von Ulrich Grude

Inhaltsverzeichnis

Der Editor TextPad

Der TextPad ist ein ziemlich ausgereifter Text-Editor, mit dem man z.B. Java-Quelldateien editieren kann (siehe auch http://www.chip.de/downloads/TextPad_12992965.html#). Er läuft (leider) nur unter Windows, nicht unter Linux. Die neuste Version kann man von der Adresse http://www.textpad.com herunterladen (ca. 3 MB). Es gibt Versionen auf Englisch, Deutsch, Französisch, Japanisch, ... Eine Einzellizenz kostet 32 US$ (etwa 25 €). Solange man keine Lizenz-Nummer eingegeben hat, wird man regelmäßig zum Kauf aufgefordert, kann den TextPad aber trotzdem ohne Einschränkungen benutzen.

Man kann den TextPad erstmal "einfach so" und ganz einfach benutzen. Er bietet einem aber auch einige kompliziertere und mächtige Befehle an, z.B. Suchen-und-Ersetzen mit Hilfe von regulären Ausdrücken, Suchen-in-mehreren-Dateien, Editieren im Blockmodus, Textdateien zwischen dem Windows-Format und dem Unix/Linux-Format hin- und her umwandeln, Dateien hexadezimal anzeigen etc. etc. Außerdem kann man ihn auf verschiedene Weisen einstellen und damit den eigenen Bedürfnissen anpassen indem man z.B. Tastenkürzel, Makros, Texbausteine, neue Dokumentenklassen oder Befehle im Menü Extras (engl. Tools) definiert.

Im SWE-Labor der BHT steht den Benutzern eine auf bestimmte Weise vorkonfigurierte Version des TextPad zur Verfügung. Es ist nicht ganz trivial, aber durchaus machbar, auf einem eigenen Rechner den TextPad entsprechend zu konfigurieren. Die folgenden Tipps sollen Ihnen dabei helfen, den TextPad im SWE-Labor (zum Erstellen und Bearbeiten von Java-Programmen) zu benutzen und auf Ihrem eigenen

Rechner zu konfigurieren. Wahrscheinlich kann man nicht alle diese Tipps in der ersten Woche des Semesters lernen, aber wenn Sie *jede Woche einen Tipp* durcharbeiten sind Sie am Ende des Semesters wahrscheinlich kaum noch von einem Profi zu unterscheiden :-).

1. Darstellung der Tipps

Darstellung (Beispiel)	Bedeutung
Menü Konfiguration	Sie sollen das Menü Konfiguration wählen (mit einem Links-Klick)
Menü Konfiguration, Einstellungen...	Im Menü Konfiguration den Menüpunkt Einstellungen... wählen (mit einem Links-Klick)
Dokumentenklasse	Links-Klick auf das Wort Dokumentenklasse
+Dokumentenklasse	Links-Klick auf das *Pluszeichen* + vor dem Wort Dokumentenklasse

2. Das aktive Dokument

Mit dem TextPad kann man viele Dokumente öffnen und quasi gleichzeitig bearbeiten. In jedem Moment ist genau eines dieser Dokumente *das aktive Dokument* und das Fenster, in dem es angezeigt wird, ist *das aktive Fenster*. Vom aktiven Fenster sagt man auch, dass es *den Fokus hat*. Ein Fenster bekommt den Fokus unter anderem dann, wenn man irgendwo in das Fenster klickt oder auf seinen Reiter (engl. tab) oder auf seinen Namen in der Dateiliste klickt (Die Dateiliste ist ein schmales Fenster am linken Rand des TextPad-Fensters).

Viele Befehle des TextPad beziehen sich immer auf *das aktive Dokument* (z.B. die Befehle Java kompilieren und Java-Programm starten im Menü Extras).

3. Die Java-Entwicklungsumgebung von Sun und den TextPad installieren

Laden Sie von der Adresse
http://www.oracle.com/technetwork/java/javase/downloads/java-se-jdk-7-download-432154.html
die Datei Java Development Kit (`jdk-7-windows-i586.exe` oder so ähnlich) herunter und installieren Sie die Datei. Die Datei Java Runtime Environment (JRE) reicht *nicht* aus (ist aber in der jdk-Datei enthalten)!

Die Abkürzungen SE, EE und ME nach dem Wort Java stehen für Standard Edition, Enterprise Edition bzw. Micro Edition. Zum *Lernen* von Java ist die Java SE besonders empfehlenswert.

Laden Sie den TextPad (von der Adresse http://www.textpad.com) herunter und installieren Sie ihn. Die Tipps in der vorliegenden Datei beziehen sich auf die *deutsche Version* des TextPad!

Wenn Sie erst den JDK und dann den TextPad installieren, finden Sie im TextPad-Menü Extras, Benutzer-Programme drei Befehle (oder: Menüpunkte) namens Java kompilieren, Java-Programm starten und Java-Applet starten.

Falls diese drei Menüpunkte noch *nicht* vorhanden sind, können Sie sie wie folgt einrichten:

Menü Konfiguration, Einstellungen, Extras

Im Fenster Einstellungen (rechts oben) auf den Knopf Hinzufügen klicken und Java SDK Befehle wählen.

4. Der Umgebungsvariablen CLASSPATH einen Wert zuweisen

Wenn man eine (fehlerfreie) `.java`-Datei (z.B. `Hallo01.java`) compiliert, erzeugt der Compiler daraus eine entsprechende `.class`-Datei (im Beispiel: `Hallo01.class`). Es wird empfohlen, diese `.class`-Dateien *getrennt* von den `.java`-Dateien abzulegen. Im SWE-Labor ist dafür (in Ihrem Heimatordner `Z:`) der Ordner `Z:\Klassen` bereits angelegt. Auf Ihrem eigenen Rechner sollten Sie irgendwo einen entsprechenden Ordner anlegen, z.B. den Ordner `C:\meineDateien\Klassen`.

Wenn der Java-Ausführer eine bestimmte `.class`-Datei oder `.java`-Datei braucht, sucht er sie (normalerweise) in allen Ordnern und `.jar`-Dateien, die in der Umgebungsvariablen namens `CLASSPATH` eingetragen sind (durch Semikolons `;` voneinander getrennt).

4.1. Uumgebungsvariablen unter Windows XP

Unter Windows XP können Sie der Umgebungsvariablen `CLASSPATH` wie folgt einen geeigneten Wert zuweisen:

Linksklick auf Start, Systemsteuerung, System, Erweitert, Umgebungsvariablen

Im Fenster Umgebungsvariablen können Sie Umgebungsvariablen wahlweise nur für den aktuellen *Benutzer* oder als *Systemvariablen* einrichten (die für alle Benutzer gelten). Richten Sie im Zweifelsfall eine Systemvariable ein.

Suchen Sie zuerst in der Spalte Variable, ob es schon eine Variable namens `CLASSPATH` gibt. Falls ja, wählen Sie sie mit der Maus und klicken Sie auf Bearbeiten. Falls es noch keine solche Variable gibt, klicken Sie auf Neu. Ein Fenster namens Systemvariable bearbeiten (bzw. Neue Systemvariable, oder Benutzervariable bearbeiten bzw. Neue Benutzervariable) sollte aufgehen.

Ein typischer Inhalt für die Variable `CLASSPATH` sieht etwa so aus:

```
C:\meineDateien\Klassen\;D:\classes;.
```

Das sind die Namen von drei Ordnern, die durch je ein Semikolon `;` voneinander getrennt sind. Der besonders kurze Name `.` (im Beispiel nach dem zweiten Semikolon) bezeichnet immer den aktuellen Arbeitsordner. Zuerst sollten Sie den Ordner angeben, in dem alle `.class`-Dateien abgelegt werden sollen, und danach den aktuellen Arbeitsordner.

Wenn der Inhalt der `CLASSPATH`-Variablen lang und kompliziert ist, sollte man ihn *nicht* im (im deutlich zu kleinen) Fenster Systemvariable bearbeiten editieren, sondern mit dem TextPad in einem Text-Fenster. Erst nachdem man den Inhalt geschrieben und geprüft hat, sollte man ihn "im Ganzen" in das Fenster Systemvariable bearbeiten kopieren (mit STRG-C und STRG-V).

