

JAVA STARS NRW
Sun Microsystems Award 2004

Projekttitlel:

TuringSim

Übersicht

TuringSim ist ein Simulator der von Alan Turing entworfenen Turing Maschine

Schule:

Name **Richard von Weizäcker Berufskolleg**
Anschrift Friedrich-Ruin-Str. 61
48249, Dülmen
Tel.: 02594783030

Projektteam:

Name **Thorsten Hake**
Anschrift Heifoer 21
48249, Dülmen
Tel.: 025904408
Email: deichblach@t-online.de
Jahrgang: 1986
Klasse: HG28i

Name **Stefan Kirschner**
Anschrift Tiberweg 4
48249, Dülmen
Tel.: 02594909147
Email: s-kirschner@web.de
Jahrgang: 1986
Klasse: HG28i

Name **Sebastian Schenk**
Anschrift Perdekamp 7
48249, Dülmen
Tel.: 02594784466
Email: basti.schenk@web.de
Jahrgang: 1985
Klasse: HG28i

Fachlehrer:

Name **Hubert Harnack**
Anschrift Kleefeld 17
59348 Lüdinghausen
Tel.: 0259121484
Email: HHAR850142@aol.com

Inhaltsverzeichnis

1	Projektbeschreibung.....	1
1.1	Unterrichtsfach.....	1
1.2	Thema.....	1
1.3	Projektidee.....	1
1.4	Nutzen für den Unterricht.....	1
2	Lösungskonzept.....	2
2.1	Aufbau der Lösung.....	2
2.2	Eingesetzte Verfahren.....	2
3	Programm-Architektur.....	4
3.1	Übersicht.....	4
3.2	Frontend.....	4
3.2.1	Attribute.....	4
3.2.2	Methoden.....	4
3.3	Befehl.....	6
3.3.1	Attribute.....	6
3.3.2	Methoden.....	6
3.4	dateiFilter.....	7
3.4.1	Attribute.....	7
3.4.2	Methoden.....	7
3.5	Debugger.....	8
3.5.1	Attribute.....	8
3.5.2	Methoden.....	8
3.6	KontextMenue.....	9
3.6.1	Attribute.....	9
3.6.2	Methoden.....	9
3.7	MagnetBand.....	10
3.7.1	Attribute.....	10
3.7.2	Methoden.....	10
3.8	saveClass.....	11
3.8.1	Attribute.....	12
3.8.2	Methoden.....	12
3.9	TableRenderer.....	12
3.9.1	Attribute.....	12
3.9.2	Methoden.....	13
3.10	UndoManager.....	13
3.10.1	Attribute.....	13
3.10.2	Methoden.....	13
3.11	UtilClass.....	14
3.11.1	Attribute.....	14
3.11.2	Methoden.....	14
3.12	ZustandsAnzeige.....	16
3.12.1	Attribute.....	16
3.12.2	Methoden.....	16
4	Benutzerschnittstelle.....	17
4.1	Konzept.....	17
4.2	Befehl eingeben.....	18
4.3	Zustand hinzufügen/entfernen.....	18
4.4	Gap einstellen.....	18
4.5	Anfangszustand und Endkonfiguration einstellen.....	19
4.6	Alphabet und Eingabewort bestimmen.....	19
4.7	Starten, stoppen, steppen und zurücksetzen der Turingmaschine.....	19
4.8	Rückgängig und Wiederherstellen.....	20
4.9	Cut, Copy and Paste.....	20
4.10	Neu, laden, speichern.....	21
4.11	Drucken.....	21
4.12	Dokumentation des Programmes.....	22

5 Referenzen.....	23
5.1 Verwendete Quellen für die Programmierung des Programmes:.....	23
5.2 Verwendete Entwicklungsumgebung:.....	23
6 Anlagen.....	24
6.1 Beispielprogramm „Dual-Dezimal.uce“.....	24
6.2 Flashanimation: „Tutorial zu TuringSim“.....	24
6.3 Quellcode zum TuringSim.....	24
6.4 Ausführbare .jar Datei zu TuringSim.....	24

1 Projektbeschreibung

Das Projekt „TuringSim“ ergab sich durch den im Informatikunterricht behandelten Stoff der theoretischen Informatik. Wo wir unter anderen auch mit der Turing Maschine konfrontiert worden sind. Eine Turing Maschine ist eine Maschine mit einem Magnetband und einem Schreib-Lese(SL)-Kopf, sie arbeitet nach einem vorgegebenen Flussdiagramm. Sie kann den SL-Kopf in Schritten nach links und rechts auf dem Magnetband bewegen. Die Befehle im Flussdiagramm haben die Form zuSchreibendesZeichen, nächsterZustand, bewegRichtungdesSLKopfes . Das Band ist beim Start der Turing Maschine mit einer Anfangsbelegung belegt, auf dieser Belegung wird dann das Flussdiagramm angewendet. Die Turing Maschine geht von ihren Anfangszustand bis zu ihren Endzustand das Flussdiagramm durch.

1.1 Unterrichtsfach

Informatik

Theoretische Informatik

1.2 Thema

Simulation einer Turing Maschine

1.3 Projektidee

Die Projektidee kam uns, als wir festgestellt hatten, wie unkomfortabel die Turing Simulations Programme an unserer Schule waren. Features die eigentlich standard sein sollten, wie zum Beispiel Kopieren und einfügen, waren in dem Simulationsprogramm nicht enthalten. Also beschlossen wir einen Turing Simulator zu programmieren der Anwendungsfreundlicher ist. -> TuringSim

1.4 Nutzen für den Unterricht

TuringSim macht es leichter die einfach Funktionsweise einer Turing Maschine zu verstehen. Außerdem weckt es mehr Interesse kleine Programme auf einem Turing Simulator zu schreiben bei denen man erkennen kann das sie funktionieren, als Flussdiagramme auf ein weißes Blatt Papier zu zeichnen, ohne irgendeine Ahnung, ob und wie das Flussdiagramm in Wirklichkeit funktioniert.

