Höhere Technische Bundeslehranstalt
und Bundesfachschule
im Hermann Fuchs Bundesschulzentrum

C Compact

Diplomarbeit

Ausgeführt im Schuljahr 2014/2015 von:

Thomas Pointhuber, 5CHELS
Peter Wassermair, 5CHELS
Fabian Hummer, 5CHELS

Betreuer:

Dipl.-Ing. Franz Matejka

21. Dezember 2014

Arbeitstitel:
C Compact

Bearbeiter:

An der Diplomarbeit beteiligte Firmen:

Firma:	Firmenname
Adresse:	Strassenname, 16b
Plz, Ort:	5280, Braunau am Inn
Kontaktperson:	Dipl. Ing. Max Mustermann
Telefon:	+43 (0)7722 / 123 12 12
E-Mail:	max.mustermann@firmenname.com

Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als angegebene Quellen und Hilfsmittel nicht direkt benutzt und die benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Ort, Datum	*Verfasser Vor- und Zunamen*	*Unterschrift/en*

Inhaltsverzeichnis

Kapitel 1

Versionsgeschichte

Dieses Kapitel enthält eine Auflistung der Versionen von C Compact. Die Versionen werden wie folgt unterteilt:

- **Pre-Alpha:** Die Versionsunterteilung begann erst im Herbst 2014. Alle Versionen, die zuvor entstanden sind, sind nicht nummeriert. Das betrifft im Wesentlichen unsere Arbeit am Institut für Systemsoftware der Johannes Kepler Universität im Juli 2014.
- **Alpha:** Versionen mit dem Namen Alpha umfassen die Entwicklungen vom Herbst 2014 bis Anfang 2015.

Nachfolgend werden alle bisher benannten Versionen aufgelistet und beschrieben. Es ist zu beachten, dass Versuche mit Schülern immer am Beginn der aktuellen Version durchgeführt wurden. Die Entwicklung einer durch einen Versuch getestete Version wurde also durch den Versuch selbst wesentlich beeinflusst und optimiert.

Pre-Alpha

7. Juli 2014 bis 30.September 2014

- Diese Version umfasst die Arbeiten in unserer Zeit an der JKU sowie die grundlegende Implementierung der drei Hauptkomponenten des Projektes:
  - Compiler
  - Interpreter
  - Benutzeroberfläche
- Dateimanagement: Neue Datei - Öffnen - Speichern
- Datentyp ,,string" (ähnlich zu verwenden wie in Java)
- Datentyp ,,bool"
- Minimaler Präprozessor

Alpha 1.0

1. Oktober 2014 bis 3. November 2014

- Variablen können sowohl als Baumstruktur, als auch in Tabellen dargestellt werden
- Beim Schließen wird überprüft, ob die aktuelle Datei verändert wurde
- Implementierung einer Standardbibliothek
- Breakpoints
- Operatoren für Inkrementation und Dekrementation
- switch/case
- Schlüsselwort ,,nop"
- Verbesserte Tests mit Travis-CI Zuweisung der Arraygrößen mit Konstanten
- Elementarfunktion ,,printf"

Alpha 1.1

4. November 2014 bis 23. November 2014,
Getestet: 5. November 2014 mit der 2AHELS

- Variablenanzeige mit Tabellen wurde entfernt
- Variablenanzeige als Baumstruktur wurde wesentlich optimiert
- Step-over- und Step-out-buttons immer sichtbar
- Erkennen von Buffer Overflows und Variablenüberlauf
- Erkennen uninitialisierter Variablen
- Schlüsselwort ,,library"
- Übergeben von Arrays als Funktionsparameter
- Speichern von Laufzeitinformationen von Variablen