4.2. Uumgebungsvariablen unter Windows 7

Unter Windows 7 können Sie der Umgebungsvariablen `CLASSPATH` wie folgt einen geeigneten Wert zuweisen:

Linksklick auf den Start-Ballon, Systemsteuerung, System, (ganz links) Erweiterte Systemeinstellungen, (unten) Umgebungsvariablen.

Dann wie bei Windows XP (siehe vorigen Abschnitt).

5. Das Framework JUnit auf Ihrem Rechner installieren

Das Framework JUnit unterstützt einen dabei, in der Sprache Java Testprogramme für Java-Programme zu schreiben. Zur Zeit (Ende 2011) gibt es von JUnit *zwei Hauptversionen*, die ältere Version 3 (und davon die Unterversion 3.8) und eine neuere Version 4 (und davon Unterversionen wie z.B. 4.3). Die beiden Hauptversionen sind nicht kompatibel miteinander: Ein Testprogramm, welches für JUnit Version 3 geschrieben wurde, läuft nicht mit JUnit Version 4 und umgekehrt.

JUnit Version 3 (Unterversion 3.8) wird ausgeliefert in Form einer Datei namens `junit.jar`.
Um JUnit zu benutzen, sollten Sie dies Datei in irgendeinen Ordner kopieren (z.B. so: `C:\MeineJarDateien\junit.jar`) und dann den Pfadnamen der Datei in die Umgebungsvariable `CLASSPATH` eintragen. Wie man das macht, wird im vorigen Tipp beschrieben.

Achtung: Wenn in der Variablen `CLASSPATH` schon etwas drinsteht, sollten Sie es *nicht löschen*, sondern nur *ergänzen*. Schreiben Sie dazu hinter den bisherigen Inhalt (als Trennzeichen) ein Semikolon `;` und dahinter den zusätzlichen Pfadnamen (z.B. `C:\MeineJarDateien\junit.jar`).

6. Ein kleines Programm zusammenklicken statt es einzutippen

Der TextPad im SWE-Labor ist so konfiguriert, dass man ein einfaches Java-Hallo-Programm in etwa 30 sec erstellen, kompilieren und ausführen lassen kann. Auch beim Erstellen größerer Programme lohnt es sich häufig, mit so einem "zusammengeklickten" Programm zu beginnen, etwa so:

1. Öffnen Sie im TextPad ein neues Dokument (z.B. indem Sie STRG-N eingeben).

2. Machen Sie das Textbausteine-Fenster sichtbar, indem Sie ALT-0 eingeben. Dieses Fenster ist normalerweise ziemlich schmal und erscheint am linken Rand.

3. Falls unter dem Fensternamen Textbausteine noch nicht Java steht, sondern ein anderer Name wie z.B. HTML Tags oder DOS Characters, dann sollten Sie auf diesen anderen Namen linksklicken und aus der sich öffnenden Liste Java wählen. Falls Java in der Liste nicht vorkommt, sitzen Sie wahrscheinlich nicht vor einem SWE-Labor-Rechner, sondern vor Ihrem eigenen Rechner, und müssen erstmal den nächsten Tipp (Textbausteine zwischen Rechnern übertragen) befolgen.

4. Doppelklicken Sie im Textbausteine-Fenster auf den Textbaustein Klasse (mit main, pln). Der Quelltext einer Klasse mit dem (vorläufigen) Namen XX sollte in ihrem neuen Dokument erscheinen (vorerst schwarz-weiss, weil der TextPad noch nicht weiß, dass das neue Dokument eine `.java`-Datei werden soll).

5. Wählen Sie (markieren Sie) mit der Maus ein Vorkommen von XX (z.B. das in der Zeile 1).

6. Drücken Sie auf F8, um das Ersetzen-Fenster zu öffnen. Geben Sie in diesem Fenster hinter Ersetzen durch: einen Klassen-und-Programm-Namen ein (z.B. `Hallo99`) und klicken Sie auf den Knopf Alle ersetzen.

Anmerkung: Der Klassen-und-Programm-Name ist die einzige Zeichenkette, die Sie (einmal) eintippen müssen. Ansonsten sollten Sie nur *Kommandos* eingeben, aber keine weiteren *Daten* eintippen (weil das Eintippen von Daten besonders fehleranfällig ist und viel Zeit kostet).

7. Wählen Sie in Zeile 1 mit der Maus den vollständigen Datei-Namen (z.B. `Hallo99.java`) und kopieren Sie ihn mit STRG-C in die Ablage.

8. Öffnen Sie das Speichern unter - Fenster, indem Sie F12 (oder STRG-S) eingeben.
Kopieren Sie mit STRG-V den Inhalt der Ablage in das Eingabefeld neben Dateiname:.
Klicken Sie auf den Knopf Speichern.

Danach sollte der Programmtext nicht mehr schwarz-weiss, sondern *farbig* dargestellt werden (denn der TextPad schließt aus dem Namen der neuen Datei, dass es sich um eine Java-Datei handelt und färbt ihre Darstellung auf dem Bildschirm entsprechend).

9. *Übergeben* Sie das neue Programm *dem Ausführer*
(konkret: kompilieren Sie es, z.B. mit STRG-1).

10. Befehlen Sie dem Ausführer, das neue Programm *auszuführen* (z.B. mit STRG-2).

Wenn Sie für diese 10 Schritte deutlich mehr als 30 sec gebraucht haben, sollten Sie sie jeden Tag ein paarmal wiederholen, bis Sie unter 30 sec kommen :-).

Im Schritt 4. haben Sie den Textbaustein Klasse (mit main, pln) benutzt. Probieren Sie auch mal ein paar andere Textbausteine aus, z.B. den Mehrzeilenkommentar, die Minuszeile oder pln (3 Abkürzungen) etc. Wie die Bausteine mit einem Zirkumflex ^ am Ende ihres Namens funktioneren, wird im Tip 8 erläutert.

7. Textbausteine (eine .tcl-Datei) übertragen

Die Textbausteine, die der TextPad im SWE-Labor kennt (siehe vorigen Tip, Schritt 4.) können Sie wie folgt auch zum TextPad auf Ihrem Rechner übertragen:

1. Unten im Anhang A sind die etwa 230 Zeilen einer sogenannten *TexPad Clip Library* wiedergegeben. Kopieren Sie diese Zeilen in eine Datei namens `java.tcl` und bringen Sie diese Datei in den Ordner

namens `Samples` Ihrer TextPad-Installation. Typischerweise hat dieser Ordner den Pfadnamen `C:\Programme\TextPad 5\Samples` oder so ähnlich. Er sollte schon mehrere .tcl-Dateien enthalten, z.B. `doschar.tcl` und `htmlchar.tcl` etc.

2. Wenn Sie danach den TextPad neu starten, sollten Ihnen die Textbausteine in der Datei `java.tcl` unter dem Namen Java zur Verfügung stehen. Falls links neben dem großen Fenster des TextPads nicht gleich das schmale Textbausteine-Fenster angezeigt wird, müssen Sie einmal `Alt-0` eingeben, oder (falls das nicht funktioniert) ein oder zweimal `Strg-F3` eingeben, bis das Fenster sichtbar wird. Wenn dann zwar Textbausteine angezeigt werden, aber nicht die Java-Bausteine, sondern z.B. die DOS Characters-Bausteine, sollten Sie mit einem Linksklick auf DOS Characters ein Menü öffnen und darin Java wählen. Danach sollten einige Java-Bausteine angezeigt werden, von denen der oberste Mehrzeilenkommentar heißt.

8. Eigene Textbausteine schreiben

In die Datei `java.tcl` (siehe vorigen Tip) können Sie eigene Textbausteine einfügen oder Sie können eine weitere, ähnliche Datei (z.B. eine namens `meinJava.tcl`) erstellen und im Ordner `Samples` ablegen. Den Namen der Datei können Sie frei wählen, aber die Erweiterung `.tcl` muss sein, damit der TextPad die Datei als *TextPad Clip Library* erkennt.