2 Lösungskonzept

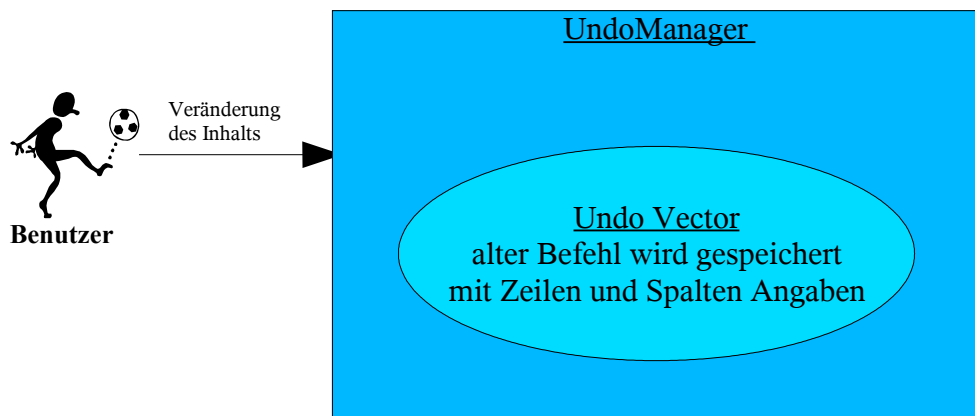
2.1 Aufbau der Lösung

Das Flussdiagramm der Turingmaschine kann über eine Tabelle, bei welcher jedes Element des Alphabets der Turingmaschine eine Spalte und jeder Zustand der Turingmaschine eine Zeile darstellt, eingegeben werden. Der Anwender kann das Alphabet und das zu verarbeitende Eingabewort selber bestimmen und eingeben. Wenn der Benutzer alle Befehle im Flussdiagramm über einen JDialog oder über copy & paste eingegeben hat, wobei die Eingaben nach der richtigen Syntax überprüft werden, und alle Einstellung betreffend des Startzustandes und der Endkonfiguration gemacht hat, kann er den Ablauf des Programmes über Buttons starten. Während des Ablaufes geht das Programm von Zustand zu Zustand und führt die Befehle aus. Da die programmierten Programme auf dem TuringSim natürlich einen Autor haben, kann man sich im Teil „Dokumentation“ des Programmes zusammen mit einem Beispiel für ein Eingabewort, einen Beispiel für einen Ausgabezustand und der Funktion des Programmes eintragen.

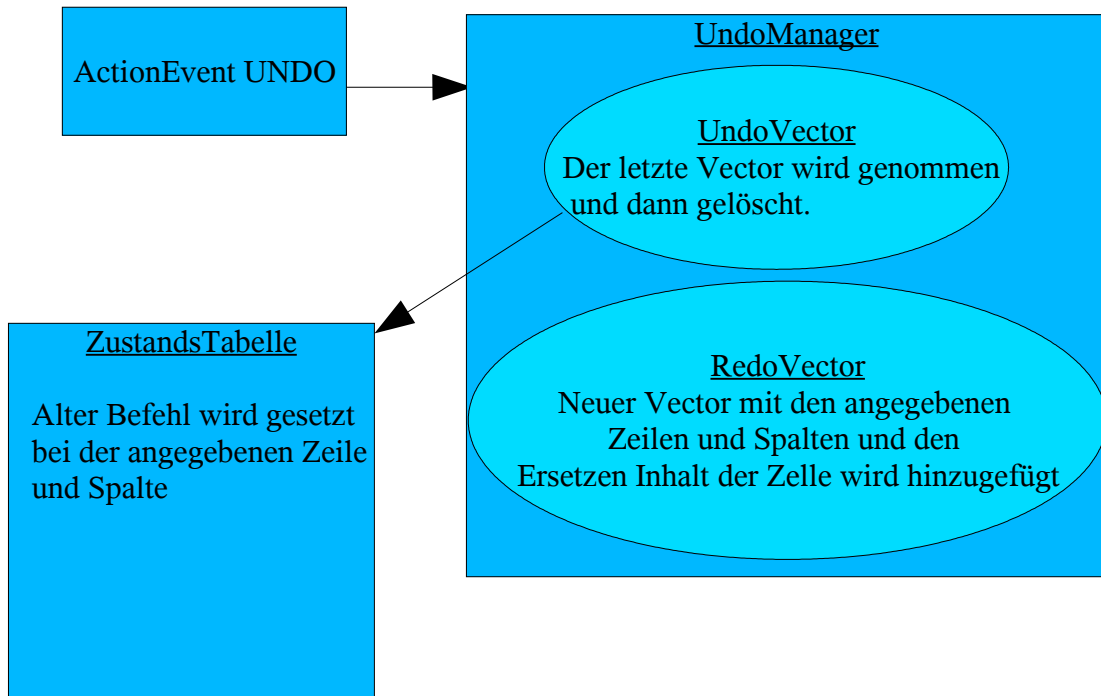
2.2 Eingesetzte Verfahren

Bei der Findung einer benutzerfreundlichen Lösung eines Turingsimulators haben wir großen Wert auf eine große Benutzerfreundlichkeit und eine leichte Bedienung gelegt. Denn was bringt das beste Programm, wenn man Stunden brauch um sich in das Programm einzuarbeiten. Deswegen haben wir die Oberfläche des Programmes nicht allzu sehr überlastet, damit der Benutzer nicht den Überblick verliert. Zu der Benutzerfreundlichkeit trägt weiterhin hinzu, dass der Anwender des Programmes den Inhalt der Zellen kopieren und an anderer Stelle wieder einfügen kann. Dieses wurde über eine Verbindung zum betriebssysteminternen Clipboard bewerkstelligt, in/aus welchem dann die Inhalte programmübergreifend gespeichert bzw geladen werden. Des weiteren haben wir uns einen so genannten „UndoManager“ für die Speicherung, Organisation und Verwaltung von durchgeführten Veränderungen an der Tabelle programmiert um ebend diese zu erfassen, da sich das `java.swing.undo` package unseres Wissens nicht auf `JTable` und deren Zelleninhalte bezieht. Mithilfe dieses UndoManagers ist es uns möglich, die „Rückgängig machen“-Funktion und die „Wiederherstellen“-Funktion auf unserer `JTable` anzubieten. Der UndoManager hat zwei Vektoren um sich die Undo- und Redo-Steps zu merken. Die nachfolgende Zeichnung soll klar machen, wie dieser „UndoManager“ funktioniert:

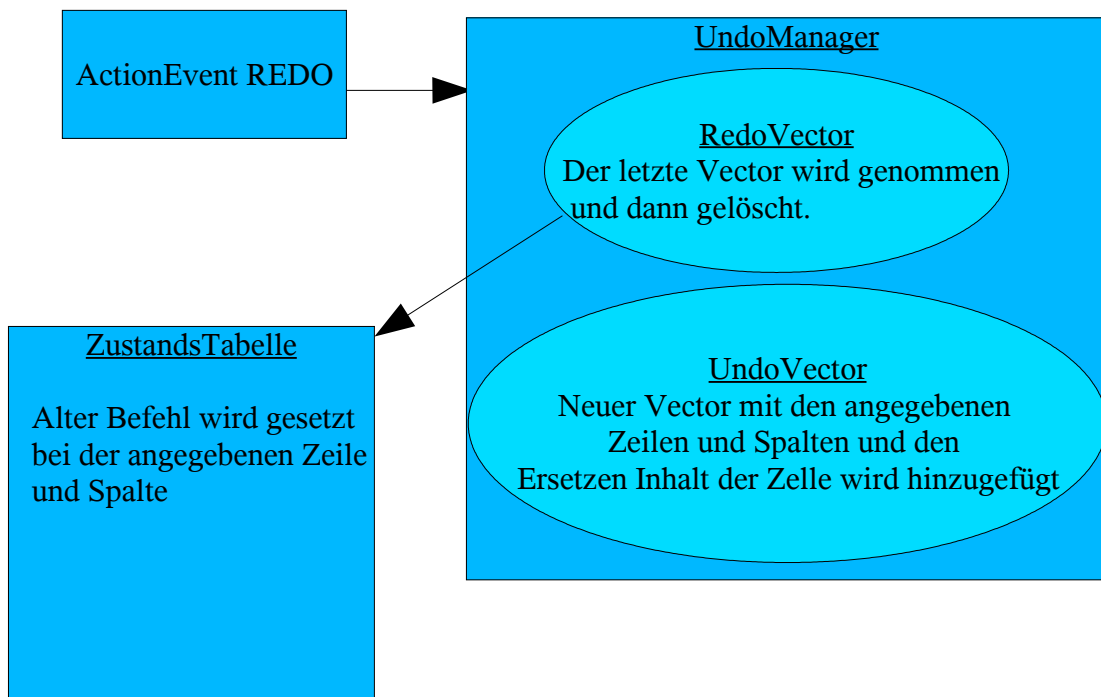
Bei Veränderung des Inhalts:



Bei Undo Aufruf:



Bei Redo Aufruf:



3 Programm-Architektur

3.1 Übersicht

Das Programm besteht aus 16 Klassen, einige wichtige sind hier Aufgeführt.

Die UML Übersicht(Klassendiagramm) des Projektes befindet sich auf der CD mit dem Namen „UML_Struktur.JPG“

Nachfolgend sind die wichtigsten Klassen des Programmes noch genauer erläutert.

3.2 Frontend

Beschreibung:

Wie alle anderen Klassen auch, hat die Klasse Frontend einen selbsterklärenden Namen: So ist diese Klasse für den Aufbau und das Funktionieren der Benutzerschnittstelle verantwortlich und erzeugt eine große Menge an SWING-Objekten und Objekten der anderen Unterklassen des Projektes. Ein Großteil der anderen Unterklassen werden von Frontend direkt verwendet.

Das macht diese Klasse auch zur größten Klasse des Projektes, da ja nicht nur für die Darstellung der Oberfläche sondern auch für die Leitung der Geschicke des gesamten Programms viele Objekte und damit viel Code benötigt wird.

Übersicht der Klasse:

Bild (größenbedingt) siehe „\frontend.jpg“

3.2.1 Attribute

s. Bild.

3.2.2 Methoden

```
public static void main(String[] args)
```

Die main-Methode ist die erste Methode, die bei der Programmausführung aufgerufen wird. Der args-Parameter wird von ihr nicht verwendet, denn sie erzeugt über die Methode „makeFrontendWithDefaultTableModel“ ein neues Programmfenster in dem für den Benutzer bereits ein neues leeres Simulationsprojekt erzeugt wurde.

```
public Frontend(String titel, int maximized, DefaultTableModel  
tabellenmodell)
```

Bei dieser Methode handelt es sich um den Konstruktor der Klasse und gleichzeitig des Prammfensters. Hier werden alle Oberflächenkomponenten erzeugt, Objekte miteinander verknüpft und Eventlistener registriert. Kurz, diese Methode stellt für den Benutzer eine Schnittstelle zum Programm her.

```
public static Frontend makeFrontendWithDefaultTableModel(  
String titel,  
int maximized,  
int anzahlZustaende)
```

Diese Methode ruft den Klassenkonstruktor auf und übergibt diesem ein neu erzeugtes DefaultTableModel, das der Konstruktor beim Erstellen des Fensters in die Ansicht integriert. Rückgabe hiervon ist ein neues Frontend-Objekt.

```
public static DefaultTableModel makeDefaultTableModel ()
```

„makeDefaultTableModel“ erzeugt ein neues Simulationsprojekt – es handelt sich um eine SWING-Tabelle mit zwei Spalten, deren erste Spalte der Nummerierung dient und in deren zweiten Spalte eine Zelle als Endzustand markiert wurde. Diese Methode wird von makeFrontendWithDefaultTableModel verwendet um den Frontend-Konstruktor die Tabelle zu übergeben.

```
public void actionPerformed(ActionEvent e)
```

Diese Methode wird bei jedem vom User auf dem Frame hervorgerufenen Ereignis aufgerufen. Sie reagiert auf die Klicks jedes Buttons, und gibt dem Programm damit erst die Möglichkeit, überhaupt das zu tun was der Benutzer verlangt.

```
public int getFehlendenZustand ()
```

Diese Methode optimiert das Hinzufügen neuer Zustände. Ist die Nummerierung der Zustände (bspw. durch vorheriges Löschen anderer Zustände) nicht lückenlos, so ermittelt diese Methode die Zustandsnummer die erste Nummer innerhalb der Lücke, und gibt ansonsten die Nummer zurück, die direkt auf die Nummer des letzten Zustands folgt.

```
public void dateiLaden ()
```

DateiLaden ruft ein FileChooser-Fenster auf, in dem der User eine Simulationsdatei auswählen kann und lädt das Projekt anschließend ins Programm (Zustandstabelle, Dokumentation, Alphabet, usw.).

```
public void dateiSpeichern ()
```

Siehe „dateiLaden“, nur in die andere Richtung (Programm->Datei).

```
public void changeAlphabet ()
```

