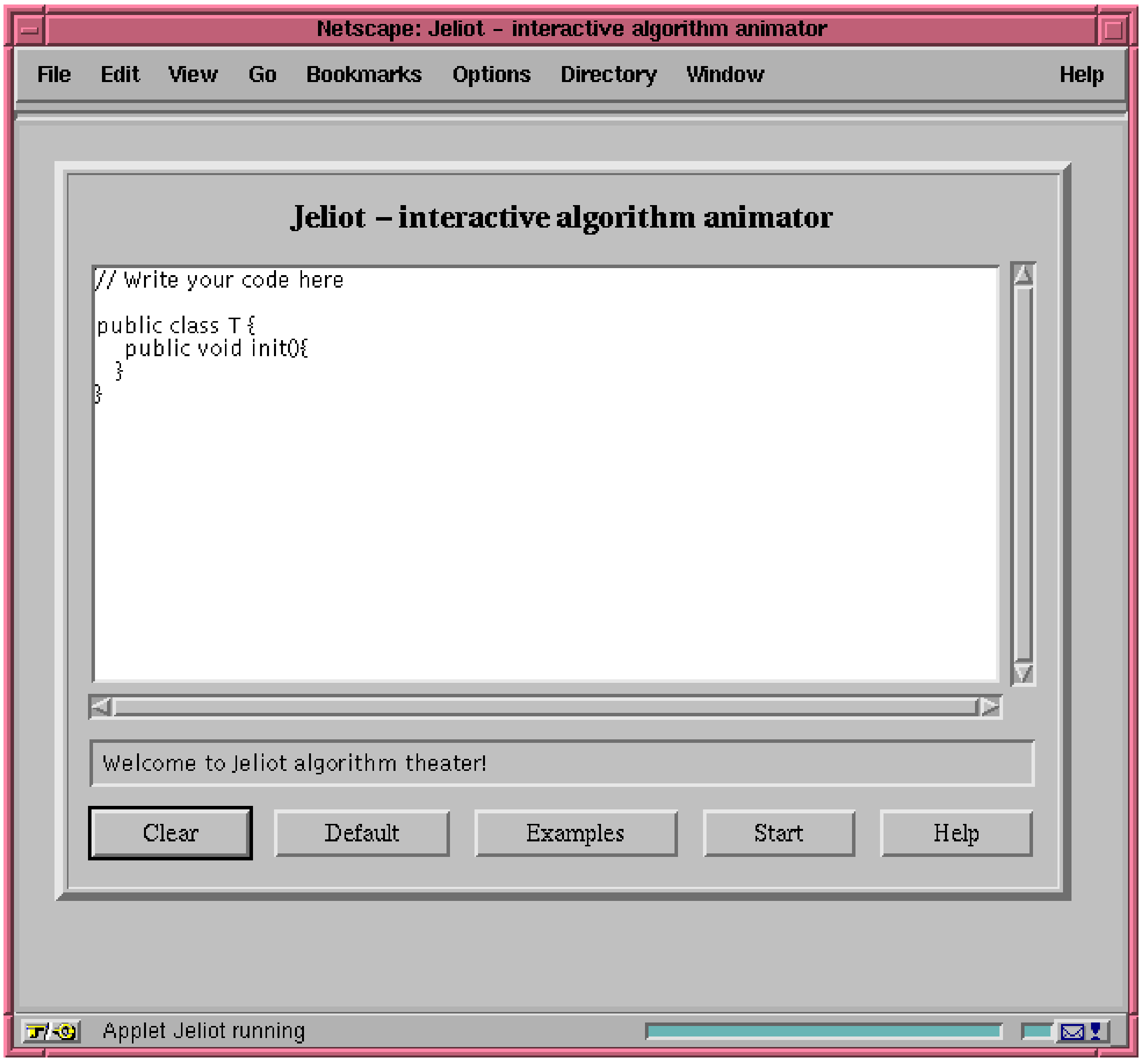
- Warum sollte man Jeliot benutzen:
  - warum tut das Programm das, was es soll ?
  - Java lernen: was ändert sich bei der Ausführung, wenn Veränderungen am Code vorgenommen werden
  - Debugger
  - Lehrer: was tut der Code des Schülers eigentlich
  - zum Spielen
- in aller Kürze: Benutzer entwickelt Java Programm, schickt dieses über das Internet an Jeliot, das wiederum das Animationsapplet erstellt und zurückschickt
- befindet sich noch in der Entwicklungsphase
- Client/Server-Struktur