Alpha 1.2

24. November 2014 bis 14. Dezember 2014,
Getestet: 3. Dezember 2014 mit der 2BHELS

- Ausführungsmodus (Texteditor, Fehler, Automatisches Debuggen, Schritt-für-Schritt-Debuggen) wird sichtbar durch ein farbiges Panel über dem Textfeld gekennzeichnet.
- Step-over und Step-out Funktionen vorläufig entfernt
- HTML-Dokumente für unterschiedliche Fehler im Debugmodus (Compiler, Preprozessor, Laufzeitfehler)

- Tooltips für die wichtigsten Bedienelemente (Buttons, etc.)
- Logarithmische Skalierung der Zeitschritte beim automatischen Debuggen
- Die größten Elemente der Benutzeroberfläche sind verschiebbar und damit besser anzupassen
- Eigener Syntax Highlighter für die Sprache von C Compact
- Ein neuer Präprozessor sorgt für reibungslosen Ablauf und ermöglicht Bedingungen wie etwa ,,#ifdef"
- Die Zeile von Laufzeitfehlern wird korrekt erkannt und ausgegeben
- Hinzufügen spezieller Elementarfunktionen wie ,,time" und ,,__assert__" für Bibliotheken und fortgeschrittene Benutzer

Alpha 1.3

Ab 15. Dezember 2014

- Arrays werden jetzt wie in C deklariert
- Präprozessor kann Fehler werfen, die dem Benutzer angezeigt werden (wenn beispielsweise eine Bibliothek nicht gefunden wurde)
- Das Projekt wurde offiziell unter der GNU General Public License 3 lizenziert[1]
- Das Questsystem wurde in die Benutzeroberfläche integriert. Dazu gehören:
  - Ein Launcher, um ein Benutzerprofil auszuwählen
  - Ein neues Panel im Hauptfenster, in dem der Benutzer Informationen über die aktuelle Quest erhält und überprüfen kann, ob sein Sourcecode die Anforderungen der Quest erfüllt
  - Ein Auswahlfenster zum Selektieren und Starten einer Quest
  - Ein an das Questsystem angepasstes System für globale Einstellungen (um zum Beispiel für Schriftgröße oder die zuletzt verwendeten Benutzerprofile zu speichern)

[1] http://www.gnu.org/copyleft/gpl.html

Kapitel 2

Darstellung der Variablen

Die Anzeige der Variablen während der Laufzeit ist eine essentielle Funktion der Benutzeroberfläche. Die anschauliche und übersichtliche Darstellung soll dem Benutzer ein fundiertes Verständnis für den Ablauf des Programmes und die Rolle der Variablen darin, sowie deren Gültigkeitsbereiche vermitteln.

2.1 Unterschiede zu einem herkömmlichen Debugger

Die Idee der Variablendarstellung während der Laufzeit ist mit einem Debugger vergleichbar. Der Begriff Debugmodus kommt besonders im Sourcecode, aber auch in der Dokumentation und in der Benutzeroberfläche immer wieder vor. Allerdings unterscheidet sich die Darstellung von Programmabläufen und Variablen in C Compact in einigen Punkten von herkömmlichen Debuggern:

- Während die Debugger professioneller Entwicklungsumgebungen eine Hilfestellung für erfahrene Programmierer sind, ist der Debugmodus von C Compact besonders auf Anfänger ausgelegt.
- Die Darstellung der Variablen ist übersichtlich und fester Bestandteil der Beutzeroberfläche von C Compact. Einsteiger und unerfahrene Programmierer sollen von Anfang an mit dem Debugger in Berührung kommen.

2.2 Darstellungsform der Variablen

Zur Darstellung wurden zwei Konzepte verfolgt:

1. Die Anzeige des Call Stacks als Liste mit separaten Tabellen für die globalen und lokalen Variablen
2. Die Anordnung aller Variablen in einem Baum, wobei lokale Variablen Funktionen untergeordnet sind

In den ersten Versionen der Benutzeroberfläche wurden die Variablen wie in Punkt 1 beschrieben dargestellt. Bereits während unseres Praktikums am Institut für System-

software der JKU wurde die zweite Form der Variablendarstellung theoretisch entworfen. Die erste grundlegende Implementierung erfolgete noch während den Sommerferien.