Diese Methode setzt vom User gemachte Änderungen am Alphabet programmintern durch. Nach einer Überprüfung des neuen Alphabets (Zeichen doppelt, nicht einzelne Zeichen durch Kommata abgetrennt, usw.) wird das Alphabet neu gesetzt und die Spalten der Zustandstabelle entsprechend angepasst.

```
public void changeWord ()
```

Hier werden vom Benutzer hervorgerufene Veränderungen an der Eingabemenge durchgeführt. Zuvor wird die Menge auf Gültigkeit (in Anbetracht des Alphabets) überprüft und dann die neue Eingabemenge (wenn die Simulation noch nicht lief) auf das MagnetBand schreibt.

```
public void addZustand ()
```

Diese Methode fügt der Zustandstabelle einen neuen Eintrag hinzu.

```
public void removeZustand ()
```

Hier wird der aktuell fokussierte Zustand aus der Tabelle entfernt.

```
public void makeTable(DefaultTableModel tablemodel)
```

Zur besseren Übersicht und Differenzierung wird das Erzeugen der Zustandstabelle vom Fensterkonstruktor abgekapselt. Hier wird anhand eines übergebenen Tablemodels eine neue Tabelle erstellt.

```
public void changeLook(String name)
```

ChangeLook macht es für den Benutzer möglich, dem Programm ein anderes Look and Feel zu geben.

3.3 Befehl

Beschreibung:

Diese Klasse ist dafür zuständig, dem Benutzer die Eingabe von neuen Befehlen für die Zustandstabelle zu ermöglichen. Die Klasse erzeugt dabei einen JDialog, der die Daten aufnimmt. Nach der Dateneingabe wird der neue Befehl als String in die Zustandstabelle eingetragen.

Übersicht der Klasse:



3.3.1 Attribute

s. Bild

3.3.2 Methoden

public Befehl(Frontend owner)

Hier handelt es sich um den Konstruktor der Befehl-Klasse. Er erzeugt das Dialogfenster zur Befehlseingabe und setzt dabei die verwendeten Komponenten auf den Dialog, registriert die Eventlistener und lädt ggf. Daten aus dem Tabellenfeld für das der Dialog aufgerufen wurde.

```
public String get[...] ()
```

Diese Methoden gibt den Wert einer der privaten Eigenschaften „Bandspule, EigenerZustand, NextZustand“ etc. zurück.

```
public void setNextZustand(String string)
```

Hier wird der Wert einer der privaten Eigenschaften „Bandspule, EigenerZustand, NextZustand“ etc. gesetzt.

```
public void stateChanged(ChangeEvent e)
```

Diese Methode reagiert auf Veränderungen der RadioButtons, mit denen die SL-Kopf Bewegung nach Verarbeiten des eingegebenen Zustands bestimmt werden, und ändert entsprechend dem neuen Komponentenstatus den Text des Befehls ab.

```
public void actionPerformed(ActionEvent e)
```

ActionPerformed reagiert auf Ereignisse auf dem Dialog und ermöglicht dem Programm so, Buttonklicks zu verarbeiten.

3.4 dateiFilter

Beschreibung:

Diese Klasse sorgt beim Öffnen von Simulationsdateien im FileChooser-Dialog dafür, dass in der Auswahlsicht nur Ordner und .uce-Dateien sichtbar sind. Sie filtert also alle Dateien mit anderer Endung aus der Ansicht heraus. Es werden mehrere Methoden der Mutterklasse FileFilter überschrieben.

Übersicht der Klasse:



3.4.1 Attribute

s. Bild

3.4.2 Methoden

```
public boolean accept(File f)
```

Für jedes Objekt das der FileChooser-Dialog in einem Verzeichnis findet ruft der File-Filter diese Methode auf, in der bestimmt werden kann ob die jeweilige Datei vom Dialog angezeigt werden soll oder nicht (boolean). Ist das Objekt ein Verzeichnis oder endet dessen Name mit „uce“, so wird true zurückgegeben, ansonsten false.

```
public String getDescription()
```

Gibt den String „uce-Dateien“ zurück, der vom FileChooser verwendet wird um den angezeigten Dateitypen zu beschreiben.

```
public static String getExtension(File f)
```

Diese Methode liefert die Dateinamenerweiterung des File-Objektes „f“ als String.

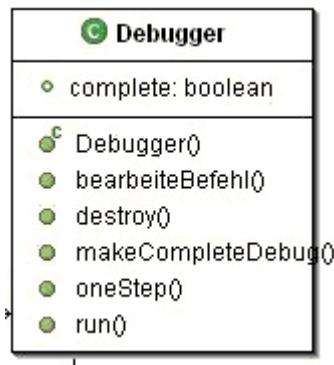
3.5 Debugger

Beschreibung:

Die Klasse Debugger ist funktionstechnisch das Herz des gesamten Programmes, denn sie ist dafür zuständig, das Programm in der Zustandstabelle ablaufen zu lassen. Debugger springt in den vordefinierten Startzustand, führt dort zunächst einen Test, dann eine Funktion auf dem jeweiligen Register aus und springt zum hier definierten Folgezustand. So lange, bis entweder das Programm die „-Ende“-Marke findet, oder bis der Anwender den Ablauf von sich aus beendet.

Des weiteren sei zu erwähnen dass es sich bei „Debugger“ um eine von Thread erbende Klasse handelt, die Oberfläche wird also nicht vom Rechenzeitverbrauch des Debuggers beeinflusst.

