

JAVA STARS NRW  
Sun Microsystems Award 2004

# Erstellung und Verwendung eines Modellrechners nach dem Von-Neumann-Prinzip

**Gymnasium Zitadelle der Stadt Jülich**  
Zitadelle, 52428 Jülich

Tel.: 02461/97860  
eMail: zitadelle.juelich@t-online.de



# Inhaltsverzeichnis

1	Projektbeschreibung .....	1
1.1	Unterrichtsfach.....	1
1.2	Thema .....	1
1.3	Projektidee und Ziele .....	1
1.4	Nutzen für den Unterricht .....	2
2	Lösungskonzept .....	3
2.1	Aufbau der Lösung .....	3
2.2	Eingesetzte Verfahren.....	3
3	Programm-Architektur.....	4
3.1	Übersicht nach BlueJ.....	4
3.2	Erläuterung in Kurzform(im README.TXT von BlueJ).....	4
3.3	Die Diagramme in UML-Form ohne Beziehungspfeile(nach Together).....	7
3.4	Dokumentation .....	8
4	Benutzerschnittstelle.....	9
4.1	Programmstart.....	9
4.2	Menüleiste .....	10
4.3	Registerfeld:.....	11
4.4	Der Speicher:.....	13
4.5	Der Editor .....	14

# 1 Projektbeschreibung

---

## 1.1 Unterrichtsfach

Grundkurs Informatik

## 1.2 Thema

Von Java zum Von-Neumann-Rechner

## 1.3 Projektidee und Ziele

Die Idee zur Programmierung eines Modellrechners stammt aus dem Unterricht bei der Behandlung des Themas „Struktur und Arbeitsweise von Computern“. Im Rahmen dieses Themas verwendeten wir den Modellrechner SOM, der in der Zeitschrift „Informatik Betrifft Uns“ im Bergmoser-Höllner Verlag in Aachen 1994 von A. Hermes veröffentlicht worden ist. Ein ähnlicher Rechner wurde zur gleichen Zeit vom Klettverlag angeboten. Der Modellrechner ist Stapel orientiert und verwendet einen Modellbefehlssatz. Er wurde in Pascal programmiert und der Interpreter ist in der angegebenen Literatur beschrieben.

Wir entschieden, den Modellrechner in Java so zu programmieren, dass er heutigen Benutzeroberflächen entspricht und den Befehlssatz so weit wie möglich beizubehalten.

Schnappschüsse von der Benutzeroberfläche des betreffenden Modellrechners zeigt das Kapitel 5 „Dokumentation des Projektverlaufs“.

Wir setzen uns die Ziele, mit dem von uns zu programmierenden Modellrechner, Schulen eine kostenfreie komfortable Version des Rechners anzubieten und ihn so zu gestalten, dass die Übersetzung von Zahlen betreffenden Javaprogrammen auf die Modellsprache und die Abbildung von Objekten im Modellspeicher transparent werden.

Die folgenden Gesichtspunkte dienen als Leitlinie

- Der Stapel orientierte Modellrechner (SOM) präsentiert Prinzipien der Speicherabbildung von Programmen höherer Programmiersprachen auf maschinenaher Ebene.
- Der Modellassembler ist symbolisch (Mnemonics). Die Befehle und Daten werden dezimal verschlüsselt. Die binäre Verschlüsselung ist eine einfache Rechnung.
- SOM veranschaulicht die Verarbeitung von Bedingungen, Schleifen und Rekursion im Modellspeicher.
- Der Modellrechner liefert eine Abbildung von Objekten und Strukturen, die auf ganzen Zahlen basieren.
- Bei den Beispielen soll der Zugriff auf globale Variablen nur über Methoden möglich sein, so dass einfache Adressierung genügt. Paaradressierung wird nicht unterstützt. Die Assemblersprache und das Modell orientieren sich am Lehrbuch „Hermes/Stobbe, Informatik Zwei, Ernst Klett Schulbuchverlag, 1995“.
- Modellbeispiele können geladen werden. Die zugehörigen Javaprogramme befinden sich im Order "Java".
- Der Ausdruck von Programmen wird unterstützt. Die Programmzeilen werden von 0 an durchnummeriert.
- Programme können erstellt, übersetzt und ausgeführt werden

- Bemerkungen werden beim Programmablauf angezeigt
- Hilfemenüs stehen zur Verfügung
- Schneller und schrittweiser Durchlauf des Von-Neumann-Zyklus sind möglich.