In der ersten numerierten Version (Alpha 1.0) waren noch beide Darstellungsformen vorhanden und konnten vom Benutzer ausgewählt und gewechselt werden. Allerdings waren diese beiden Darstellungsformen sehr unterschiedlich implementiert, was Anfangs dazu führte, dass die Darstellung der Variablen als Baum nicht mehr als ein zusätzliches Feature ohne den vollen Funktionsumfang der Variablendarstellung als Call Stack war.

Für die Version Alpha 1.1 sollte ursprünglich das Variablendarstellungssystem so verändert werden, dass beide Anzeigemodi über ein gemeinsames Interface angesprochen werden können. Nach ausführlicher Besprechung und Evaluierung der unterschiedlichen Variablendarstellungen haben wir aber beschlossen, die Variablen nur als Baum darzustellen. Dadurch haben sich folgende Vorteile ergeben:

1. Die Implementierung einer neuen Funktion in der Variablenanzeige muss nur noch einmal erfolgen und nicht für jeden Anzeigemodus extra vorgenommen werden.
2. Daraus resultiert auch ein durchdachteres und einfacheres Bedienungskonzept der Benutezroberfläche, da dieses nur an eine Variablendarstellung angepasst werden muss.
3. Davon profitiert zuletzt der Benutzer, der eine komplette und sauber implementierte Anzeige und Erklärung der Variablen während des Programmablaufes vorfindet.

2.3 Darstellung der Variablen in getrennten Tabellen

Diese Form der Variablendarstellung wurde erstmals im Dokument Ausführungsumgebung für C– von Herrn Professor Blachek beschrieben und nach dieser Vorlage implementiert[1]. Ein Vorteil dieser Darstellungsform war die Übersichtlichkeit und Einfachheit besonders beim Debuggen von einfachen Programmen. Leider verlor das Konzept an Übersichtlichkeit, wenn fortgeschrittene Programme mit komplexen Strukturen abgearbeitet wurden. Eine Tabelle konnte jeweils nur eine Ebene der Variablen, also beispielsweise den Inhalt einer Struktur oder die lokalen Variablen einer Funktion anzeigen.

2.4 Darstellung der Variablen als Baum

2.4.1 Darstellungsschema

Die Darstellung der Variablen in Form einer Baumstruktur hat den Vorteil, dass alle Variablen in einem Feld untergrbracht sind und die Gültigkeitsbereiche der Variablen sofort ersichtlich sind.
Alle Elemente befinden sich in einem Ordner, der den Namen der Datei trägt. Dadurch