Eine Baustein-Sammlung sollte mit Zeilen ähnlich den folgenden beginnen:

```
1 !TCL=123, von Ulrich Grude, September 2005
2 !TITLE=Java
3 !SORT=N
4 !CHARSET=ANSI
```

Die Zahl hinter `!TCL` kann ziemlich frei gewählt werden, sollte aber zwischen 1 und 999 liegen und *eindeutig* sein (d.h. alle gleichzeitig benutzen Sammlungen sollten *unterschiedliche* Nummern haben). Hinter `!TITLE=` gibt man den *Namen* der Baustein-Sammlung an. Diesen Namen kann man später im Textbausteine-Fenster wählen (siehe vorigen Tip). Weitere Erläuterungen zu diesen Zeilen findet man in der TextPad-Hilfe unter Textbausteine und Sammlungen direkt bearbeiten.

Ein typischer Textbaustein sieht z.B. so aus (die Zeilen-Nummern gehören *nicht* zum Baustein):

```
5 !TEXT=stat-pub-XXX-Methode
6    static public XXX (PP) {
7    } //
8
9 !
```

Die erste Zeile muss mit `!TEXT=` beginnen. Dahinter gibt man einen *Namen* für den Baustein an. Auf diesen Namen kann der Benutzer später doppelklicken, um den Baustein in sein Dokument einzufügen. Die letzte Zeile darf nur ein Ausrufezeichen `!` enthalten (wie hier in Zeile 9). Die Zeilen dazwischen (im Beispiel: Zeile 6 bis 8) enthalten den *Text* des Textbausteins.

Beachten Sie, dass der Text um drei Zeichen eingerückt ist, denn der TextPad kopiert ihn "zeichengetreu" ohne seine Einrückung zu verändern.

Die (leere) Zeile 8 bewirkt, dass nach einem Einfügen des Bausteins der Cursor *unter* dem eingefügten Text steht, und nicht *am Ende der letzten Zeile*. Wenn Sie das unbequem finden, können Sie die Zeile 8 auch weglassen.

Es folgt ein Textbaustein für Fortgeschrittene (Baustein-*Schreiber* und -*Anwender*):

```
10 !TEXT=stat-pub-void-Methode^
11    static public void \^ (PP) {
12    } //
13
14 !
```

Der Name dieses Bausteins (`stat-pub-void-Methode^`) endet mit dem Zeichen `^` (einem Zirkumflex). Damit wird dem Anwender signalisiert, dass er ein geeignetes Wort mit der Maus auswählen (mar-

kieren) sollte, bevor er den Baustein in sein Dokument einfügt. Die Zeichen \^ im Text des Bausteins (siehe Zeile 11) werden dann durch das ausgewählte Wort ersetzt.

Anmerkung: Leider wird nur das *erste Vorkommen* von \^ ersetzt. Besser wäre es, wenn alle Vorkommen von \^ ersetzt würden. Auch der TextPad ist noch nicht perfekt :-).

9. Zeilen-Nummern

Zeilen-Nrn. kann man im TextPad auf *zwei* voneinander unabhängige Weisen an- und ausschalten, wie im Folgenden erläutert wird.

9.1. Zeilen-Nummern auf dem Bildschirm anzeigen lassen

Menü Konfiguration, Einstellungen..., Ansicht, ganz unten
vor Zeilennummern ein Häkchen machen.

Achtung: Diese Zeilen-Nrn erscheinen nur *auf dem Bildschirm*, werden aber nicht in die editierte Datei geschrieben und werden auch *nicht gedruckt*. Diese Einstellung gilt pauschal für alle *Dokumentenklassen* (d.h. für Java-Dateien, Text-Dateien, XML-Dateien etc.).

9.2. Zeilen-Nummern ausdrucken lassen

Menü Konfiguration, Einstellungen, +Dokumentenklasse, +Java, Drucken
Vor Zeilennummern ein Häkchen machen.

Achtung: Diese Zeilen-Nrn erscheinen *nur beim Ausdrucken* (Menü Datei, Drucken...), werden aber nicht in die editierte Datei geschrieben und werden auch nicht auf dem Bildschirm angezeigt. Diese Einstellung kann (und muss) für jede Dokumentenklasse (z.B. für Java-Dateien, Text-Dateien, XML-Dateien etc.) einzeln an- bzw. abgeschaltet werden.

10. Java-Texte richtig einrücken

Wir gehen hier davon aus, dass der TextPad eine Dokumentenklasse namens *Java-Dateien* kennt, dass alle Dateien, deren Namen mit der Erweiterung `.java` enden, zu dieser Dokumentenklasse gehören und dass Sie gerade eine Datei dieser Klasse editieren (z.B. eine Datei namens `Hallo.java`).

Solange Sie konsequent darauf verzichten, die Cursor-Tasten (←, ↑, → und ↓) zu benutzen oder den Cursor mit der Maus zu positionieren, rückt der TextPad den Text, den Sie eingeben (bei der Eingabe) korrekt ein, indem er sich an die folgenden 3 Regeln hält:

Regel R1: Wenn eine Zeile mit einer *öffnenden* geschweiften Klammer { *endet*, wird die nächste Zeile um eine Stufe *mehr* eingerückt als ihre Vorgängerin.

Regel R2: Wenn eine Zeile mit einem anderen Zeichen (ungleich {) *endet*, wird die nächste Zeile *genau so weit* eingerückt wie ihre Vorgängerin.

Regel R3: Wenn eine Zeile mit einer *schließenden* geschweiften Klammer } *beginnt*, wird sie um eine Stufe *weniger* eingerückt als ihre Vorgängerin.

Beispiel: Geben Sie in eine Datei namens `Otto.java` die folgenden Zeilen ein und beobachten Sie genau was passiert, wenn Sie am Ende einer Zeile auf die Return-Taste drücken oder am Anfang einer Zeile eine schließende geschweifte Klammer } eingeben:

```
aaa
aaa {
   bbb
   bbb {
      ccc
      ccc
   }
}
```

Die beiden aaa-Zeilen werden *gleich weit* eingerückt, weil die erste *nicht* mit { endet (R1).
Die erste bbb-Zeile wird um eine Stufe *mehr* eingerückt, weil ihre Vorgängerin mit { endet (R2).

Die beiden bbb-Zeilen werden *gleich weit* eingerückt, weil die erste *nicht* mit { endet (R1).
Die erste ccc-Zeile wird noch *mehr* eingerückt, weil ihre Vorgängerin mit { endet (R2).
Die beiden ccc-Zeilen werden *gleich weit* eingerückt, weil die erste *nicht* mit { endet (R1).

Wenn Sie am Ende der letzten ccc-Zeile auf Return drücken, wird der Cursor erstmal unter dem ersten c positioniert. Aber wenn Sie dann das Zeichen } eingeben, erscheint es nicht an dieser Stelle, sondern um eine Einrückstufe weiter links (unter dem ersten b, wegen R3). Dieses Verhalten des TextPad ist besonders merk-würdig (wörtlich gemeint).

Wenn Sie danach wieder auf Return drücken, wird der Cursor erstmal unter dem Zeichen } (und somit auch unter dem ersten b) positioniert. Aber wenn Sie dann das Zeichen } eingeben, erscheint es nicht an dieser Stelle, sondern um eine Einrückstufe weiter links (unter dem ersten a, wegen R3).

Die Regeln 1 bis 3 beschreiben vollständig, wie der TextPad Java-Quelltexte richtig einrückt. Stören Sie ihn möglichst wenig dabei :-).

Anmerkung: Der TextPad kann Text nur *während der Eingabe* richtig einrücken. Wenn man einen Text chaotisch eingegeben hat (was mit Hilfe der Cursortasten ←, ↑, → und ↓ und der Maus möglich und gar nicht so schwer ist :-), hilft der TextPad einem *nicht* mehr beim Formatieren. Man muss den Text dann (mühsam) von Hand formatieren oder mit einem geeigneten Programm (z.B. mit Dr. Java, NetBeans oder Eclipse oder ...). Programmtexte gleich richtig eingerückt einzugeben ist deutlich eleganter und empfehlenswerter, als chaotisch eingeben und nachträglich reparieren.

Merke: Die Einrückung eines Quelltextes ist *keine Verzierung*, sondern ein wichtiges *Arbeitsmittel* (welches das Lesen des Textes und das Finden von Fehlern erheblich erleichtern kann).