#### 1.4 *Nutzen für den Unterricht*

- Verständnis der Struktur und Arbeitsweise eines Von-Neumann-Rechners
- Präsentation der Abbildung von höheren Strukturen auf Assembler-Ebene (Übersetzungsstrategien)
- Maschinenorientierte Programmierung (teilsymbolischer Assembler, Sprungbefehle, Unterprogramme, Objekte, Laufzeitkellerverwaltung, Heapverwaltung)
- Schnittstellenbestimmung und Aufteilung in Teilprojekte
- Vertiefung in Java (Drucken, Dateibehandlung, Layout, Swingklassen, Menüs)
- Für Lehrer und Schüler: Angebot eines Programms zur Demonstration der Struktur und Arbeitsweise eines Von-Neumann-Rechners. Das Programm kann im Unterricht zum gleichnamigen Thema eingesetzt werden.
- Es ist zu unterscheiden zwischen dem Einsatz des Modellrechners und seiner Konstruktion. Beides sind mögliche Unterrichtsthemen.  
Die Modellrechnerentwicklung setzt gute Kenntnisse in Java voraus und kann als Projekt bearbeitet werden.  
Der Einsatz des Modellrechners hingegen deckt den Bereich „Struktur und Arbeitsweise eines Rechners“ ab. Hier steht die Übersetzung einfacher Javaprogramme in Maschinen orientierte Anweisungen und deren Abarbeitung von der Maschine im Mittelpunkt.

## 2 Lösungskonzept

---

### 2.1 Aufbau der Lösung

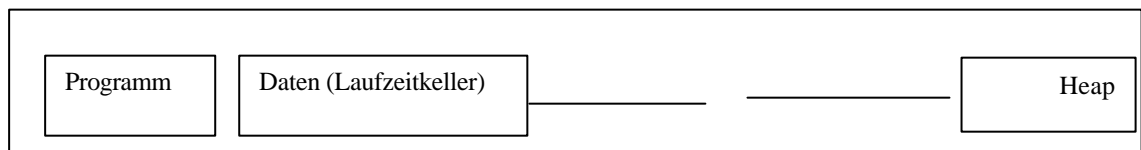
Zu den Komponenten des Von-Neumann-Rechners existieren analoge Javaklassen mit entsprechenden Abhängigkeiten und Kommunikationsmöglichkeiten. Es existiert eine Applikation, die Laden, Speichern und Dateiauswahl zulässt, als auch ein Applet zur Verwendung des Programms im Internet. Exemplarische Programme können auch im Applet ausgewählt werden. Zum Speichern dieser oder neuer Programme sind die Editorfunktionen des „Clients“ anzuwenden.

Das Programmpaket trennt die Kontrolle möglichst vom Modell und den jeweiligen Sichten. Die Benutzerführung ist durch Menüs gesteuert.