Übersicht der Klasse:



3.5.1 Attribute

s. Bild

3.5.2 Methoden

```
public Debugger(MagnetBand band, Frontend frontend)
```

Der Konstruktor dieser Klasse erzeugt ein neues Debugger-Objekt und stellt Objektverbindungen zum Magnetband und zum Frontend her.

```
public void run()
```

Run dient dazu, den Debugvorgang zu differenzieren. Der Anwender hat die Möglichkeit, entweder das gesamte Programm oder nur einen einzigen Schritt des Programmes ausführen zu lassen. Dies wird hier über den boolean „complete“ realisiert. Je nach Wert wird hierbei eine andere Methode aufgerufen („oneStep“ oder „makeCompleteDebug“).

```
public String bearbeiteBefehl(String zustand)
```

BearbeiteBefehl ist – spirituell ausgedrückt – das Zentrum des Herzens. Diese Methode übernimmt den Kern des Debuggings; sie erhält Befehle aus der Zustandstabelle (über den Parameter „zustand“) und „befolgt“ sie.

```
public void oneStep()
```

OneStep macht einen Debugging-Schritt innerhalb des Flussdiagramms. Hier wird der Zustand ausgelesen und beim entsprechenden Aufruf an „bearbeiteBefehl“ übergeben, was

den Befehl verarbeitet. Zudem wird hier ein Erreichen des Programmendes und ein Erreichen einer nicht definierten Marke behandelt.

```
public void makeCompleteDebug ()
```

Im Gegensatz zu „oneStep“ wird in „makeCompleteDebug“ das komplette Programm durchlaufen bis das Ende erreicht ist, der User den Ablauf beendet oder bis der Debugger eine nicht definierte Marke gefunden hat.

```
public void destroy ()
```

Dies hält den Thread an und gibt dessen Speicher wieder für das System frei. Um nicht nach vielen Abläufen mehrere vollschlanke Thread-Leichen im Speicher zu haben wird dies nach dem Debuggen aufgerufen.

3.6 KontextMenue

Beschreibung:

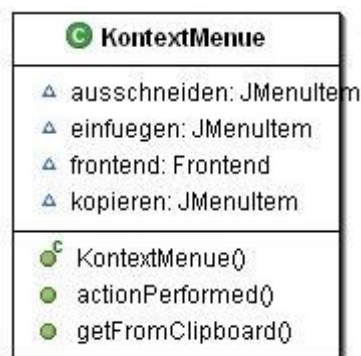
Die Kontextmenüklasse, die bei Rechtsklick auf die Tabelle ein Kontextmenü anzeigt mit drei Einträgen:

Kopieren: Zum kopieren der Zelleninhalte

Ausschneiden: Zum ausschneiden der Zelleninhalte

Einfügen: Zum einfügen der kopierten oder ausgeschnittenen Zelleninhalte

Übersicht der Klasse:



3.6.1 Attribute

s. Bild

3.6.2 Methoden

1. **public** KontextMenue(Frontend frontend)

Konstruktor: Erstellt die Menü Buttons und registriert diese im ActionListener

2. **public void** actionPerformed(ActionEvent e)

EventListener: Reagiert auf die Mausklicke auf dem Kontextmenü

3. **public static void** kopieren(String s)

Kopiert die selektierten Zellen und stellt diese im Clipboard systemweit zur Verfügung

4. **public** String getFromClipboard() {

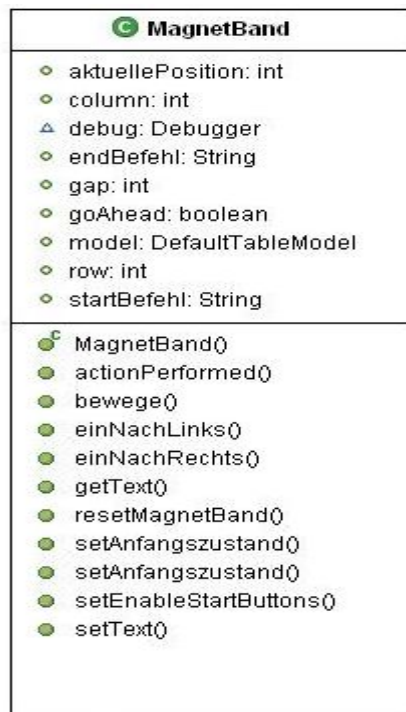
Liest die Clipboard befindlichen Daten und fügt diese, nach einer Syntaxüberprüfung, in die Tabelle ein.

3.7 MagnetBand:

Beschreibung:

Die Klasse MagnetBand ist für die Visualisierung und Organisation des physischen Magnetbandes zuständig.

Übersicht der Klasse:



3.7.1 Attribute

s. Bild

3.7.2 Methoden

1. **public** MagnetBand(DefaultTableModel a, Frontend frontend) {

Initialisiert das Magnetband (was eine Tabelle ist), und füllt diese mit „#“ - Zeichen.

2. **public void** bewege(String richtung)

Bewegt den Cursor auf der Tabelle sowie die Abbildung des Bereiches auf dem Magnetband in die in der Variable „richtung“ befindliche Richtung

3. **public void** einNachLinks() {

Diese Methode ist notwendig um nach links zu laufen auch wenn man in der Tabelle eigentlich schon in der Spalte 0 ist und man eigentlich nicht mehr nach links laufen kann

4. **public void** einNachRechts() {

Diese Methode ist notwendig um nach rechts zu laufen auch wenn man schon in der Spalte 19 ist und man eigentlich nicht mehr nach rechts laufen kann

5. **public** String getText() {

Gibt den Text auf dem Magnetband wider.

6. **public void** setText(String text) {

Setzt Text auf das Magentband

7. **public void** setAnfangszustand(String buchstaben) {

Der eingehende String wird geteilt und an setAnfangszustand (Vector buchstaben) übergeben

8. **public void** setAnfangszustand(String buchstaben) {

Diese Methode wird aufgerufen wenn sich entweder das Wort ändert, oder wenn das Magentband zurückgesetzt wird. Desweiteren werden die alten Zellen des Magentbandes überschrieben und durch das neue Wort ersetzt. Außerdem wird der Cursor auf 0 gesetzt, und die Mitschrift der Veränderungen (links, mitte, rechts) wieder auf den Anfangszustand gebracht. Der Startbefehl wird ebenfalls zurückgesetzt.

9. **public void** setEnableStartButtons(**boolean** aFlag) {

Macht die Buttons für die Benutzer zugänglich

10. **public void** resetMagnetBand() {

Setzt das Magentband-Objekt zurück

3.8 saveClass

Beschreibung:

Eine Abbildung der „.uce“ Dateien zum abspeichern des Programmes

Übersicht der Klasse:



3.8.1 Attribute

s. Bild

3.8.2 Methoden

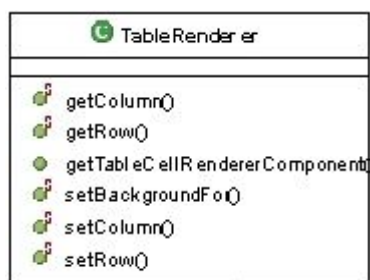
1. **public void** speichern(Frontend frontend) {
 Speicher das aktuelle Projekt

3.9 TableRenderer

Beschreibung:

Die Klasse rendert die ZustandsTabelle der Frontend Klasse. Sie ist nötig damit man den Hintergrund einer Speziellen Zelle verändern kann.