11. Arbeitsbereiche: Wozu und wie?

WennSie möchten, dass beim Starten des TextPad alle Dateien geöffnet werden, die beim Beenden der letzten Sitzung geöffnet waren, sollten Sie zuerst einmal folgende Einstellung vornehmen:

Menü Konfiguration, Einstellungen, Allgemein
Vor Beim Start die zuletzt bearbeiteten Dateien laden ein Häkchen machen.

Das allein reicht aber nicht aus. Zusätzlich müssen Sie einen *Arbeitsbereich* (eine `.tws`-Datei, tws wie *TextPad Work Space*) erstellen, etwa so:

Menü Datei, Arbeitsbereich, Speichern unter

Das Fenster Speichern unter sollte aufgehen.

Navigieren Sie zu dem Ordner, in dem die `.tws`-Datei abgelegt werden soll (voreingestellt ist der aktuelle Arbeitsordner).

Geben Sie hinter Dateiname: einen Namen für die Datei an, z.B. `JavaAB01`
(die Erweiterung `.tws` wird automatisch ergänzt).

Klicken Sie auf Speichern.

Wenn Sie den TextPad beenden, wird er in dieser `.tws`-Datei Informationen über alle gerade geöffneten Dateien speichern. Und wenn Sie den TextPad durch einen Klick (oder Doppelklick) auf sein Symbol (engl. icon) starten, wird er alle in dieser Datei beschriebenen Dateien öffnen (und in jeder Datei wird der Cursor genau an der gleichen Stelle stehen wie beim letzten Beenden des TextPad und falls damals etwas ausgewählt war, wird es wieder ausgewählt sein).

Sie können auch mehrere Arbeitsbereiche (mehrere `.tws`-Dateien) erstellen und den TextPad starten, indem Sie auf eine davon doppelklicken.

Weitere Einzelheiten zu diesem Thema finden Sie in der TextPad-Hilfe unter dem Stichwort Arbeitsbereich.

12. Den Inhalt einer Datei hexadezimal darstellen

Anmerkung: Mit den Bezeichnungen *binäre Darstellung*, *hex-Darstellung* und *hexadezimale Darstellung* ist häufig dieselbe Art von Darstellung gemeint. Bei dieser Darstellung wird jedes Byte eines Dokuments durch zwei hex-Ziffern dargestellt, von denen jede für vier Binärziffern steht. Z.B. steht die hex-Ziffer `A` für die vier Binärziffern `1010` und die hex-Ziffer `3` steht für `0011`. Auf diese Weise kann man auch transparente Zeichen (Leerzeichen, Tab-Zeichen), nicht-druckbare Zeichen (Zeilenwechsel, Seitenwechselzeichen etc.) und andere Byte-Inhalte lesbar machen.

Der TextPad kann nicht nur *Textdateien* (z.B. Java-Quelldateien) darstellen, sondern auch sog. *Binärdateien* (z.B. `.exe`-, `.class`- oder `.wav`-Dateien etc.). Textdateien werden "wie allgemein bekannt" dargestellt. Dagegen werden Binärdateien auf dem Bildschirm hexadezimal dargestellt.

Wenn Sie eine Datei öffnen, entscheidet normalerweise der TextPad, ob er sie als Textdatei oder als Binärdatei darstellen soll (und da er viel Erfahrung hat, sind seine Entscheidungen in aller Regel sehr gut :-). Wenn Sie wollen, können Sie ihm die Entscheidung aber auch abnehmen und z.B. eine Java-Quelldatei als Binär-Datei darstellen lassen, etwa so:

Öffnen Sie das Fenster Datei(en) öffnen, z.B. indem Sie STRG-O (O wie open) eingeben. Dieses Fenster sieht etwa so aus:

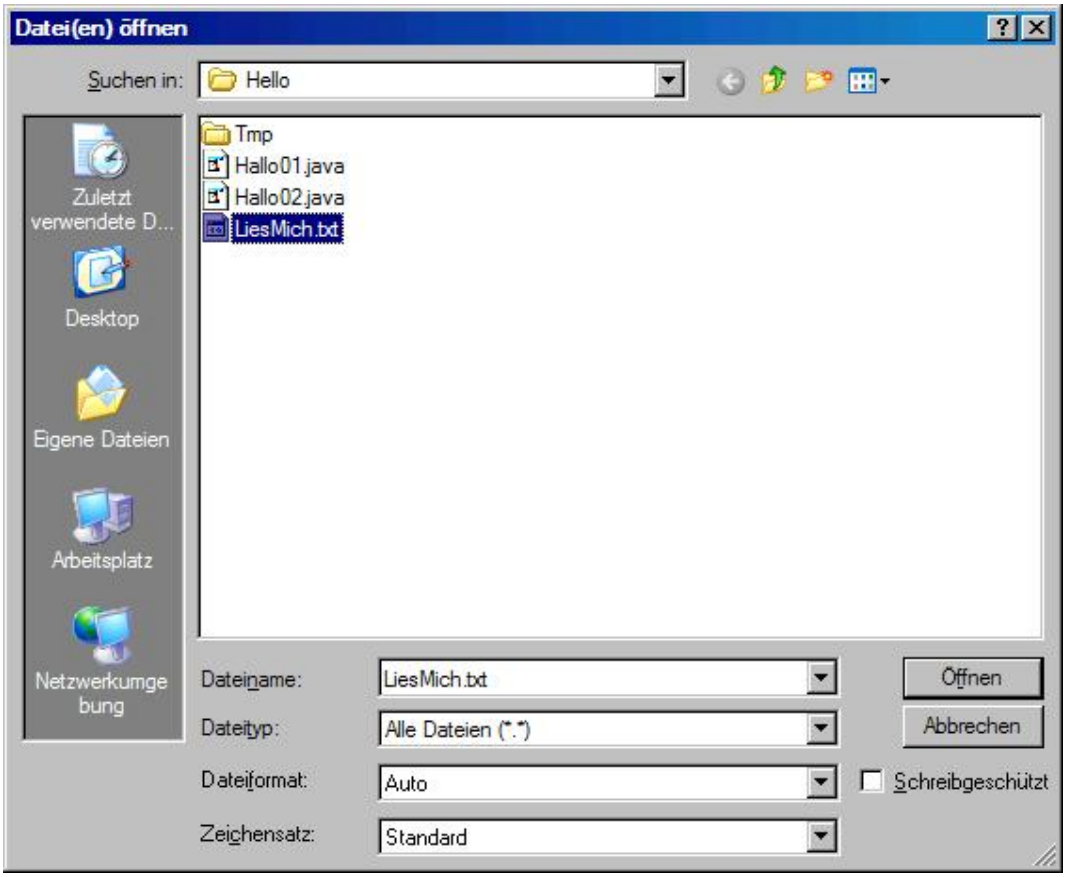

Abb. 1: Eine Datei öffnen (Auto, Text oder binär?)

Hier zeigt es gerade den Inhalt eines Ordners `Hello` an, in dem sich ein Ordner namens `Tmp` und drei Dateien befinden. Der Benutzer ist offenbar gerade dabei, eine Datei mit dem Namen LiesMich.txt zu öffnen.

Der Eintrag Auto im Eingabefeld Dateiformat: (im unteren Rand des Fensters) bedeutet, dass der TextPad (automatisch) entscheiden soll, ob er die Datei als *Text* oder besser als *Binärdatei* öffnet. Statt Auto kann man in diesem Eingabefeld auch Text oder `binär` wählen um selbst zu bestimmen, dass die Datei als Text- bzw. als Binärdatei geöffnet werden soll.