[1] Ausführungsumgebung für C—, Günther Blaschek, V1.0, 2014-06-18

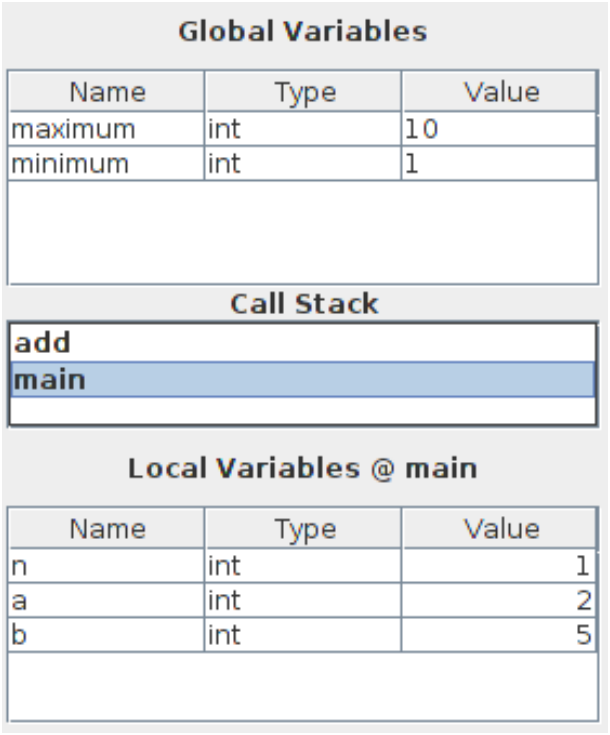


Abbildung 2.1: Darstellung der Variablen in separaten Tabellen

wird symbolisiert, dass die Variablen ein Teil des Programmes sind. Dem Programm selbst untergeordnet sind globale Variablen und Funktionen. Lokale Variablen sind Funktionen untergeordnet, Varibalen in Strukturen sind der Struktur untergeordnet.
Zusätzlich werden folgene Elemente in der Tabelle farbig markiert:

- Funktionen sind hellblau hinterlegt. Dadurch soll hervorgehoben werden, dass es sich hierbei nicht um einfache Datenstrukturen handelt.
- Die Variablen, die zuletzt geändert wurden, sind gelb hinterlegt. Grundsätzlich kann pro Schritt des Debuggers nur eine Variable geändert werden. Werden aber einige Schritte übersprungen, zum Beispiel mit dem Schlüsselwort ,,library", so werden mehrere Variablen in einem Debuggerschritt geändert.
- Variablen, die noch nicht initialisiert wurden, sind grau hinterlegt.

Bei Rechtsklick auf eine Variable im Baum wird ein Kontextmenü geöffnet. Mit der Option ,,Jump to declaration" kann der Benutzer zur Deklaration der Variable im Sourcecode springen. Die entsprechende Zeile wird markiert.

2.4.2 Grundlegende Komponenten

Die Variablen werden in einer Tabelle mit Baumstruktur, in Kommentaren im Sourcecode deshalb oft ,,TreeTable" genannt, angezeigt. Die TreeTable ist eine Kombination aus einem JTree und einer JTable.
Im Folgenden werden diese Komponenten beschrieben und der Aufbau der TreeTable erläutert:

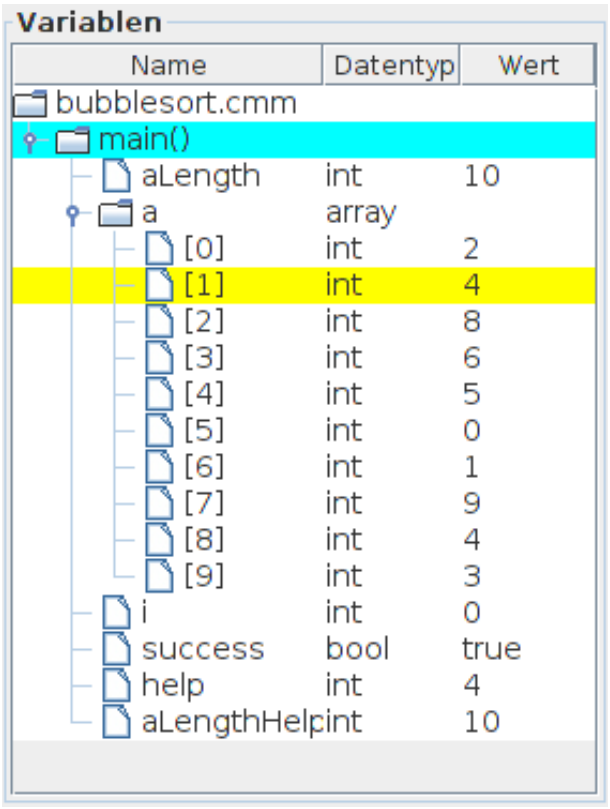


Abbildung 2.2: Darstellung der Variablen als Baum

JTree - Baum

Ein JTree[2] ist Swing-Element, das Daten hierarchisch untereinander darstellet. Die Daten sind allerdings nicht im JTree selbst gespeichert, sondern in externen Objekten. Ein JTree kann entweder mit einem Datenmodell oder mit dem Hauptknoten eines Baumes initialisiert werden[3]. Da die Daten eines JTree einen Baum abbildet, kann durch diese Struktur traversiert werden. Solche Operationen[4] werden beispielsweise benötigt, um alle Knoten eines Baumes zu öffnen (expand) oder zu schließen (collapse).
Wird ein JTree direkt mit einem Baum initialisiert, müssen die Knoten das Interface TreeNode implementieren[5]. Für einfache Aufgaben können Standardklassen, wie etwa DefaultMutableTreeNode[6] verwendet werden.
Mehr Möglichkeiten bietet aber die explizite Verwedung eines Datenmodells. Ein Datenmodell regelt die Interaktion des JTree mit seinem Datenbaum und stellt Methoden zum Einfügen, Löschen und Ändern von Knoten sowie zum Auslesen des Pfades eines Knotens zur Verfügung.
Der JTree in der TreeTable von C Compact verwendet ein eigenes Datenmodell, TreeTableDataModel. Der Baum besteht aus Objekten der Klasse DataNode. Da ein eigenes Datenmodell verwendet wird, muss DataNode das Interface TreeNode nicht implemen-

[2] http://www.codejava.net/java-se/swing/jtree-basic-tutorial-and-examples
[3] http://docs.oracle.com/javase/tutorial/uiswing/components/tree.html
[4] http://www.java-tutorial.ch/core-java-tutorial/expande-or-collapse-all-nodes-in-a-jtree
[5] http://docs.oracle.com/javase/7/docs/api/javax/swing/tree/TreeNode.html
[6] https://docs.oracle.com/javase/7/docs/api/javax/swing/tree/DefaultMutableTreeNode.html

tieren.

JTable - Tabelle

Tabellen können in Swing mit dem Komponenten JTable[7] dargestellt werden. Wie auch bei der Klasse JTree werden die Daten nicht im Objekt von JTable gespeichert. Im einfachsten Fall kann eine Tabelle durch ein zweidimensionales Array initialisiert werden[8]. In der Regel wird aber ein Datenmodell wie etwa DefaultTableModel[9] verwendet. Ein für eine Tabelle verwendbares Datenmodell ist ein Objekt einer Klasse, die das Interface TableModel[10] implementiert.

2.4.3 Implementierung

Die grundlegende Implementierung erfolgte anhand eines Blogeintrages[11], mittlerweile wurde der Code allerdings stark an die Anforderungen der Benutzeroberfläche angepasst und erweitert.

Im folgenden Klassendiagramm wird der Aufbau der TreeTable dargestellt. Swing-Komponenten sind Dunkelblau, von Java vorgegebene Klassen und Interfaces sind Hellblau dargestellt. Die meisten Klassen sind selbst implementiert und daher mit weiß gekennzeichnet.

TreeTable.java

Diese Klasse ist die Hauptklasse der TreeTable und, da sie von JTable erbt, eine Swing-Komponente. Hier laufen alle Teile der TreeTable, wie etwa Datenmodelle und Renderer, zusammen.

VarTreeTable.java

VarTreeTable ist eine Wrapperklasse für TreeTable. Sie übernimmt alle wichtigen Funktionen zur Kommunikation mit dem Variablenbaum und bildet so eine Schicht zwischen dem Debugger und der TreeTable. Auf diese Weise wird der Code übersichtilcher und ist besser strukturiert. TreeTableView hat folgende Methoden:

- **init** Initialisiert die TreeTable mit einem Standard-Datenmodell. Dieses besteht nur aus einem Hauptknoten (root), der den Namen der aktuellen Datei trägt; es werden keine Variablen angezeigt. Das Standard-Datenmodell wird immer dann angezeigt, wenn der Debugger nicht aktiv ist, also wenn der Benutzer den Sourcecode editiert.
- **standby** Löscht das aktuelle Datenmodell und übergibt ein Standard-Datenmodell an die TreeTable, sodass wie zu Beginn nur der aktuelle Dateiname angezeigt wird. Diese Methode wird immer aufgerufen, wenn der Debugger beendet wird.