Übersicht der Klasse:



3.9.1 Attribute

s. Bild

3.9.2 Methoden

1. **public class** TableRenderer **extends** DefaultTableCellRenderer {
Diese Klasse stellt den ZellenRenderer einer jeden Zelle der ZustandTabelle
2. **public static void** setBackgroundFor(**int** row, **int** column, **boolean** marked) {
Diese Methode setzt die markierte Zelle. Wenn marked==false ist ist somit keine Zelle markiert
3. **public static int** getColumn() {
Gibt die Spalte zurück
4. **public static int** getRow() {
Gibt die Spalte zurück
5. **public static void** setColumn(**int** i) {
Setzt Spalte
6. **public static void** setRow(**int** i)
Setzt Zeile

3.10 UndoManager

Beschreibung:

Mit dem UndoManager werden alle Inhaltsveränderungen der Zustandstabelle des Frontends registriert und können rückgängig gemacht werden. Die Klasse musste erstelle werden da nach unseren Erkenntnissen eine Undo-Funktion bei der Tabelle seitens des Packages javaax.swing.undo.* nicht besteht.

Übersicht der Klasse:



3.10.1 Attribute

s. Bild

3.10.2 Methoden

1. **public** UndoManager(Frontend frontend) {
Dieser Konstrucktor setzt das attribut this.frontend=frontend

2. **public void** textChanged(String befehl, **int** row, **int** column) {
 Diese Methode fügt einen Vector undoSteps hinzu. In diesem Vector sehen die Parameter:
 Befehl: Beinhaltet den Befehl
 row: Beinhaltet die Zeilen Information
 column: Beinhaltet die Spalten Information

1. **public void** actionPerformed(ActionEvent e) {
 Hier werden bei jeder Aktion die von REDO und UNDO Actionquellen ausgehen die entsprechenden REDO und UNDO Funktionen aufgerufen, die Zellen verändert, der entsprechende REDO und UNDO Vector aus den undoSteps/redoSteps Vecotr gelöscht und dem redoSteps/undoSteps Vecotr hintergrund.

2. **private void** setRedoBefehl(String befehl, **int** row, **int** column) {
 Diese Methode fügt automatisch einen Vector zum Vector redoSteps hinzu in diesem Vector stehen die Parameter:
 Befehl: Beinhaltet den Befehl
 row: Beinhaltet die Zeilen Information
 column: Beinhaltet die Spalten Information

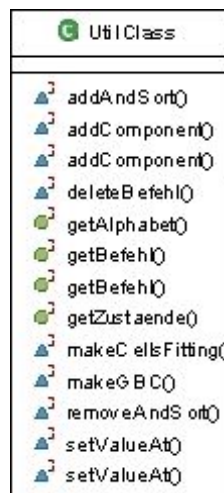
1. **public void** reset() {
 Der UndoManager wird zurückgesetzt

3.11 UtilClass

Beschreibung:

Diese Klasse beherbergt viele Methoden die von anderen Klassen verwendet werden.
 Eine Werkzeug Klasse.

Übersicht der Klasse:



3.11.1 Attribute

s. Bild

3.11.2 Methoden

1. **public static** Vector getZustaeende(JTable table) {
 Diese Mehtode liefert einen Vecotr zurück, in welchem alle Zustände der Tabele sind. Die Zustände sind in diesem Fall alle Inhalte der ersten Spalte
2. **public static** Vector getAlphabet(JTable table) {
 Diese Methode liefert einen Vector zurück, in welchen das Alphabet von der Tabelle ist. Das Alphabet ist hier die Namen aller Spalten.
3. **public static** String getBefehl(String zuSchreiben, String zuStand, JTable table) {
 Diese Mehtode sucht mit der Hilfe der Kombination aus zuSchreiben und Zustand aus der Tabelle den Befehl der an dieser Stelle steht. Der Befehl ist der Rückgabewert.
4. **public static** String getBefehl(
 Diese Methode mach genau das gleich wie gehtBefhel() jedoch wird die Zelle gleich markiert. Markiert wird sie allerdings nur dann, wenn sie den TableRenderer als Renderer.
5. **static void** deleteBefehl(String befehl, JTable table) {
 Die Tabelle wir nach einem spezifischen Befhel abgesucht, welche dann gelöscht wird.
6. **static void** addComponent
 Diese Methode fügt einen GridBagLayout eine Komponente hinzu:
 cont Der Container
 gbl Der GridBagLayout-LayoutManager
 c Die Komponente
 x Die x-Position innerhalb des Containers
 y Die y-Position innerhalb des Containers
 width Die breite der Komponente
 height Die höhe der Komponente
 weightx Das "x-Gewicht" der Komponente
 weighty Das "y-Gewicht" der Komponente
 a Der Anker der Komponente
7. **static** GridBagConstraints makeGBC(
 In dieser Mehtode werden die GridbagConstraints erzeugt
8. **static void** addComponent(
 Fügt eine Komponente hinzu
9. **static void** makeCellsFitting(JTable table) {
 Mit dieser Mehtode werden die Zellen der Tabelle auf einen bestimmten Wert gesetzt
10. **static void** setValueAt(
 Es wir eine neuer Befehl mit hilfe der Koordinaten gesetzt
11. **static void** setValueAt(
 Ein Befehl wird an der aktuellen Position gesetzt
12. **static void** addAndSort(JComboBox box, String item) {
 Es wir dein String in der JcomboBox hinzugefügt und der dann entstehende Inhalt der JComboBox wird sortiert.
13. **static void** removeAndSort(JComboBox box, String item)

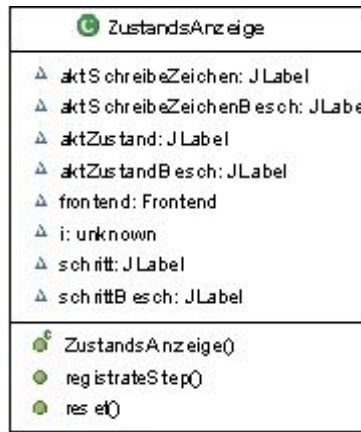
Es wird ein String einer JComboBox gelöscht und der Inhalt der JComboBox wird sortiert

3.12 ZustandsAnzeige

Beschreibung:

Diese Klasse ist für die Zustandsanzeige zuständig. Die Zustandsanzeige besteht aus einem JLabel für den aktuellen Zustand, einem JLabel für das aktuell zu schreibene Zeichen, und einem JLabel für die gemachten Schritte beim durchlauf des Flussdiagramms.