Lässt man die Datei `LiesMich.txt` als Binärdatei öffnen, stellt der TextPad ihren Inhalt z.B. so dar:

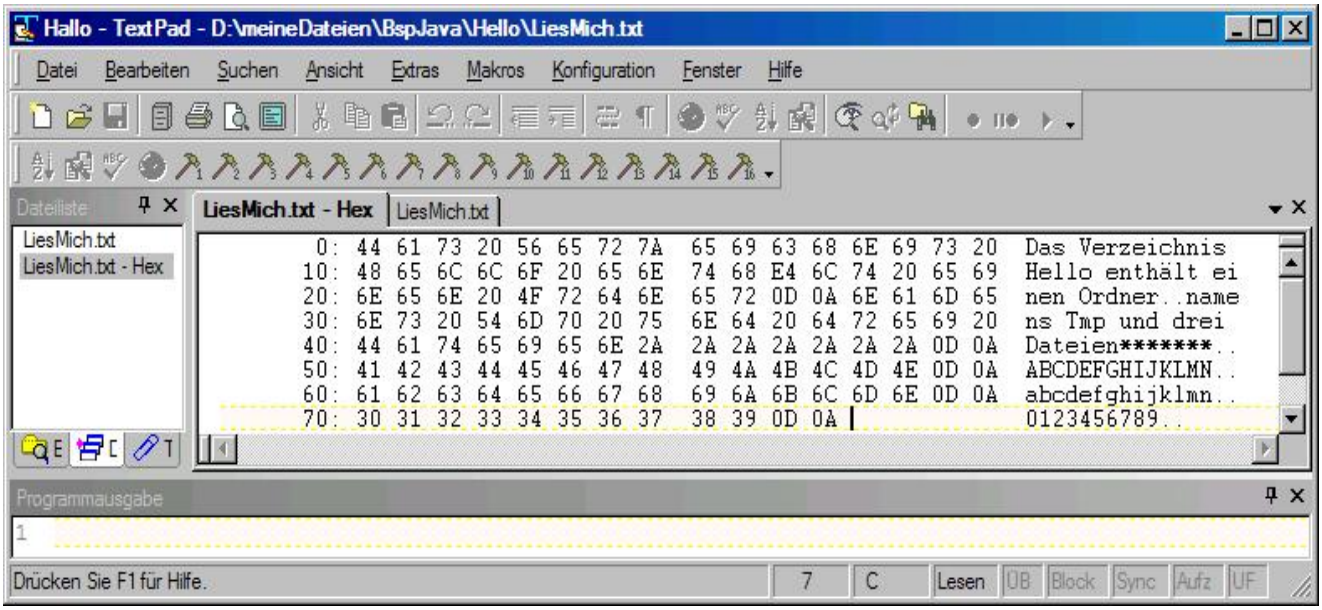

Abb. 2: Eine Textdatei in hexadezimaler Darstellung

Jede Zeile der Darstellung beginnt mit einer (hexadezimalen) Adresse 0, 10, 20, 30, ... etc. Dann kommen 16 Bytes in hexadezimaler Darstellung (z.B. `44 61 73 ...`) und am Ende der Zeile dieselben 16 Bytes als Zeichen dargestellt (z.B. `Das Verzeichnis`). Nicht-druckbare Zeichen werden hier durch einen Punkt `.` repräsentiert. Ob ein Punkt einen Punkt oder ein nicht-druckbares Zeichen repräsentiert, kann man an Hand der hexadezimalen Darstellung unterscheiden.

In der Dateiliste (das ist das schmale Fenster ganz links in der Abbildung, darüber steht "Dateiliste" weiss auf grau, nicht ganz leicht zu lesen) erkennt man, dass der Benutzer die Datei `LiesMich.txt` *zweimal* geöffnet hat: Einmal als Text und einmal als Binär-Datei (erkennbar an dem Zusatz `- Hex` nach dem Dateinamen).

In einem TextPad-Fenster mit einer binären Darstellung kann man grundsätzlich *keine* Veränderungen vornehmen (Daten *einzugeben* oder *zu löschen* etc. ist nicht möglich). In einem Fenster mit einer Text-Darstellung kann man normalerweise zwar Veränderungen vornehmen, aber solange dieselbe Datei auch noch binär geöffnet ist, kann man die Änderungen nicht abspeichern.

13. Den Partner einer Klammer finden

Wenn Sie (in irgendeinem TextPad-Fenster) den Cursor vor oder hinter einer *Klammer* positionieren (oder die Klammer auswählen) und `STRG-M` eingeben (M wie match), springt der Cursor automatisch zur "Partner-Klammer". Falls der Cursor sich nicht bewegt, gibt es keine Partner-Klammer.

Als Klammern gelten dabei die folgenden 8 Zeichen: `( [ { < > } ] )`

Dieser `STRG-M`-Befehl kann einem das Entfernen einer überzähligen oder das Einfügen einer fehlenden Klammer erheblich erleichtern. Das gilt besonders für *geschweifte Klammern*, die in Java-Programmen tief geschachtelt vorkommen können.

14. Die Einrücktiefe und die Behandlung von Tab-Zeichen konfigurieren

Die Einrückung von Programmtexten kann man durch *Tabulator-Zeichen* (kurz: Tab-Zeichen) oder durch *Leerzeichen* realisieren. Beide Möglichkeiten haben Vor- und Nachteile. Leerzeichen haben den Vorteil, dass sie von allen Editoren gleich dargestellt werden (was für Tab-Zeichen nicht gilt). Der TextPad "kann" beide Möglichkeiten. Die entsprechenden Einstellungen kann (und muss) man für jede Dokumentenklasse separat vornehmen. Für *Java-Dateien* geht das etwa so:

Menü Konfiguration, Einstellungen, +Dokumentenklasse, +Java, Tabulator

Dadurch sollte ein Fenster wie das in Abb. 3 aufgehen.

Hinter Standard-Tabstopps: und Einzugsgröße: sollte man *die Breite einer Einrückstufe* eingeben. Die Wirkung der beiden Häkchen vor Neue Tabzeichen in Leerzeichen umwandeln und Vorhandene Tabulatoren beim Speichern in Leerzeichen umwandeln geht aus der Beschriftung ziemlich klar hervor (mit "Tabulatoren" sind Tab-Zeichen gemeint).

Abb. 3: Einrücktiefe und Behandlung von Tab-Zeichen festlegen

Anmerkung: Die Beispielprogramme zum Buch "Java ist eine Sprache" sind alle mit den in Abb. 3 wiedergegebenen Einstellungen erstellt worden. Das Benutzen und Bearbeiten der Beispielprogramme mit dem TextPad wird besonders einfach, wenn man die *gleichen* Einstellungen verwendet.

15. Nützliche Tasten-Kürzel

Um eine ganze Zeile in die Ablage zu kopieren oder auszuschneiden, kann man sie mit der Maus auswählen und dann mit `STRG-C` kopieren bzw. mit `STRG-X` ausschneiden.

Schneller geht es, wenn man den Cursor irgendwo in der zu bearbeitenden Zeile positioniert und sie mit dem Tastenkürzel `ALT-C` in die Ablage kopiert oder mit `ALT-Y` löscht.

Die Tastenkürzel `ALT-C` und `ALT-Y` sind im TextPad nicht fest vorgegeben, können vom Benutzer aber relativ leicht definiert werden (und sind im SWE-Labor bereits definiert).

Wie definiert man Tastenkürzel? Ein Beispiel:

Im Menü Ansicht gibt es einen Punkt Ganzer Bildschirm, den man an- und wieder abschalten kann. Damit schaltet man von der *Normal-Ansicht* zur *Ganzer-Bildschirm-Ansicht* bzw. umgekehrt von der *Ganzer-Bildschirm-Ansicht* zurück zur *Normal-Ansicht*. Als Beispiel soll für diesen Menüpunkt `ALT-Z` als Tastenkürzel definiert werden. Dazu geben wir folgenden Befehl ein:

Konfiguration, Einstellungen, Tastatur

Ein Fenster wie das folgende sollte sich öffnen:

Unter Kategorien: wählen wir

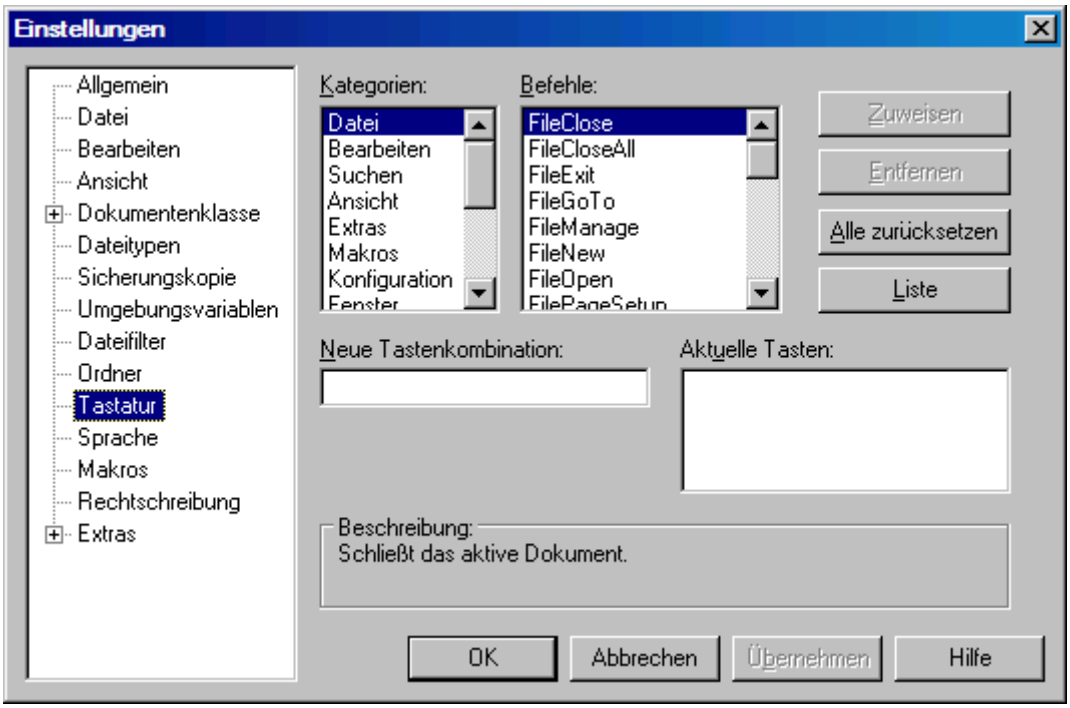

Abb. 4: Ein Tastenkürzel definieren

Ansicht und unter Befehle: den Punkt ViewFullScreen. Dann positionieren wir den Cursor im Fensterchen unter Neue Tastenkombination: und geben dort das neue Tastenkürzel ALT-Z ein (d.h. wir halten die Alt-Taste gedrückt und drücken zusätzlich auf die Z-Taste). Zum Abschluss klicken wir auf den Knopf Zuweisen (rechts oben), auf den Knopf Übernehmen (rechts unten) und schließlich auf OK. Fertig.

Als Test geben wir dann mehrmals nacheinander ALT-Z ein und beobachten, ob wirklich zwischen der Normal-Ansicht und der Ganzer-Bildschirm-Ansicht hin und her gewechselt wird.

In den *Menüs* des TextPad (Datei, Bearbeiten, Suchen, ..., Hilfe) werden für viele Menüpunkte bereits entsprechende Tastenkürzel *angezeigt*. Die wichtigsten dieser Kürzel (für die Menüpunkte, die man am häufigsten benutzt) sollte man möglichst bald auswendig lernen. Dadurch kann man seine Arbeitsgeschwindigkeit erheblich erhöhen.

Es folgt eine Tabelle mit wichtigen Kürzeln (in die noch freien Kästchen können Sie ihre Favoriten eintragen):

Kürzel	Kurzbeschreibung	Kürzel	Kurzbeschreibung
STRG-1	Java kompilieren	STRG-M	Partner-Klammer finden <([{}])>
STRG-2	Java-Programm starten	STRG-Q I	Nicht-druckbare Zeichen zeigen
ALT-C	Aktuelle Zeile kopieren	STRG-L	Markierte Buchstaben *klein* machen
ALT-Y	Aktuelle Zeile löschen	STRG-U	Markierte Buchstaben *groß* machen
F5	Suchen	STRG-EINGABE	Neue Zeile nach aktueller Zeile
F8	Suchen und Ersetzen	STRG-UMSCHALT-EINGABE	Neue Zeile vor aktueller Zeile
STRG-P	Aktives Dok. drucken	STRG-UMSCHALT-P	Seitenansicht (vor dem Drucken)
STRG-F	Nächstes suchen	STRG-UMSCHALT-F	Voriges Markiertes suchen
STRG-I	Markiertes/aktuelle Zeile *mehr* einrücken	STRG-UMSCHALT-I	Markiertes/aktuelle Zeile *weniger* einrücken
STRG-S	Aktives Dok. speichern	STRG-UMSCHALT-S	Alle Dokumente speichern
STRG-T	Zeichen/Zeilen vor und nach Cursor vertauschen	STRG-UMSCHALT-T	Wörter vor und nach dem Cursor vertauschen
STRG-TAB	Nächstes Fenster	STRG-UMSCHAL-T-TAB	Voriges Fenster
STRG-FI	Kurze Statistik in der Statuszeile	STRG-F3	Textbausteine anzeigen/verbergen
F11	Dateiliste anzeigen/verbergen	STRG-F11	Programmausgabefenster anzeigen/verbergen

16. Der TextPad-Befehl "Java kompilieren"

Wenn Sie mit dem TextPad eine `.java`-Datei erstellt haben, können Sie sie sofort dem Java-Ausführer übergeben (konkret: kompilieren), ohne den TextPad zu verlassen, und zwar so:

Menü Extras, Benutzer-Programme, Java kompilieren (oder: STRG-1)

Achtung: Der TextPad bezieht solche Befehle immer auf *das aktive Dokument* (siehe oben Tipp 2). Falls z.B. das Fenster Programmausgabe aktiv ist und Sie den obigen Befehl eingeben, wird die Programmausgabe dem Ausführer übergeben. Der Ausführer (konkreter: der Java-Compiler) wird sie ablehnen (weil sie nicht in einer `*.java`-Datei steht) und eine Fehlermeldung ähnlich der folgenden ausgeben (in das Fenster Programmausgabe):

```
Usage: javac <options> <source files>
use -help for a list of possible options
```

Was der TextPad-Befehl Java kompilieren genau bewirkt, können Sie festlegen und einstellen. Geben Sie dazu folgenden Befehl ein:

Menü Konfiguration, Einstellungen, +Extras, Java kompilieren

Ein Fenster wie das folgende sollte sich dadurch öffnen:

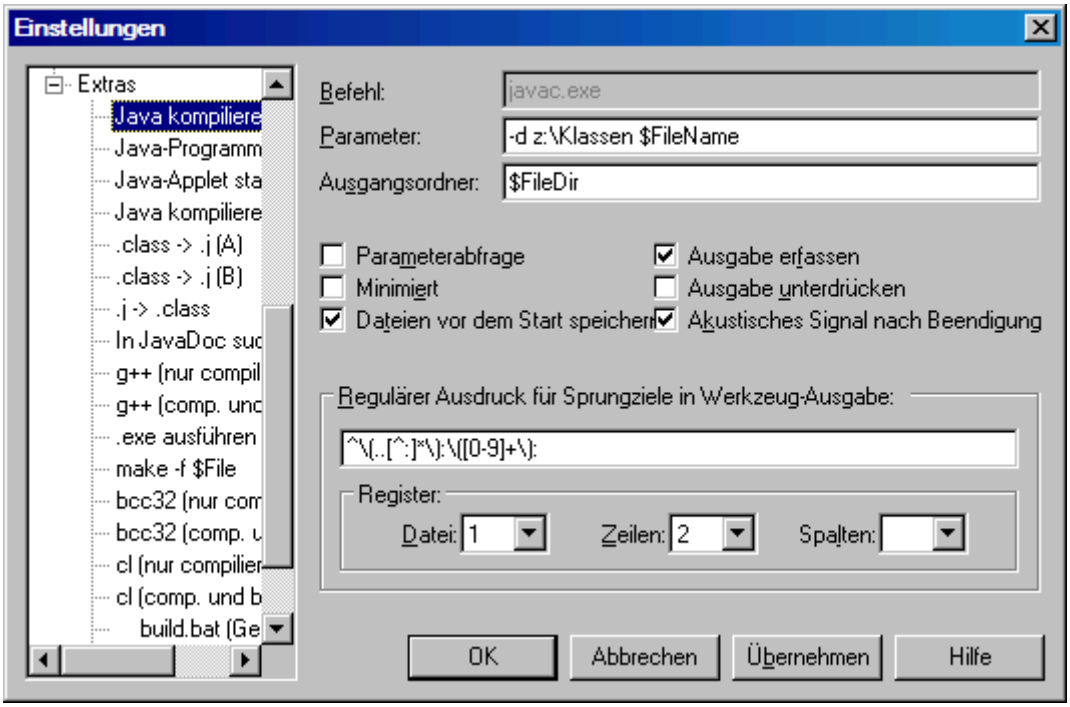

Abb. 5: Einstellungen für den Befehl Java kompilieren

Dieses Fenster enthält die Einstellungen für den Befehl Java kompilieren (der ganz links unter Extras hervorgehoben dargestellt wird).