- Animationen basieren auf der Theatermetapher:

| | | |
|---|---|---|
| Algorithmus | – | Skript |
| Animationsfenster | – | Stage |
| Grafische Repräsentation in Form, Farbe, Größe, Ort | – | Akteur |
| Datenobjekt | – | Rolle |
| Benutzer | – | Direktor |

# Aus der Sicht des Benutzers

- Das Hauptfenster:



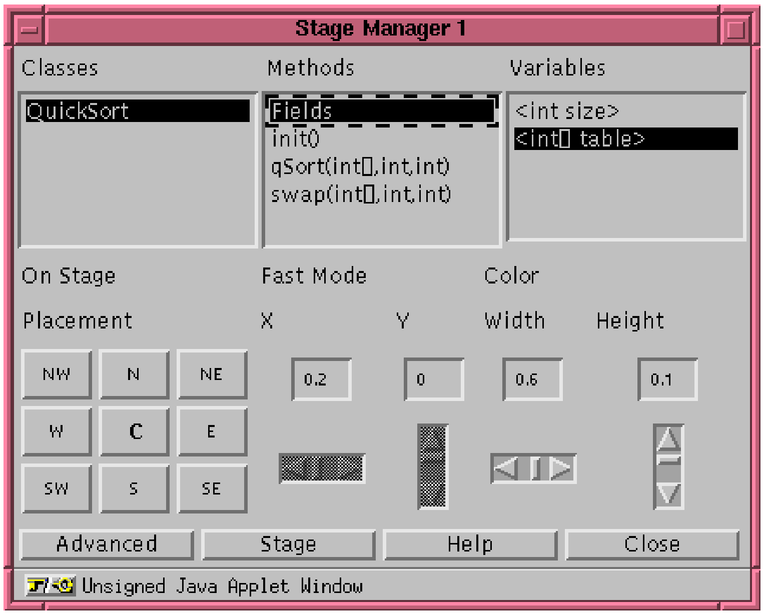

  - Applet mit Eingabemöglichkeit
  - Eingabe: Algorithmus in Java
  - Start Button: Algorithmus $\longrightarrow$ Server

- Precompiler
  - Extraktion aller möglichen animierbaren Variablen
  - Liste dieser Variablen $\longrightarrow$ Hauptfenster
- Java Compiler

- Direktor-Fenster

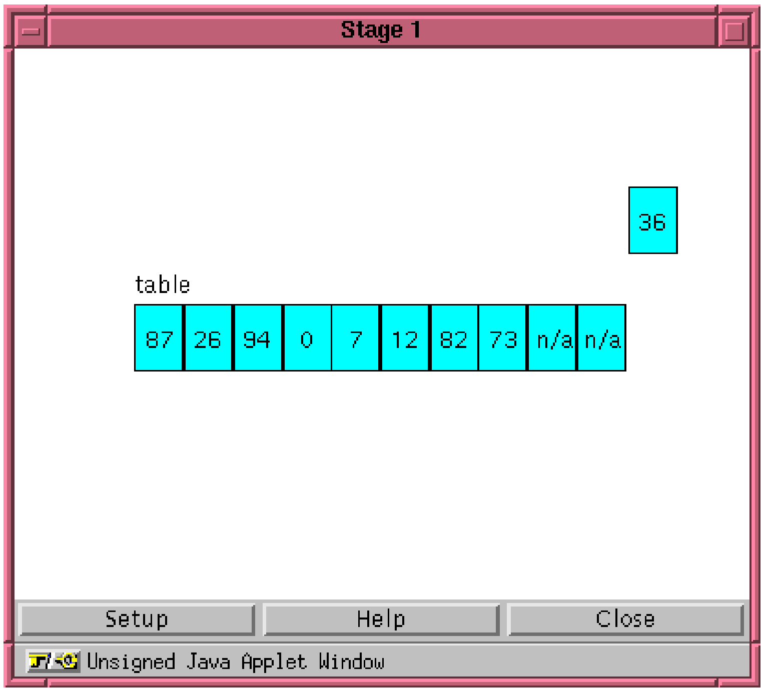

  - Steuerung des Animationsablaufs (Direktor eines Theaterstücks): Play/Stop, New Stage, Step, Run to, Trace on/off, Geschwindigkeit
  - Anzeige, welcher Teil des Algorithmus abgearbeitet wird