Übersicht der Klasse:



3.12.1 Attribute

s. Bild

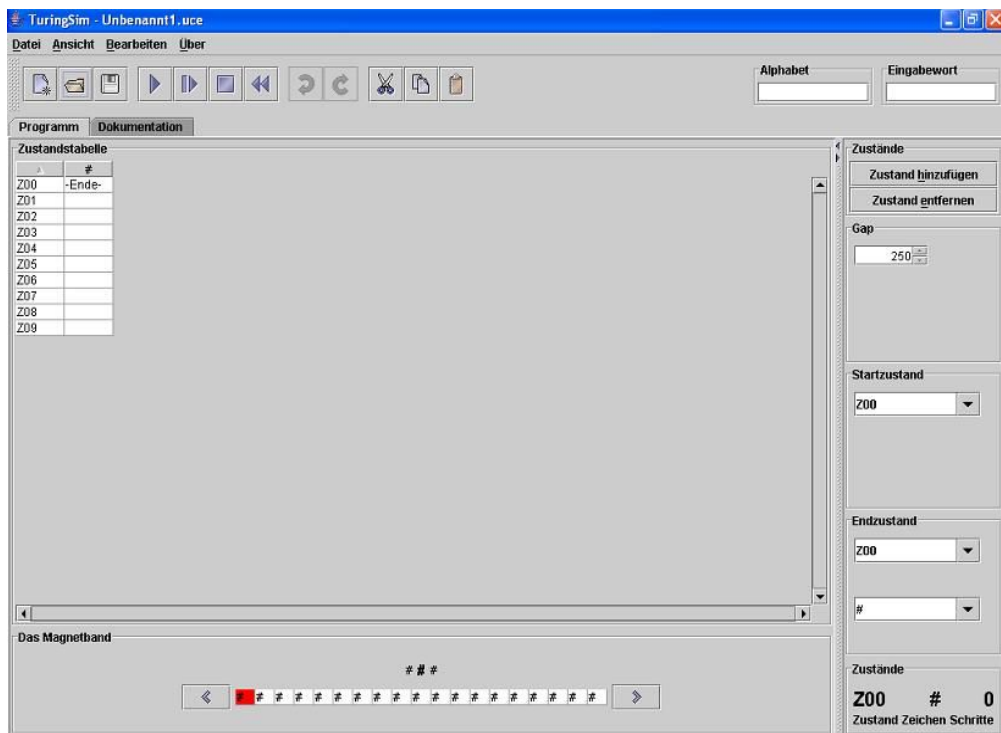
3.12.2 Methoden

7. **public class** ZustandsAnzeige **extends** JPanel {
Konstruktor baut die Zustandsanzeige auf
8. **public void** registrateStep(String befehl) {
Diese Methode wird aufgerufen wenn ein Schritt gemacht wurde um die Anzeige zu aktualisieren
9. **public void** reset() {
Die Anzeige wird auf den Anfangszustand zurückgesetzt

4 Benutzerschnittstelle

4.1 Konzept

Der Benutzer hat ein JFrame als Schnittstelle zum Programm. Die Oberfläche ist so konzipiert worden, das in möglichst kurzer Zeit jede Funktion des Programmes erreicht werden kann, ohne langes Suchen. Die Buttons sind zu diesem Zweck alle mit Mnemonics ausgestattet, die es dem Benutzer erlauben, schnell auf die Funktion des Buttons zuzugreifen. Außerdem kann der Benutzer selber entscheiden, ob die Zustandstabelle mit dem Magnetband den ganzen Platz des ContentPane ausmachen, oder „nur“ 4/5tel des Platzes. Man sieht, das der Benutzer viele Möglichkeiten hat, das visuelle Erscheinen des Programmes zu verändern, unter anderen auch durch die 4 integrierten Look and Feels. Hier können Sie einen verkleinerten Screenshot des Programmes im Startzustand sehen:



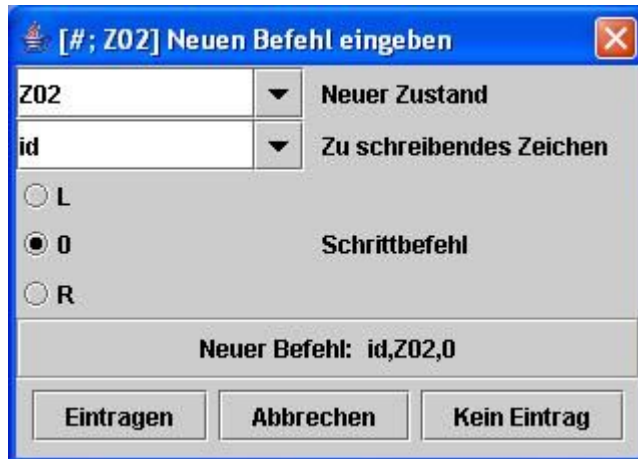
Die nachfolgenden 11 Funktionen des Programmes geben einen kleinen Eindruck über die Möglichkeiten von TuringSim.

4.2 Befehl eingeben

Beschreibung:

Damit der Benutzer einen Befehl in die Tabelle eintragen kann, muss er auf eine Zelle doppelklicken. Daraufhin öffnet sich ein JDialog, wo er sich einen Möglichen Folgezustand, ein zu schreibendes Zeichen, und eine Bandlaufrichtung aussuchen kann.

Screenshot:



4.3 Zustand hinzufügen/entfernen

Beschreibung:

Wenn die Anzahl der Zustände verringert oder vergrößert werden soll, kann man dieses mit dem Buttons „Zustand hinzufügen“ und „Zustand entfernen“ machen. Wenn man einen Zustand entfernen will, muss man allerdings vorher die Zeile des zu entfernenden Zustand selektieren.

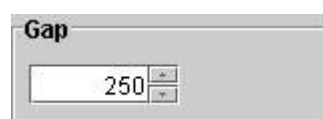
Screenshot:



4.4 Gap einstellen

Beschreibung:

Man kann sich selber entscheiden, ob man eine Lücke zwischen den Schritten der Turingmaschine während des Durchlaufs haben will. Diese Lücke kann man in diesem Spinner einstellen.