Wenn man diesen Befehl aufruft, wird ein Programm namens javac.exe gestartet (siehe das waagerechte Fenster neben Befehl:). Das ist der Java-Compiler der Firma Sun. Warum javac.exe nur grau angezeigt wird und nicht geändert werden kann, ist dem Autor dieses Papiers nicht klar.

Im waagerechten Fenster neben Parameter: kann man Parameter für das Programm javac.exe angeben. Der Makroname $FileName (siehe auch unten Tipp 17) wird vom TextPad durch den Namen des aktiven Dokuments ersetzt (siehe oben Tipp 2). Davor steht die Option -d z:\Klassen. Sie bewirkt, dass der Java-Compiler alle `.class`-Dateien, die er erzeugt, (nicht im Arbeitsordner sondern) im Ordner z:\Klassen ablegt. Diese Einstellung ist im SWE-Labor richtig. Auf Ihrem Rechner sollten Sie einen entsprechenden Ordner erstellen (z.B. `C:\meineDateien\Klassen`) und hinter -d angeben. Das -d soll an directory for .class-files erinnern.

In dem Fensterchen unter Regulärer Ausdruck für Sprungziele in Werkzeug-Ausgabe: steht der folgende *reguläre Ausdruck*:

```
^\(..[^:]*\):\([0-9]+\):
```

Er beschreibt, *wo* in einer Fehlermeldung (des Java-Compilers javac.exe) der *Datei-Name* und wo die *Zeilen-Nr.* steht, auf die sich die Meldung bezieht Er macht es möglich, dass man durch einen Doppelklick auf die erste Zeile einer Fehlermeldung automatisch zu der fehlerhaften Stelle im Quellprogramm springt.

Beispiel: Eine typische Fehlermeldung des Java-Compilers von Sun sieht etwa so aus:

```
Hallo01.java:3: ';' expected
    static public void main(String[] sonja)
                                           ^
```

Man erkennt, dass in der ersten Zeile dieser Meldung der Datei-Name `Hallo01.java` und die Zeilen-Nr. `3` steht. Wenn man auf diese erste Zeile der Meldung doppelklickt, springt der TextPad in die Zeile `3` der Datei `Hallo.java`. Die Zeichenketten `Hallo01.java` und `3` findet er mit Hilfe des oben angegeben regulären Ausdrucks.

17. Der TextPad-Befehl "Java-Programm starten"

Nachdem man alle Quelldateien eines Java-Programms dem Ausführer übergeben (d.h. kompiliert) hat und der Ausführer sie alle akzeptiert hat (statt sie mit Fehlermeldungen abzulehnen), kann man das Programm *starten*, etwa so:

Menü Extras, Benutzer-Programme, Java-Programm starten (oder: STRG-2)

Achtung: Bevor man ein Programm startet, sollte man die Datei mit der *Hauptklasse* des Programms aktivieren (d.h. zum aktiven Dokument machen, siehe oben Tipp 2). Die *Hauptklasse* ist die, die genau so heißt wie das Programm und die `main`-Methode des Programms enthält. Einfache Programme bestehen nur aus *einer* (Haupt-) Klasse.

Was der TextPad-Befehl Java-Programm starten genau bewirkt, können Sie festlegen und einstellen. Geben Sie dazu folgenden Befehl ein:

Menü Konfiguration, Einstellungen, +Extras, Java-Programm starten

Ein Fenster wie das folgende sollte sich dadurch öffnen:

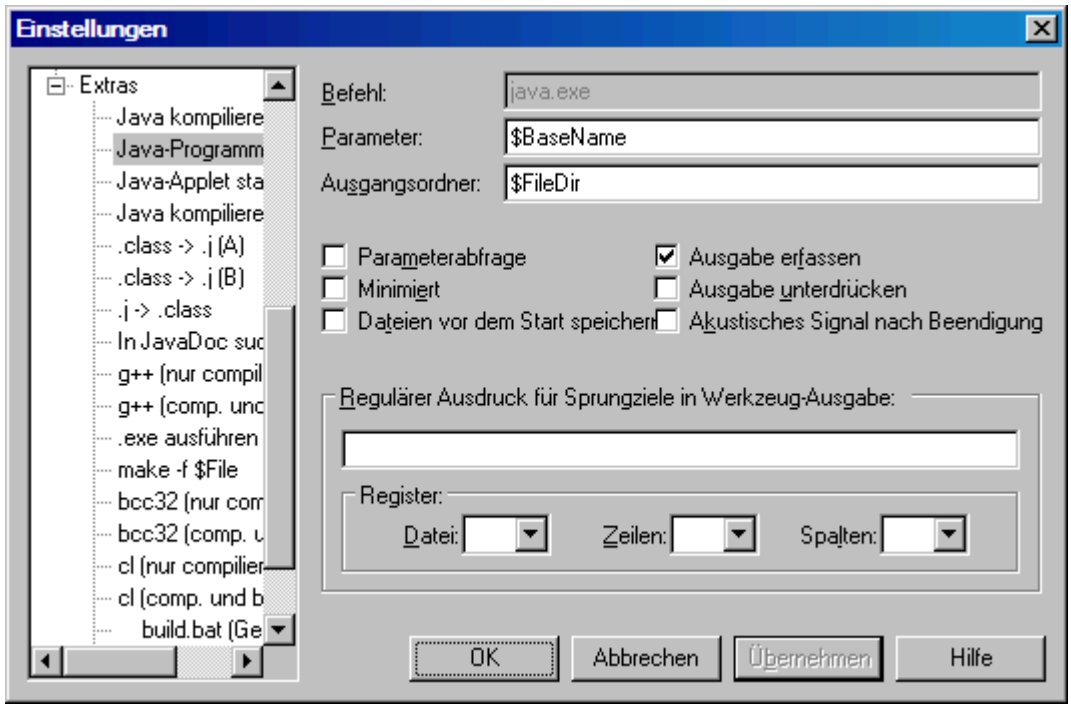

Abb. 6: Einstellungen für den Befehl Java-Programm starten

Dieses Fenster enthält die Einstellungen für den Befehl Java-Programm starten (der ganz links unter Extras hervorgehoben dargestellt wird).

Wenn man diesen Befehl aufruft, wird ein Programm namens java.exe gestartet (siehe das waagerechte Fenster neben Befehl:). Das ist der Java-Bytecode-Interpreter der Firma Sun. Warum java.exe nur grau angezeigt wird und nicht geändert werden kann, ist dem Autor dieses Papiers nicht klar.

Im waagerechten Fenster neben `Parameter:` kann man Parameter für das Programm `java.exe` angeben. Der Makronamen $BaseName (siehe auch unten Tipp 17) wird vom TextPad durch den Namen der aktiven Datei (ohne Erweiterung) ersetzt. Beispiel: Wenn die aktive Datei `Hallo.java` heißt, wird $BaseName durch `Hallo` (ohne `.java`) ersetzt.

Normalerweise wird das durch $BaseName bezeichnete Java-Programm in einer *DOS-Eingabeaufforderung* gestartet. Das ist ein Fenster mit schwarzem Hintergrund und weißer Schrift, in dem die *Ausgaben* des Programms angezeigt werden und in das man *Eingaben* für das Programm eingeben kann.