[7] http://docs.oracle.com/javase/tutorial/uiswing/components/table.html
[8] http://www.java2s.com/Tutorial/Java/0240__Swing/CreatingaJTable.htm
[9] http://docs.oracle.com/javase/8/docs/api/javax/swing/table/DefaultTableModel.html
[10] http://docs.oracle.com/javase/8/docs/api/javax/swing/table/TableModel.html
[11] http://www.hameister.org/JavaSwingTreeTable.html

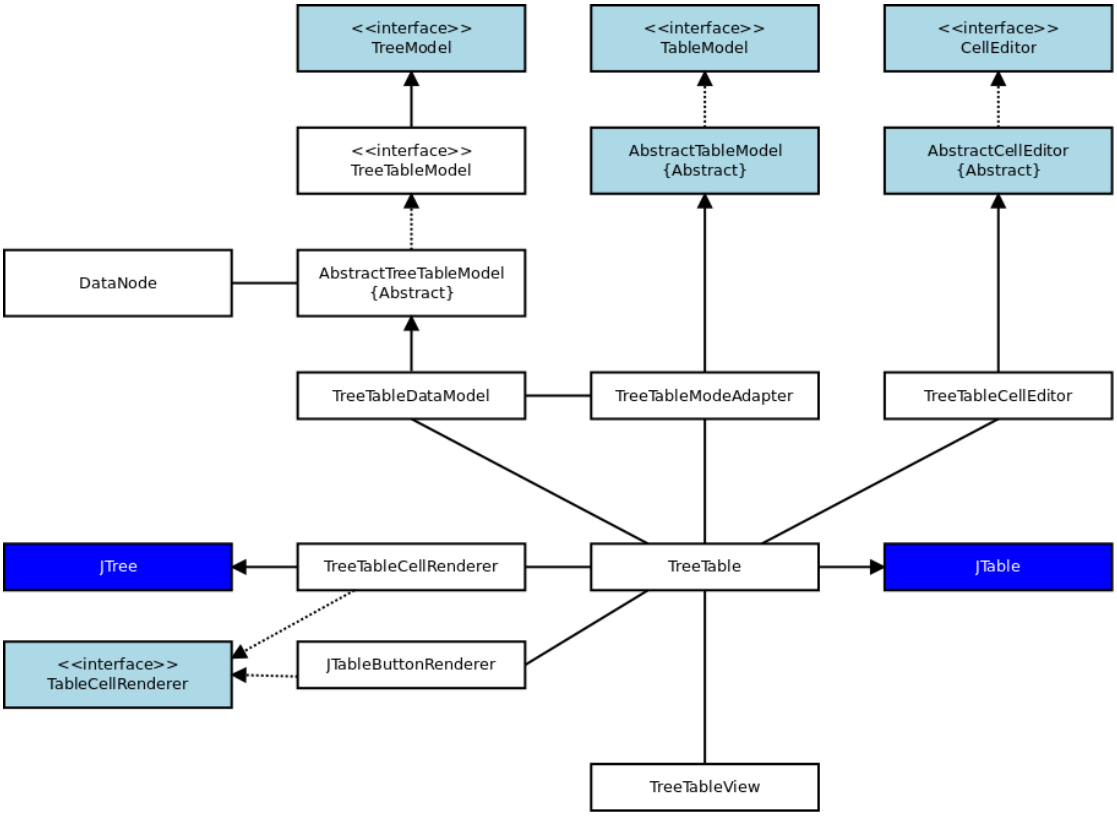


Abbildung 2.3: Klassendiagramm der TreeTable

- **update** Aktualisiert das Datenmodell der TreeTable im Debugmodus und zeigt die Variablen im Speicher des Interpreters an.
- **highlightVariable** Mit dieser Methode wird die Addresse einer zuletzt geänderten Variable an die TreeTable übergeben. Diese Variable wird in der Tabelle markiert. Es ist möglich, diese Funktion mehrmals aufzurufen und so mehrere Variablen zu markieren. Wenn das Datenmodell mit **update** aktualisiert wird, verfallen alle Addressen.
- **updateFontSize** Bei Aufruf dieser Methode wird die Schriftgröße der Tabelle geändert und die gesamte TreeTable neu gezeichnet. Diese Methode wird aufgerufen, wenn die Einstellungen der Benutzeroberfläche geändert wurden.