### 2.2 Eingesetzte Verfahren

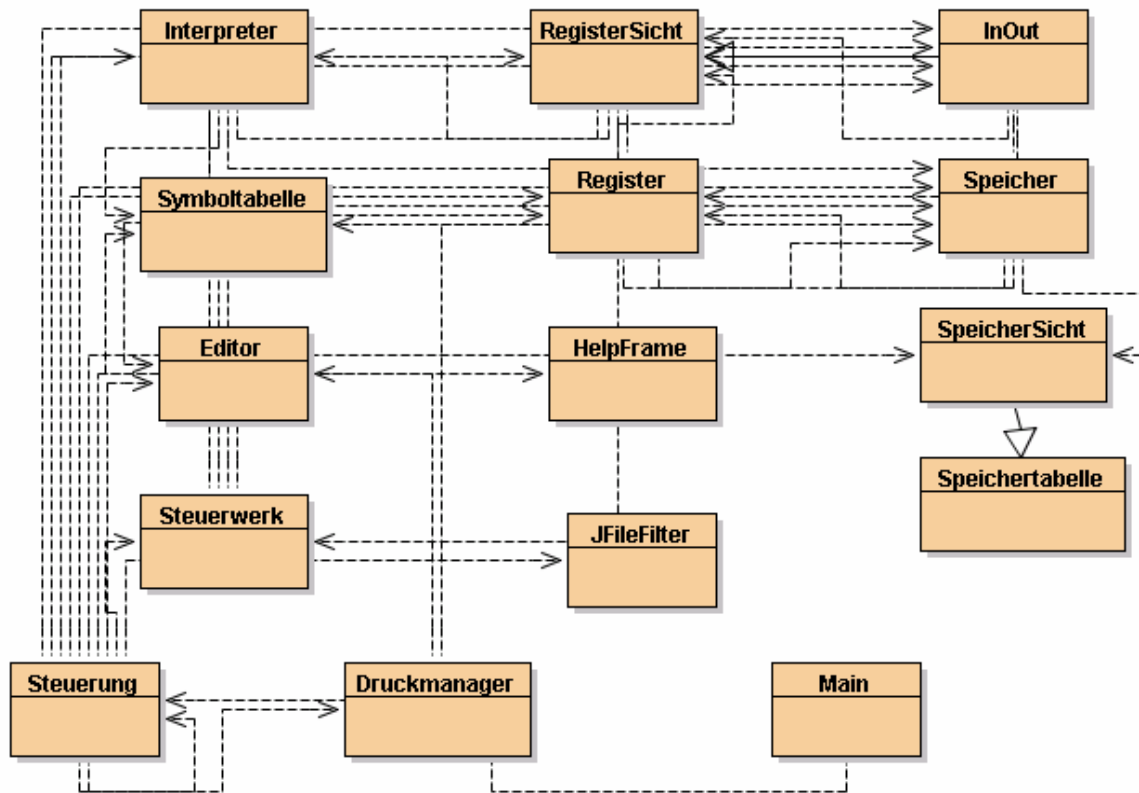
Die Javaklassen bilden die Grundlage für die Aufteilung der Gruppenarbeit. Die Vorgehensweise ist Objekt orientiert. Eingesetzte Datenstrukturen sind Reihungen, Listen (Vector), Hash-Tabellen (Hash-Map), Stapel und Dateien (File). Es werden die Standardlayoutverfahren von Java genutzt (Borderlayout, Gridbaglayout).

Der Speicher des Rechners gliedert sich in den Programmteil, den als Laufzeitkeller konstruierten Datenteil und dem Heap, dem der Datenteil entgegen wächst. Die Speicherverwaltung des Heaps ist nach dem First-Fit-Verfahren organisiert. Die Verschlüsselung erfolgt nicht wie im Von-Neumann-Rechner binär, sondern dezimal.



# 3 Programm-Architektur

## 3.1 Übersicht nach BlueJ



## 3.2 Erläuterung in Kurzform(im README.TXT von BlueJ)

PROJECT TITLE: stack oriented mashine

PURPOSE OF PROJECT: demonstrating the Von-Neumann-Principle and learning principle of compileres und assemblers

VERSION or DATE: 8.7.2004

HOW TO START THIS PROJECT: new Main()

AUTHORS: Gymnasium Zitadelle Juelich

USER INSTRUCTIONS: Start mit Main.

Man wähle aus dem Menü ein Programm und führe es schnell (run) oder schrittweise (step) aus.

Die Beschreibung der Klassen in Kurzform

### Main

Enthält das Startprogramm

### Steuerung

Die Benutzeroberfläche (Laden, Speichern, Wechsel zwischen Editoren, Drucken, Übersetzen, Ausführen, Hilfedateien, Edieren)

### **Symboltabelle**

Zuordnung von Symbole der Modellsprache zu Zahlen und umgekehrt

```
public static String getKey(String mnemo)
public static String getMnemo(String key)
public static String getDescription(String mnemo)
```

### **Steuerwerk** (Führt ein uebersetztes Programm aus)

Zyklus: Befehl lesen, interpretieren, ausführen

### **Speicher**

Datenspeicher, Programmspeicher, Heap, dezimale Codierung (statt binärer)  
Verknüpfungen mit Register, Speichersicht, Speichertabelle

### **Register**

Spezielle Speicherzellen  
Verknüpfung mit RegisterSicht

### **RegisterSicht**

Darstellung des Speichers, Oberklasse zu InOut

### **InOut**

Ein/Ausgabe-Einheit des Rechners (Unterklasse von RegisterSicht)

### **Interpreter** (Übersetzt ein Assemblerprogramm in Dezimalcode)

```
public Interpreter(Speicher speicher)
public void interpretiere()
```

### **Editor**

Ein Editor zur Eingabe oder Veränderung von Assemblerprogrammen

### **HelpFrame**

Ein Fenster (JFrame) zur Ausgabe von Hilfedateien

### **Druckmanager**

Sorgt für den Ausdruck von Programmen. Die Programmzeilen werden von 0 an nummeriert.