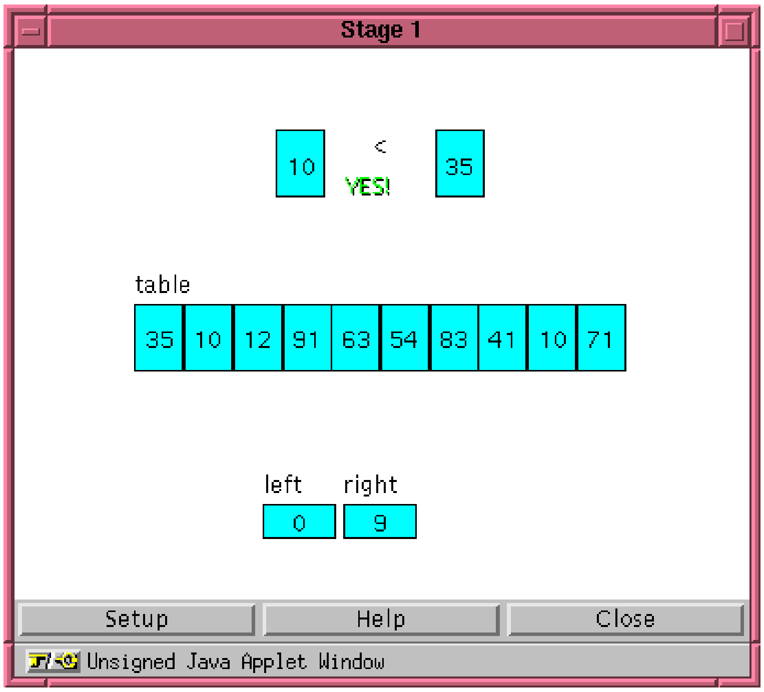
- Stage-Manager:




  - Jede Stage hat einen Stage-Manager
  - Auswahl der Variablen aus den Klassen und Methoden des Algorithmus
  - Aufstellung und Ausstattung der Variablen (Form, Farbe, Ort)
    **Form:** Oval, Kreis, Quadrat
    **Farbe:** Auswahl aus 13 Farben
    **Ort:** Auswahl der Himmelsrichtungen

- Animation:


Stage 1
36
table
87
26
94
0
7
12
82
73
n/a
n/a
Setup
Help
Close
Unsigned Java Applet Window


Stage 1
10
<
YES!
35
table
35
10
12
91
63
54
83
41
10
71
left
right
0
9
Setup
Help
Close
Unsigned Java Applet Window