TreeModel.java

TreeModel ist ein Interface für alle Datenmodelle eines JTree.

TreeTableModel.java

Dieses Interface erweitert das Interface TreeModel so, dass es mit einer TreeTable kompatibel wird. TreeTableModel definiert zusätzliche Methoden, die für das Datenmodell einer Tabelle benötigt werden, beispielsweise ,,getColumnName()”.

AbstractTreeTableModel.java

Diese abstrakte Basisklasse implementiert einige konkrete Methoden für die Verwendung als Datenmodell eines JTree. In Objekten dieser Klasse werden der Hauptnoten (root) des Baumes gespeichert. Außderdem regelt AbstractTreeModel den Ablauf von Änderungsevents in der Datenstruktur des Baumes.

DataNode.java

Ein Datenknoten (DataNode) ist ein Knoten des in der TreeTable abgebildeten Baumes. In diesen Knoten sind alle nötigen Informationen zu den angezeigten Variablen - auch jene, die nicht im Baum selbst sondern in der Tabelle angezeigt werden - gespeichert. Die Klasse DataNode hat folgende Felder:

```
1 // Informationen, die in der Tabelle angezeigt werden
2 private final String name;
3 private Object type;
4 private Object value;
5
6 // Zusätzliche Informationen
7 private int address;
8 private int declaration;
```

Die TreeTable besitzt drei Spalten: In der ersten befindet sich der Baum mit den Namen der Variablen. in der zweiten Spalte wird der Datentyp der Variable und in der dritten ihr Wert gezeigt. Die Felder für Datentyp und Wert haben den Typ Object, da sich in der Tabelle an der Stelle eines Wertes auch ein JButton befinden kann.
Zusätzlich werden die Addresse und die Deklarationszeile der Variable gespeichert. Die Adresse dient der eindeutigen Identifikation der Variable, zum Beispiel beim Traversieren durch den Variablenbaum. Dadurch können lokale Variablen auch bei mehrfachem Aufruf einer Funktion unterschieden werden.
Bei Rechtsklick auf eine Variable in der TreeTable wird ein Kontextmenü mit der Option "Jump to declaration" geöffnet. Im Sourcecode wird dann die Zeile markiert, die im Feld "declaration" des betreffenden Datenknoten gespeichert ist.

TreeTableDataModel.java

TreeTableDataModel erbt von AbstractTreeTableModel beinhaltet einen weiteren Schritt zu einem Tabellen-Datenmodell. Beispielsweise sind Methoden implementiert, die im Datenmodell einer Tabelle benötigt werden. Außerdem wird in dieser Klasse das kontrete Aussehen der Tabelle festgelegt, indem die Tabellenspalten definiert sind:

```
1 // Column names
2 static protected String[] columnNames = {_("Name"), _("Type"),
     _("Value")};
3
4 // Column types
```

```
5 static protected Class<?>[] columnTypes = {TreeTableModel.class
    , String.class, Object.class};
```

TableModel.java

Diese Interface bildet die Grundlage für alle Datenmodelle einer JTable[12].

AbstractTableModel.java

In dieser abstrakten Basisklasse sind bereits die meisten Methoden des Interface TreeModel implementiert[13]. Der Benutzer muss lediglich die folgenden Methoden selbst definieren:

```
1 public int getRowCount();
2 public int getColumnCount();
3 public Object getValueAt(int row, int column);
```

Diese Methoden werden in der Klasse TreeTableModelAdapter implementiert.