### **JFileFilter**

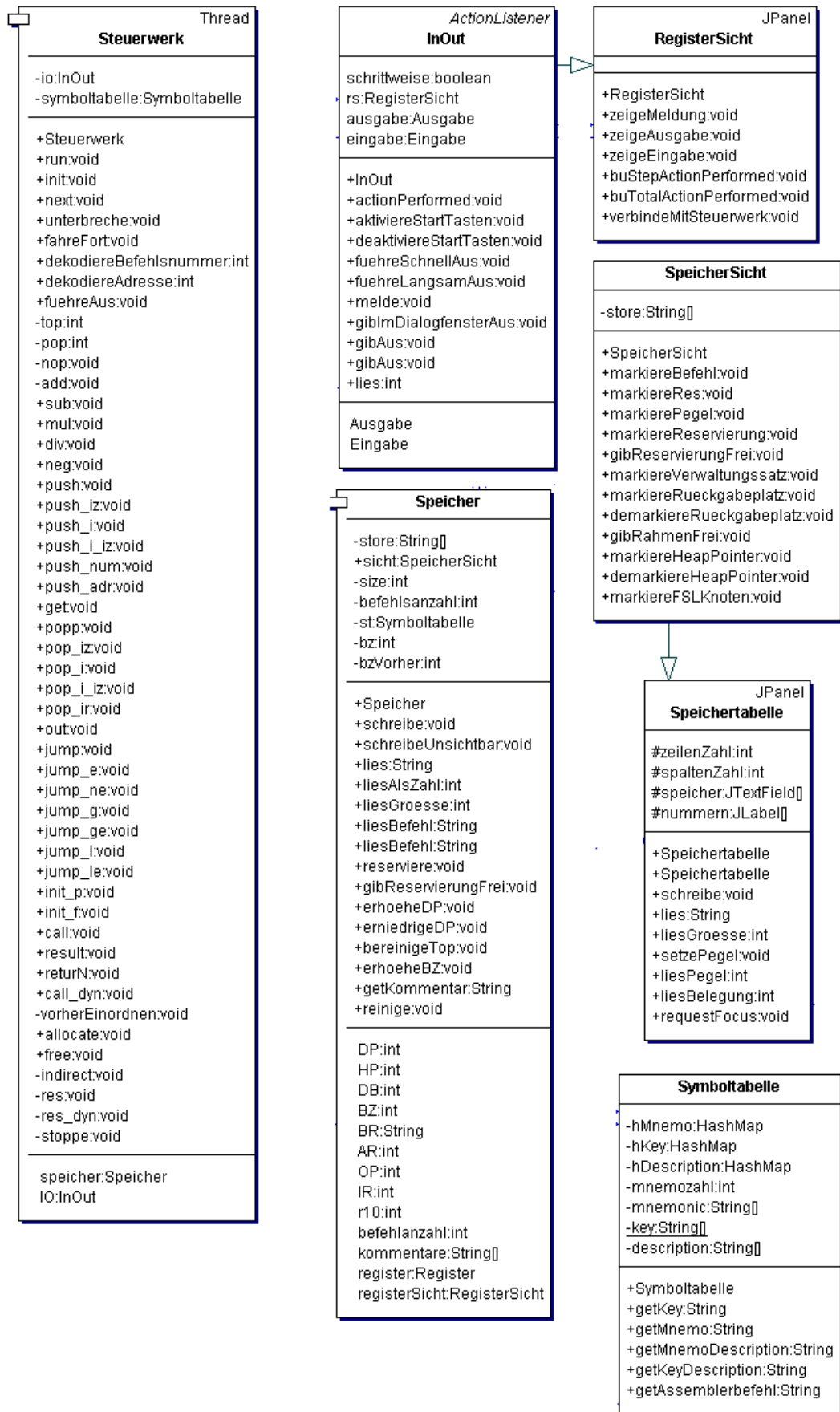
Dient zur Auswahl von Dateien

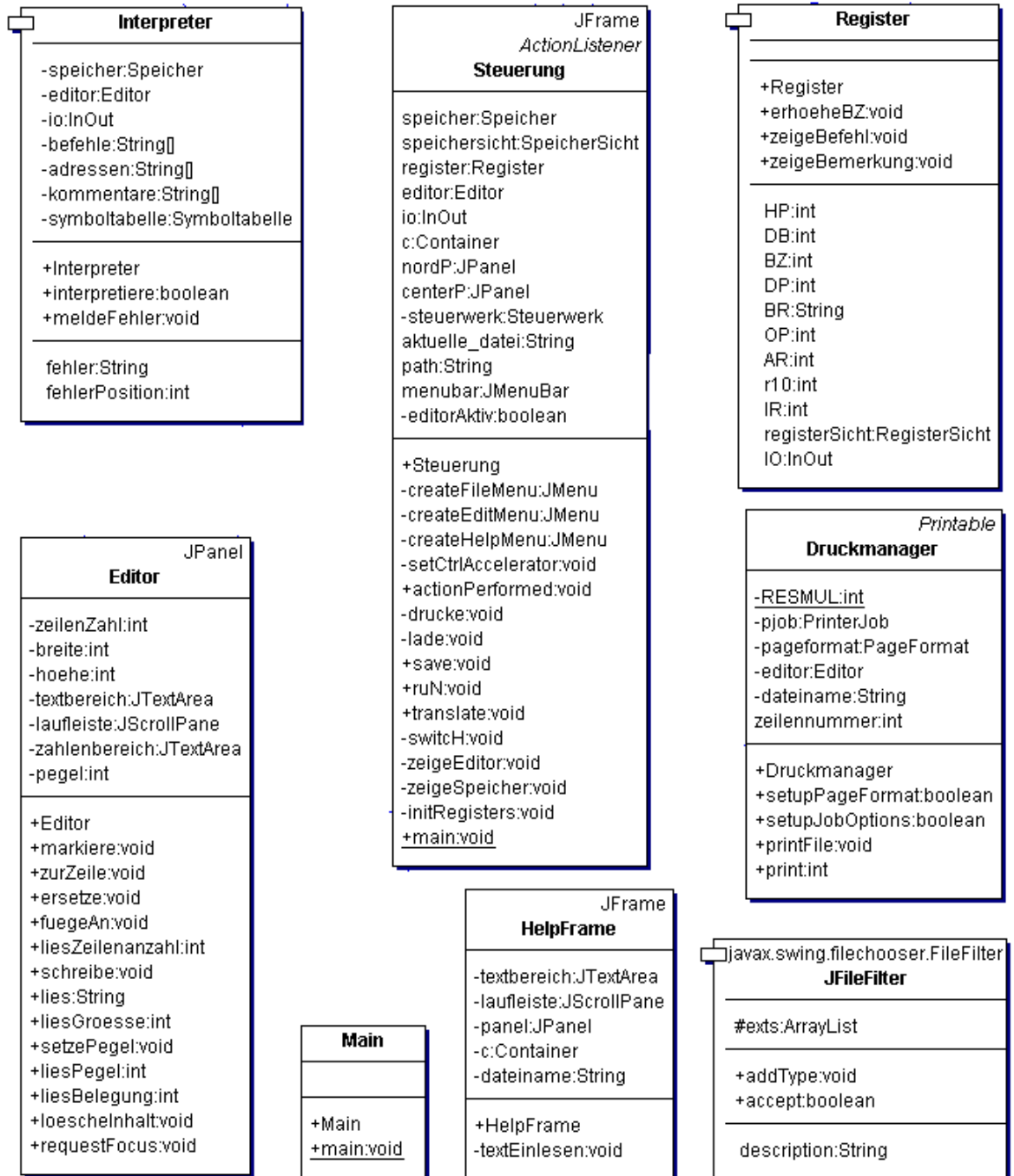
### **SteuerungApplet**

- Die Steuerung als Applet. Sie wird von "index.html" aufgerufen.
- Im Gegensatz zur Applikation sind Druck- und Speicherfunktionen nicht integriert.
- Es können vorgegebene Musterbeispiele aus einer Resourcedatei gewählt und geladen werden. Die Datei hat den Namen "mol.zip".
- Zum Speichern von erstellten Programmen können "cut & past"-Funktionen des Rechners genutzt werden.