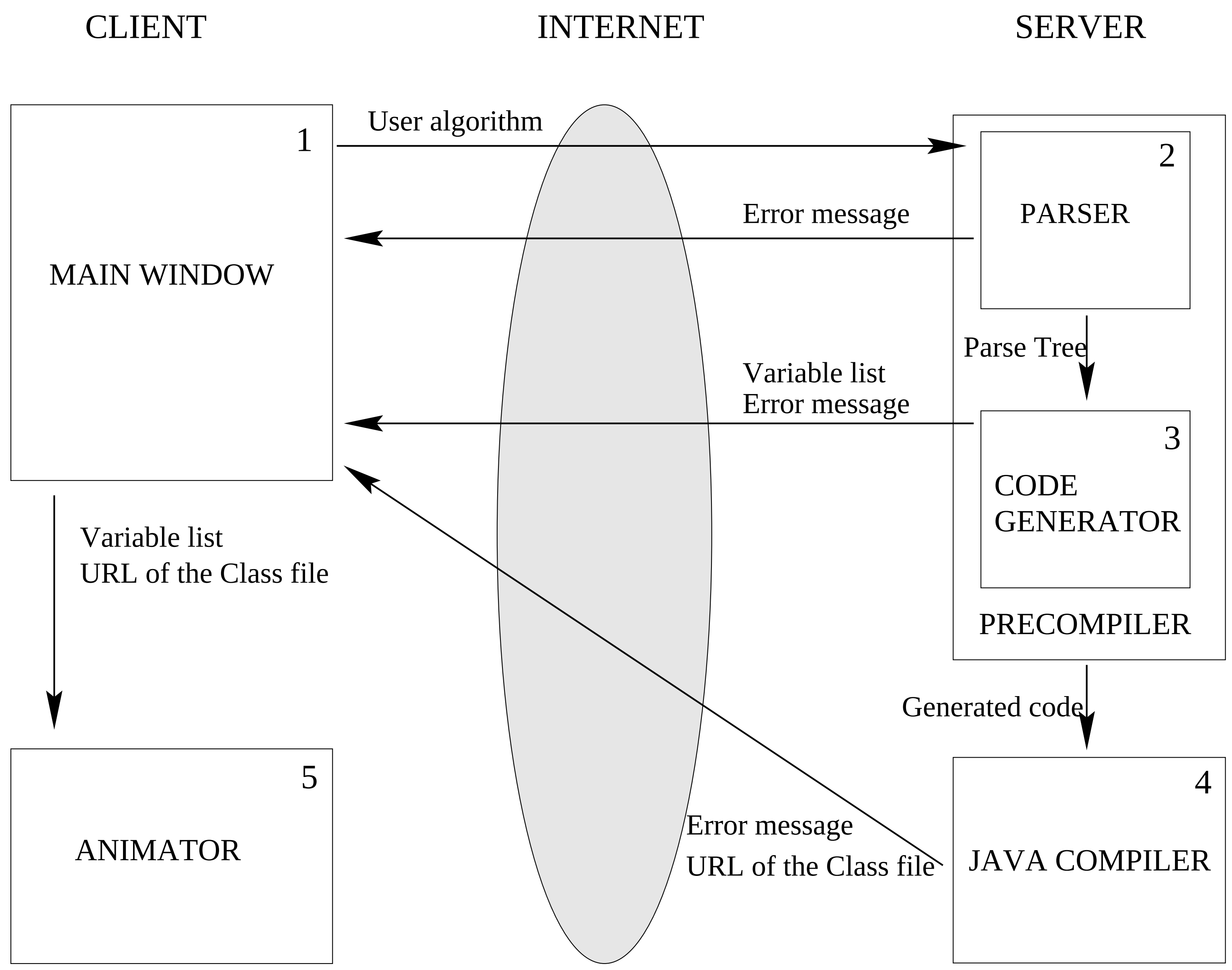
– Darstellung der Variablen:
  in der gewählten Form steht der aktuelle Wert der Variablen, der Name steht außerhalb der Form

– Darstellung der Operationen:
  **Zuweisung:** Wert wird in das entsprechende Objekt bewegt
  **Vergleich:** Objekte werden in einen Vergleichsraum bewegt, in dem dieser Vergleich farblich codiert dargestellt wird

– Das Direktor-Fenster hebt die entsprechende Zeile im Algorithmus hervor

– Darstellung der Variablen kann während der Animation über Setup geändert werden

# STRUKTUR


CLIENT
INTERNET
SERVER
1
MAIN WINDOW
User algorithm
2
PARSER
Error message
Parse Tree
Variable list
Error message
3
CODE
GENERATOR
PRECOMPILER
Variable list
URL of the Class file
Generated code
5
ANIMATOR
4
JAVA COMPILER
Error message
URL of the Class file

Implementierung

- Hauptfenster-Applet

- Server
  - Server empfängt alle Anfragen vom Hauptfenster
  - für jeden neuen Client wird ein Socket allokiert und ein Kommunikationsthread gestartet

- Kommunkationsthread
  - der KT legt für jeden Client ein Verzeichnis an, in das die erforderlichen Dateien geschrieben werden
  - er lauscht ständig dem Socket, der für ihn vom Server allokiert wurde