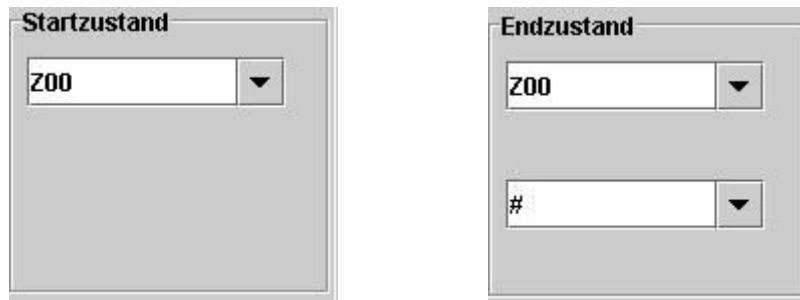


4.5 Anfangszustand und Endkonfiguration einstellen

Beschreibung:

Eine Turingmaschine braucht einen Anfangszustand und eine Endkonfiguration. Der Anfangszustand wird durch eine Combo Box bestimmt und die Endkonfiguration durch eine Kombination aus zwei Combo Boxen (eine mit dem Alphabet und die andere mit den Zuständen).

Screenshots:



4.6 Alphabet und Eingabewort bestimmen

Beschreibung:

Das Alphabet und das Eingabewort der Turingmaschine werden über zwei Textfelder eingegeben. Wobei sich das Alphabet aus mehreren einzelnen Zeichen besteht, und deswegen diese Zeichen auch mit einem ',' als Trennzeichen eingegeben werden müssen.

Screenshot:



4.7 Starten, stoppen, steppen und zurücksetzen der Turingmaschine

Beschreibung:

Damit die Turingmaschine gestartet wird gibt es oben in der Knopfleiste „Kontroll-Buttons“. Hiermit kann die Turingmaschine nur einen Schritt machen („step“), starten („start“), stoppen („stop“), und zurückgesetzt werden.

Screenshot:



4.8 Rückgängig und Wiederherstellen

Beschreibung:

Die Buttons zum Rückgängig machen und Wiederherstellen von Eingaben in der Zustandstabelle befinden sich einerseits im Menü unter Bearbeiten und andererseits in der Knopfleiste.

Screenshots:

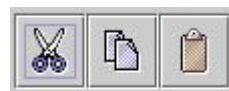
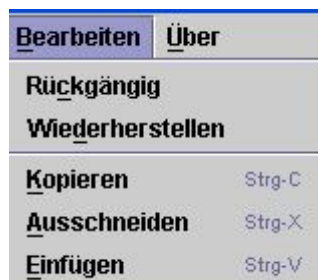


4.9 Cut, Copy and Paste

Beschreibung:

Zum benutzen der Cut, Copy und Paste Funktionen des Programms gibt es wieder mehrere Möglichkeiten. Zum einen kann man die Funktion über die Tastenkombinationen STRG+X (Cut) , STRG+C (Copy) und STRG+V (Paste) ansprechen, zum anderen aber auch noch über das Menü Bearbeiten und über die Buttons auf der Knopfleiste.

Screenshots:



4.10 Neu, laden, speichern.

Beschreibung:

Damit die Programmiererei des Turing Programmes nicht umsonst gewesen ist. Kann man das Aktuelle Turing Programm abspeichern. Dieses wird in eine „.uce“ Datei gespeichert, welche man an einem beliebigen Zeitpunkt wieder öffnen kann. Natürlich besteht auch die Möglichkeit ein neues Turing Programm zu schreiben, ohne den TuringSim neu zu starten. Die Funktionen speichern, laden, und neu stehen im Menü Datei und in der Knopfleiste zur Verfügung.

Screenshots:



4.11 Drucken

Beschreibung:

Wie man bereits schon am letzten Screenshot sehen konnte, besteht die Möglichkeit eine Seite einzurichten, und darauf dann die Zustandstabelle auszudrucken. Der Pfad zu diesen Aktionen ist „Datei -> Seite einrichten“ und „Datei -> Drucken“.

4.12 Dokumentation des Programmes

Beschreibung:

Man kann seinem mit TuringSim geschriebenen Turing-Programm mit Hilfe der Dokumentations-Pane eine Dokumentation mit den Informationen über den Autor, über die Eingabesyntax, die Ausgabesyntax und die Funktion des Programmes hinzufügen. Diese Dokumentation kann vom Autor passwortgeschützt werden, damit nur er die Beschreibung ändern kann. Das Passwort kann natürlich auch geändert werden.

Screenshot:

The screenshot shows a software window titled 'TuringSim' with two tabs: 'Programm' and 'Dokumentation'. The 'Dokumentation' tab is active. The interface is divided into several sections:

- Autor:** A single-line text input field.
- Ein korrektes Beispiel für ein Eingabewort der Turing Maschine:** A single-line text input field.
- Ein Beispiel für die Belegung des Bandes nach dem Ende des Durchlaufs:** A single-line text input field.
- Funktion des Programmes:** A large, empty text area for describing the program's function.
- Passwort-Schutz der Programmbeschreibung:** A section containing two text input fields labeled 'Passwort' and 'Passwort Wdh.', and a button labeled 'Passwort setzen'.

5 Referenzen

5.1 Verwendete Quellen für die Programmierung des Programmes:

[1] <http://java.sun.com>

[2] <http://www.java.de>

[3] <http://www.java-forum.org/de>

[4] <http://java.sun.com/docs/books/tutorial/uiswing/components/example-1dot4/TableSorter.java>

(Die Klasse TableSorter.java wurde komplett übernommen)

5.2 Verwendete Entwicklungsumgebung:

[1] [Eclipse](#)

6 Anlagen

6.1 Beispielprogramm „Dual-Dezimal.uce“

Die Codierung von Dual in Dezimal Zahlen auf einer Turingmaschine

6.2 Flashanimation: „Tutorial zu TuringSim“

Eine Flashanimation die in TuringSim einführen soll. Der Umgang wird anhand des Beispiels „Dual-Dezimal.uce“ erklärt.

6.3 Quellcode zum TuringSim

Der Quellcode befindet sich im Ordner „Quellcode“.

6.4 Ausführbare .jar Datei zu TuringSim

Die .jar Datei ist zu finden unter den Namen TuringSim1_3.jar