### 3.3 Die Diagramme in UML-Form ohne Beziehungspfeile(nach Together)





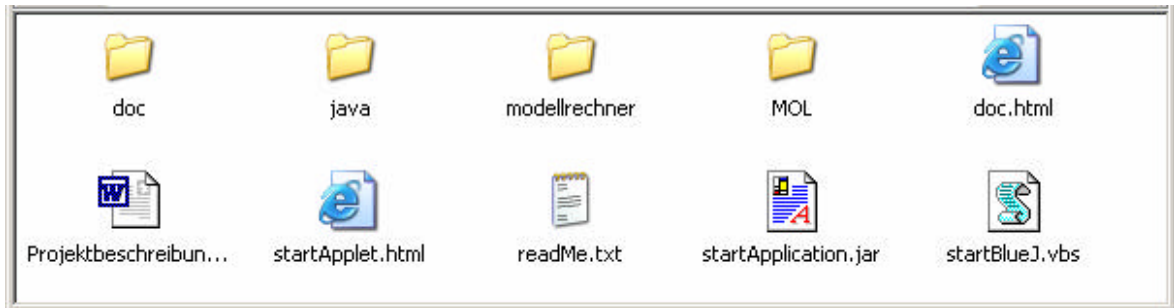
### 3.4 Dokumentation

Die Programmdokumentation liegt als HTML-Dokument aus der CD vor. Sie ist mit javadoc erstellt worden. Sie enthält zusätzlich die in 3.1 bis 3.3 dargestellten Informationen.

(Von der Überschrift führt ein Verweis (hyperlink) auf die Dokumentation, auf der CD: doc.html bzw. doc/index.html)

## 4 Benutzerschnittstelle

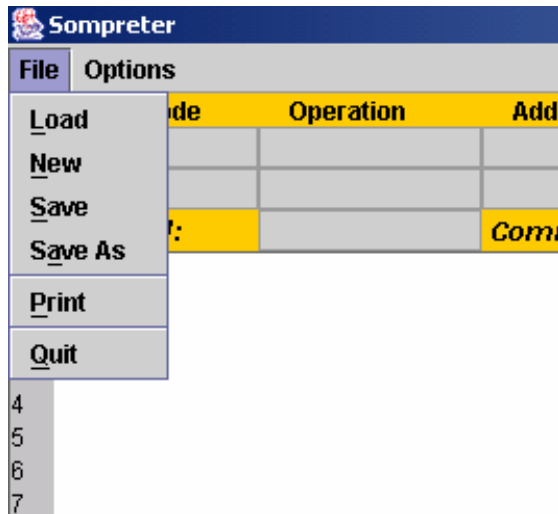
### 4.1 Programmstart



startApplet.html	Start des Modellrechners als Applet. Das Programm ruft im Ordner „Modellrechner“ die Datei „index.html“ auf.
startApplication.jar	Start der Modellrechners als Applikation Das Programm aktiviert im Ordner „Modellrechner“ die Jar-Datei „somApplication.jar“. Das Hauptprogramm lautet „Main.java“. Eine Öffnung von „Main.java“ ist auch mit BlueJ möglich: Doppelklick auf „blueJ.pkg“
java	Der Ordner enthält die Quelltexte der Java-Programme, die als Vorlage zur Übersetzung in die Modellsprache MOL dienen
MOL	Der Ordner „MOL“ enthält die Musterprogramme der Modellsprache „MOL“ (maschine oriented language)
modellrechner	Der Ordner „Modellrechner“ enthält sämtliche Quelltexte der Java-Programme, die den Modellrechner erzeugen. Startprogramme: Main.class, somApplication.jar, index.html
doc	Der Ordner enthält die von javadoc erzeugte Dokumentation mit zusätzlichen Veranschaulichungen. Startdatei: index.html
doc.html	Das Programm startet die Dokumentation im Verzeichnis doc
Projektbeschreibung.doc	Diese Beschreibung als Word-Dokument
Readme.txt	Kurze Starthinweise

## 4.2 Menüleiste

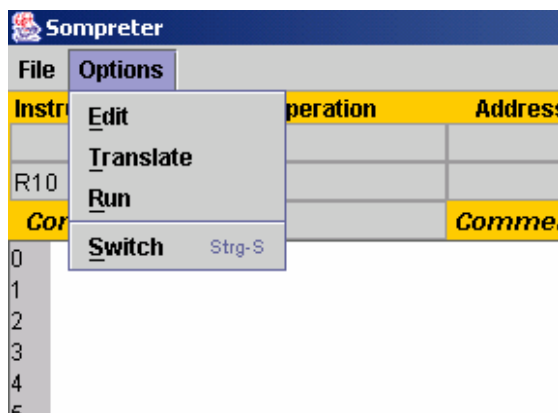
### File:



- Load: Öffnet Menü gesteuert eine bereits existierende MOL-Datei.
- New: Öffnet einen leeren Texteditor zum Schreiben eigener Programme:
  - je eine Anweisung pro Zeile
  - keine Leerzeilen
  - Kommentare rechts neben die Anweisung durch mindesten zwei Leerzeichen getrennt.
- Save: Speichert die Änderungen einer bereits vorhandenen Datei.
- Save As: Ruft ein Menü zur Speicherung der Datei auf.
- Print: Öffnet einen Druckdialog
- Quit: Beendet das Programm

In der HTML-Version fehlen die Menüpunkte zum Speichern und Drucken. Zum Laden besteht ein Angebot aus einer Resource.

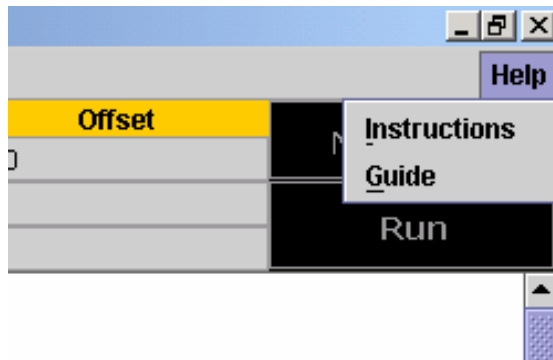
### Options:



- Edit: Öffnet den Texteditor. Hier können Sie ihren Code als Mnemonic sehen und ändern.

- Translate: Übersetzt den Mnemonic -Code in Dezimalcode und öffnet die Speicheransicht. Das Programm kann schrittweise ausgeführt werden. Der Benutzer kann Zustandsveränderungen des Speichers verfolgen und gegebenenfalls Eingaben machen.
- Run: Hiermit wird das komplette Programm in der Speichersicht einmal ausgeführt.
- Switch: Wechsel zwischen der Mnemonic - und der Dezimaldarstellung.

### Help:



- Instructions: Hier finden Sie die zur Verfügung stehenden Modell-Assembler-Befehle (siehe Anhang).
- Guide: Erläutert die Bedienung in Kurzform

### 4.3 Registerfeld:

Sompreter								Help
File Options								Help
Instructioncode	Operation	Address	Stacksegment	Stackpointer	Instructionpointer	Heappointer	Offset	Next Step
			0	0	0		0	
R10				In:		Out:		Run
Command:		Comment:						
0								
1								
2								

- Instructioncode (Befehlsregister): Hier steht der Dezimalcode des Befehls.
- Operation (Operationsregister): Hier steht der Befehlsteil des Codes.
- Adress (Adressregister): Hier steht die Adresse, die der Befehl benutzt.
- Stacksegment (Datenbasis): Hier steht der Beginn des aktuellen Datenraumes.
- Stackpointer (Datenpegel): Hier steht die Stelle des letzten Elementes des Stapels.
- Instructionpointer (Befehlszähler): Hier steht, welcher Befehl als nächstes ausgeführt wird.
- Heappointer : Hier steht der Anfang des freien Speichers am Ende der Liste.
- Offset (Indexregister): Hier steht ein Wert, der zur Adresse hinzugefügt werden muss.

- R10 : Hilfsregister. Es wird bei Methodenaufrufen benutzt
- In : Hier steht die vorangegangene Eingabe.
- Out : Hier findet man die Ausgabe.
- Command : Das Feld zeigt den aktuellen Assemblerbefehl (Mnemonic).
- Comment : Hier steht eine vorprogrammierte Erklärung des Befehls, es sei denn, man hat im Quellcode selbst einen Kommentar eingegeben. Der Kommentar steht in derselben Zeile wie der Befehl, durch mindestens zwei Leerzeichen getrennt.
- Next Step : Hiermit wird der nächste Befehl ausgeführt.
- Run: Hiermit wird das komplette Programm einmal ausgeführt.