TreeTableModelAdapter.java

TreeTableModelAdapter bildet die Verbindung zwischen dem Datenmodell der Tabelle und dem des Baumes. Diese Klasse erbt von AbstractTreeTableModel und enthält eine Referenz auf TreeTableDataModel. Viele Methoden, wie etwa ,,getCulumnName()" oder ,,getValueAt()" sind in TreeTableDataModel implementiert, da sich in dieser Klasse auch das eigentliche Datenmodell der TreeTable befindet, und werden in TreeTableModelAdapter übernommen. Beispielsweise greifen die folgenden Methoden auf die Implementierung in TreeTableDataModel zurück:

```
1 public Object getValueAt(int row, int column) {
2    return treeTableModel.getValueAt(nodeForRow(row), column);
3 }
4
5 public boolean isCellEditable(int row, int column) {
6    return treeTableModel.isCellEditable(nodeForRow(row), column
       );
7 }
```

CellEditor.java

Dieses Interface bildet die Grunglage für alle Klassen, die CellEditor einer JTable sind[14]. Ein CellEditor ist für die Dateneingabe in eine Tabelle verantwortlich[15]. Die einfach-

[12] http://docs.oracle.com/javase/7/docs/api/javax/swing/table/TableModel.html
[13] http://docs.oracle.com/javase/7/docs/api/javax/swing/table/AbstractTableModel.html
[14] https://docs.oracle.com/javase/7/docs/api/javax/swing/CellEditor.html
[15] http://docs.oracle.com/javase/tutorial/uiswing/components/table.html#editor

se Form eines CellEditors ist ein DefaultCellEditor[16]. Dieser unterstützt die Eingabe von Daten in die Tabelle mithilfe eines Textfeldes, einer Checkbox oder einer Combobox.

AbstractCellEditor.java

AbstractCellEditor ist eine abstrakte Basisklasse für alle Arten eines CellEditor in Swing[17]. Einige grundlegende Methoden sind bereits implementiert.

TreeTableCellEditor.java

Daten in der TreeTable sollen vom Benutzer zwar nicht verändert werden können, wenn das Editieren der Tabelle mit der Methode ,,setEditable()" aber deaktiviert wird, werden auch Events (wie etwa Mausklicks) nicht mehr registriert. TreeTableCellEditor gibt also Events in der Tabelle bei Bedarf an den JTree weiter und verbietet Events in Zellen, die nur Text enthalten.

TableCellRenderer.java

In diesem Interface sind alle Methoden definiert, die zum Rendern von Zellen einer JTable benötigt werden[18].

TreeTableCellRenderer.java

Diese Klasse erbt von JTree und wird verwendet, um einen Baum in die erste Spalte der Treetable zu rendern. Außerdem ist diese Klasse dafür verantwortlich, dass die Zeilen der Tabelle und des Baumes die selbe Höhe haben und dass die Zellen die richtige Farbe erhalten.

TableButtonRenderer.java

Diese Klasse wird als Renderer für die zweite und dritte Spalte der TreeTable verwendet. Sie erlaubt das Verwenden von Swing-Komponenten wie etwa JButtons als Zellendaten und sorgt dafür, dass die Zellen die korrekte Hintergrundfarbe haben.

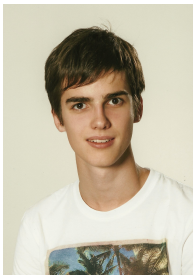
[16] https://docs.oracle.com/javase/7/docs/api/javax/swing/DefaultCellEditor.html
[17] https://docs.oracle.com/javase/7/docs/api/javax/swing/AbstractCellEditor.html
[18] https://docs.oracle.com/javase/7/docs/api/javax/swing/table/TableCellRenderer.html

Kapitel 3

Autoren

Fabian Hummer

Geburtstag, Geburtsort:	14.10.1995, Gmunden
Anschrift:	Im Gsperr 36 4810 Gmunden Österreich
E-Mail:	f.hummer@traunseenet.at



Abbildungsverzeichnis