– Nachrichten:
**Anfragen** (Client an Server):

1. zu animierender Code
2. Stop requests: nicht mehr benötigte Resourcen werden freigegeben und der thread stoppt

**Antworten** (Server an Client):

1. Liste der Objekte, die animiert werden können
2. Referenz auf die Klasse, die geladen werden muß um die Animation zu starten
3. Fehlermeldungen von Parser, Code Generator, Java Compiler

- Verarbeitung des Quellcodes: Thread transformiert den erhaltenen Code in ein Java Class File

  1. Parser
     - Syntaxcheck
     - Erzeugung des Parse Tree

2. Code Generator

   - Modifikation des Codes

     – Erzeugung von selbstanimierenden Objekten (Ersetzung der Daten Objekte durch instanziierte Klassen)

     – 2 Arten von Klassen:
       **Rollenklasse:** ersetzen die entsprechenden Variablen im Quellcode (z.b.: int durch Rint) und beinhalten Aufrufe an Akteurklasseninstanzen

       **Akteurklasse:** visualisieren die Rolle auf der entsprechenden Stage

     – R-Klassen wissen was zu tun ist und die A-Klassen wissen, wie es zu visualisieren ist

     – Es gibt Animationsklassen für primitive Typen (z.B.: int, boolean) und array, queue, stack

  - jede Instanz einer Rollenklasse besitzt eine eindeutige ID, die die gewünschten Objekte mit den Instanzen, die während der Ausführung geschaffen werden, verbindet.

- Erzeugung einer Hierarchie animierende Objekte
  **Klassenknoten:** Name der Klasse
  **Methodenknoten:** Name der Methode, Typ des Rückgabewertes, Typ der Argumente
  **Datenobjektknoten:** Name des Objektes, Typ, eindeutige ID (für Animation nötig)
- Abspeichern des erzeugten Codes in eine Datei

3. Start des Java Compiler

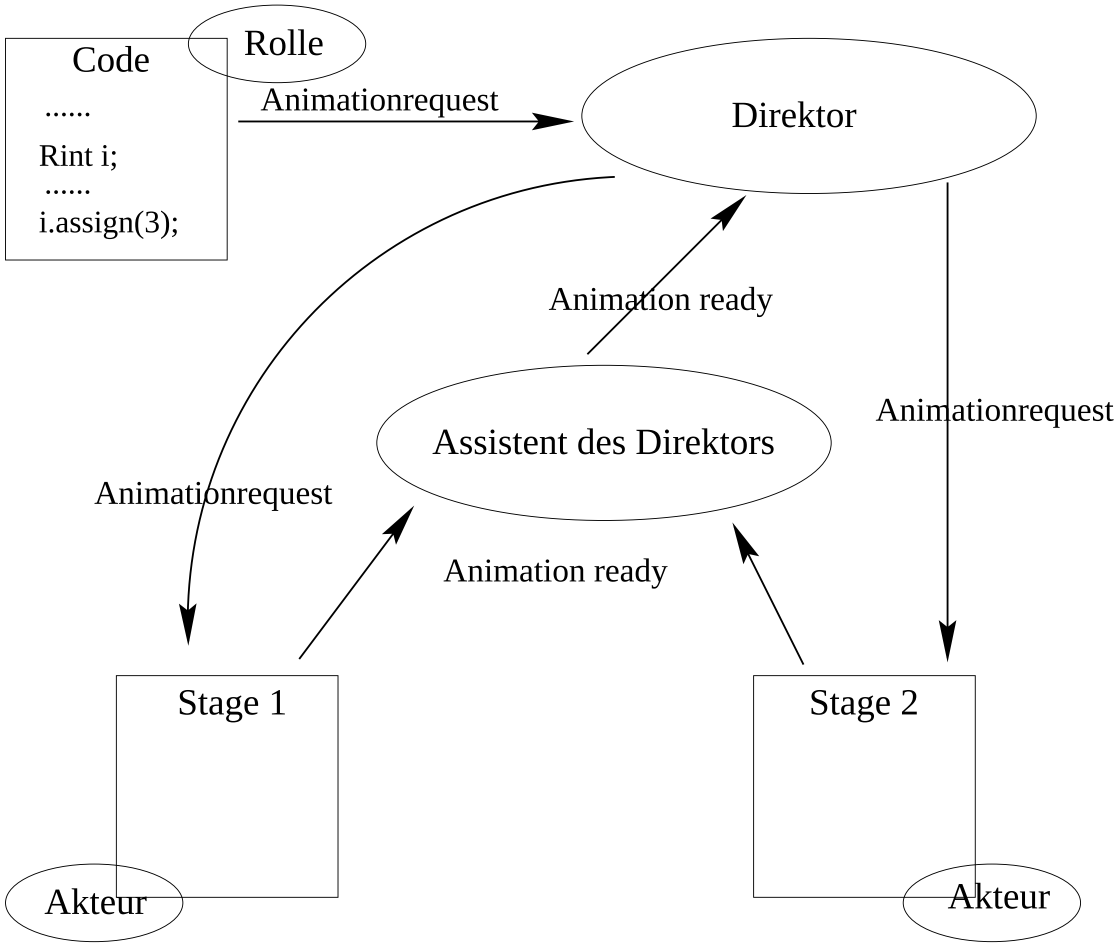
```
public class Beispiel
{
 public void init(){
  int j=5;
  int k;
  k = 20;
  while (k>j) {
    System.out.println(k);
    k--; }
}}
```

```
import jeliot.roles.*;
public class Beispiel
{
 public void init (){
  Rint j=new Rint(1,5);
  Rint k=new Rint(2);
  k.assign(20);
  while (k.compare(">",j)){
    System.out.println(k.retrieve()) ;
    k.assign("-++",-1);}
}}
```