### **Die Programmerstellung und Ausführung in Kurzform**

1. Öffnen Sie eine neue Datei (File->New).
2. Schreiben Sie ihren Code mithilfe der vorprogrammierten Befehle, die Sie bei Help->Instructions finden.
3. Sie können den Code kommentieren, indem Sie in derselben Zeile (hinter ihrem Befehl) ihren Kommentar schreiben. Dieser muss durch mindestens 2 Leerzeichen vom Befehl getrennt sein.
4. Wenn Sie ihren Code übersetzen und bei Fehlerfreiheit in Dezimaldarstellung sehen möchten, klicken Sie auf Options->Translate.
5. Zur Ausführung ihres Programms haben Sie 2 Möglichkeiten:
  - a) Sie führen das Programm in einem Durchgang aus.  
(entweder Options->Run oder "Run"-Button)
  - b) Sie führen ihr Programm Schritt für Schritt durch.  
("Next Step"-Button)
6. Wenn eine Eingabe erforderlich ist, erscheint ein Eingabefenster. Die Eingabe muss mit der Enter-Taste bzw. mit dem "Ok"-Button bestätigt werden.
7. Beim Ende des Programms erscheint eine entsprechende Meldung.

#### 4.4 Der Speicher:

Die Abbildung zeigt ein Speicherausschnitt während eines Programmablaufs. Der aktuelle Befehl liegt in Zeile 2 vor, während der Befehlszeiger (instructionpointer) bereits nach zum nächsten Befehl zeigt. 27000000 ist der Code des Befehls get. Der im Programm enthaltene Kommentar „fakultaet von?“ wird angezeigt.

C:\Dokumente und Einstellungen\hermes2\Desktop\Wettbewerb\MOL\FAKASS.MOL

File		Options					
Instructioncode	Operation	Address	Stacksegment	Stackpointer	Instructionpointer		
27000000	27	0	22	26	3		
27				In:			
<b>Command:</b>	GET 0	<b>Comment:</b>	fakultaet von?				
90000001	0	0	52	78	104		
51000000	1	27	53	79	105		
27000000	2	28	54	80	106		
52000008	3	29	55	81	107		
30000000	4	30	56	82	108		
20000000	5	31	57	83	109		
37000000	6	32	58	84	110		
99000000	7	33	59	85	111		
20000000	8			86	112		
43000013	9			87	113		
24000001	10			88	114		
53000000	11			89	115		
40000021	12			90	116		
20000000	13			91	117		
51000000	14			92	118		
20000000	15			93	119		
24000001	16			94	120		
11000000	17			95	121		
52000008	18			96	122		
12000000	19			97	123		
53000000	20			98	124		
54000000	21			99	125		
	22			100	126		
	F			101	127		
	SS			102	128		

fakultaet von?	
<input type="text" value="5"/>	
<input type="button" value="OK"/>	<input type="button" value="Abbrechen"/>

## 4.5 Der Editor

Das folgende Bild zeigt das zum obigen Beispiel passende Assemblerprogramm im Editor.

Der Befehl in der Zeile mit der Nummer 2 lautet „get“. Rechts befindet sich der Kommentar, der vom Rechner als solcher behandelt und ausgegeben wird. Die Registerinhalte sind bezüglich des Editors ohne Belang. Sie sind Relikte eines vorangehenden Programmablaufs.

H:\Unterricht\Informatik\Wettbewerb\MOL\FAKASS.MOL					
File Options					
Instructioncode	Operation	Address	Stacksegment	Stackpointer	
90000001	90	1	22	21	1
51				<b>In:</b>	€
<b>Command:</b>	RES 1	<b>Comment:</b>	int a; a --> 0; 1 Platz		
0	res 1	int a; a --> 0; 1 Platz			
1	init_f	Initialisieren: Funktionsaufruf			
2	get	fakultaet von ?			
3	call F	Aufruf von fakultaet			
4	pop 0	a = fakultaet(4)			
5	push 0	a --> Laufzeitkeller			
6	out	Ausgabe von a			
7	stop	End.			
8	F: push 0	Beginn der Methode (Funktion) fakultaet(n); n --> 0			
9	jump_g M1	if ( n > 0 ) { Sprung nach M1; }			
10	push_num 1	fakultaet := 1			
11	result	Ergebniszuteilung			
12	jump M2	Sprung nach M2			
13	M1: push 0	return n * fakultaet(n-1);			
14	init_f	Init: Funktionsaufruf			
15	push 0	n-1 --> Stack			
16	push_num 1				
17	sub				
18	call F	Aufruf von fakultaet in F			
19	mul	multipliziere			
20	result	Funktionsergebnissicherung			
21	M2: return	Ruecksprung			
22					