- KT versendet den Namen des zu ladenen Klassenfiles als Nachricht

- Animator
  - zur Animation wird Animationsapplet Klasse vom Server geladen
  - handhabt Anfragen von dem Applet
  - beinhaltet Direktor, Stages, Kommuninkation zwischen Direktor und Stages, GUI
  - animiert Operation und übergibt Kontrolle wieder an Applet

# Animator


Code
Rolle
......
Rint i;
......
i.assign(3);
Animationrequest
Direktor
Animation ready
Animationrequest
Assistent des Direktors
Animationrequest
Animation ready
Stage 1
Stage 2
Akteur
Akteur

- Animation
  - Direktor leitet Animationsaufrufe der RKI an alle Stages weiter
  - Jede Stage überprüft, ob diese Variable auf ihr animiert wird. Falls diese Variable gefunden wird, wird der Aufruf an die Variable weitergeleitet, die die Animationsroutinen aufrufen. Die Animationsroutinen befinden sich in der Klasse JAPI (Jeliot Animation Primitive Interface).
  - z.Zt. werden nur Vergleiche und Zuweisungen unterstützt
  - zur Synchronisierung wird ein Prozess (Assistent des Direktors) benutzt.

# EJava

- Eigenständige Sprache
- beinhaltet selbstanimierende Datentypen (Rstack, Rqueue, Rint, Rbyte...)
- Funktionalität ist eingeschränkt
  - Dimensionsbeschränkung bei Feldern
  - keine Superklassen möglich, außer der Applet Klasse
  - mehr reservierte Worte als in Java
  - Unicode Characters sind nicht erlaubt

# Fazit

- Vorteile:
  - geringer Arbeitsaufwand, um Animation zu erstellen
  - gute Bedienung
  - mehrere Stages (Views)
  - Plattformunabhängig
  - Erweiterbar
- Nachteile:
  - arbeitet zu nah am Code
  - Restriktionen

- Aussichten:
  - z.Zt. ist das System zu komplex um von Studenten während ihres Studiums weiterentwickelt zu werden
  - neue komplexe animierbare Datentypen sollen hinzugefügt werden (binäre Bäume, Listen)
  - Java lernen mit Hilfe von Jeliot
  - Benutzer soll mehr Möglichkeiten erhalten, die Akteure zu manipulieren

http://verosaari.cs.helsinki.fi:8807/Jeliot/classes/Jeliot.html