Das Häkchen vor Ausgabe erfassen bewirkt, dass *keine* DOS-Eingabeaufforderung geöffnet wird. Statt dessen "fängt der TextPad die Ausgabe des Programms ab" und zeigt sie in einem TextPad-Fenster. Dieses TextPad-Fenster bietet mehr Komfort als eine DOS-Eingabeaufforderung (man kann rauf- und runter-scrollen, Text auswählen und kopieren, Umlaute äöüÄÖÜ werden korrekt angezeigt, ...).

Achtung: Dieses TextPad-Fenster dient dem gestarteten Programm als *Standard-Ausgabe* und als *Standard-Eingabe*, es ist aber *nur als Ausgabe geeignet*. Falls das Programm versucht, etwas von seiner Standard-Eingabe einzulesen (z.B. mit `EM.liesInt()` oder einem ähnlichen Befehl) tritt ein Fehler auf.

Das Häkchen vor Ausgabe erfassen ist also nur sinnvoll bei Programmen, die *keine Daten von der Standardeingabe einlesen*. Programme, die von ihrer Standard-Eingabe lesen, sollten (ohne dieses Häkchen) in einer DOS-Eingabeaufforderung gestartet werden. Die ist (zwar unkomfortabel aber) als Ausgabe *und als Eingabe* geeignet.

18. Makronamen für Dateien in TextPad-Befehlen

Angenommen, im aktiven Fenster wird die Datei `z:\Java\Beispiele\Hallo01.java` angezeigt. Dann haben in einem TextPad-Befehl wie etwa Java kompilieren oder Java-Programm starten (siehe die vorigen beiden Tipps) die folgenden Makronamen die angegebenen Bedeutungen:

Makroname	Bedeutung	Erläuterung
`$File`	`z:\Java\Beispiele\Hallo01.java`	Voller Name der Datei
`$FileName`	`Hallo01.java`	Nur Name der Datei (*mit* Erweiterung)
`$BaseName`	`Hallo01`	Name *ohne* Erweiterung
`$FileDir`	`z:\Java\Beispiele\`	Nur Verzeichnis, ohne Dateiname
`$FilePath`	`\Java\Beispiele\Hallo01.java`	Voller Name ohne Laufwerk
`$UNIXPath`	`/Java/Beispiele/Hallo01.java`	Voller Name in Unix-Notation

Weitere Informationen zu diesem Thema findet man in der TextPad-Hilfe unter dem Stichwort Makros für Werkzeug-Parameter.

19. In mehreren Dateien nach einer Zeichenkette suchen

Angenommen, Sie haben in einem Ordner namens `z:\Java\BspJasp` zahlreiche Beispielprogramme und möchten wissen, in welchen davon ein bestimmter Befehl (z.B. der Befehl `continue;`) vorkommt. Mit dem TextPad können Sie das etwa so herausfinden:

Menü Suchen, In Dateien suchen... (oder: STRG-F5)

Ein Fenster wie das folgende sollte sich öffnen. Darin können Sie eine *Suche in mehreren Dateien* starten (nachdem Sie eingetragen haben, *wo* nach *was* gesucht werden soll):

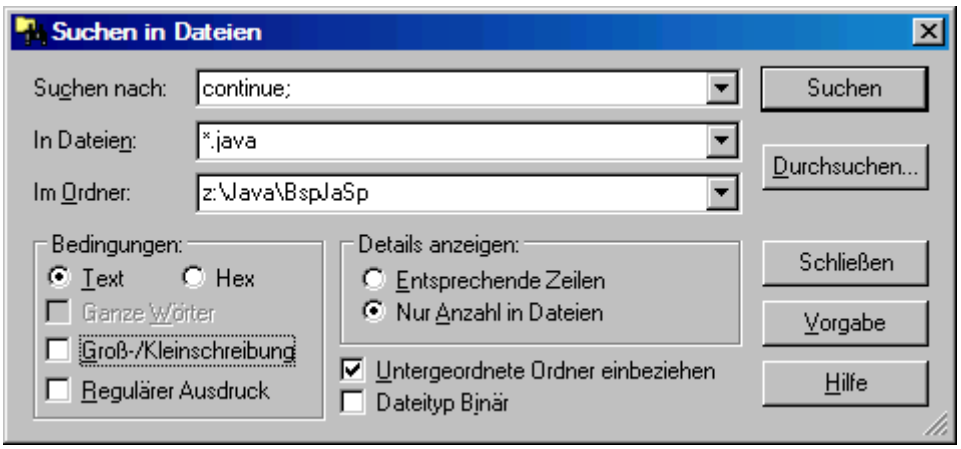

Abb. 7: Eine Suche in mehreren Dateien beschreiben

Neben Suchen nach: geben Sie die Zeichenkette ein, nach der gesucht werden soll.

Neben In Dateien: können Sie die Suche auf bestimmte Dateien einschränken, z.B. auf solche, deren Namen mit der Erweiterung `.java` enden. Soll in allen Dateien gesucht werden, sollten Sie hier nur einen Stern * eintragen.

Neben Im Ordner: geben Sie den Ordner an, in dem gesucht werden soll. Wenn auch in allen *Unterordnern* dieses Ordners (und rekursiv in den Unterordnern dieser Unterordner und in den ... etc.) gesucht werden soll, müssen Sie vor Untergeordnete Ordner einbeziehen ein Häkchen ma-

chen. Statt neben Im Ordner etwas *einzutippen*, können Sie auch auf Durchsuchen... klicken und (mit Maus-Klicks) zu dem Ordner hin *navigieren*.

Es empfiehlt sich, den Knopf Durchsuchen... möglichst selten mit dem dicht darüber stehenden Knopf Suchen zu verwechseln :-). Statt Durchsuchen... sollte es eigentlich Zu dem zu durchsuchenden Ordner navigieren heißen, aber diese Bezeichnung passte wohl nicht auf den Knopf :-).

Details anzeigen: Angenommen, die gesuchte Zeichenkette kommt in einer Datei namens `Hallo17.java` *dreimal* vor. Wenn Sie unter Details anzeigen: die Alternative Entsprechende Zeilen wählen, wird das Suchergebnis für die Datei `Hallo17.java` *drei* Zeilen enthalten, etwa wie die folgenden:

```
Hallo17.java(147): if (a > b) continue;
Hallo17.java(158): continue; // Fuer seltene Faelle!
Hallo17.java(190): continue;
```

Die Zahlen 147, 158, ... sind Zeilen-Nummern. Der Text dahinter ist der Inhalt der betreffenden Zeile. Natürlich enthält jede der Zeilen die gesuchte Zeichenkette `continue;` .

Wählen Sie dagegen (unter Details anzeigen:) die Alternative Nur Anzahl in Dateien, dann enthält das Suchergebnis für die Datei `Hallo17.java` nur *eine* Zeile, die mit der *Anzahl der Vorkommen* endet, etwa so:

```
z:\Java\BspJaSp\Hallo17.java: 3
```

Hier ist 3 eine *Anzahl* und keine *Zeilen-Nummer*.

Mit einem Klick auf den Knopf Suchen startet man die Suche. Das Ergebnis wird in einem speziellen TextPad-Fenster namens Such-Ergebnis angezeigt und besteht aus Zeilen ähnlich den oben (nach Details anzeigen:) angegebenen.

Trip: Ein Doppelklick auf eine der Ergebniszeilen (im Fenster Such-Ergebnis) öffnet die betreffende Datei. Und falls die angeklickte Zeile mit einer Zeilen-Nummer endet, positioniert der TextPad den Cursor in dieser Zeile. Das ist sehr praktisch.

20. Reguläre Ausdrücke

Vermutlich sind Sie bereits mit den folgenden regulären Ausdrücken vertraut:

Der reguläre Ausdruck	bezeichnet alle (Datei-) Namen die
`*.txt`	mit der Erweiterung `.txt` enden
`Hallo.*`	mit `Hallo.` beginnen
`*.???`	mit einer 3 Zeichen langen Erweiterung enden

Die regulären Ausdrücke des TextPad sind "verwandt" mit diesen Ausdrücken, aber im Detail ein bisschen anders und wesentlich mächtiger. Man kann sie an *zwei* verschiedenen Stellen verwenden:

1. Beim Konfigurieren von Befehlen wie Java kompilieren im Menü Extras (um durch einen Doppelklick auf eine Fehlermeldung zu der fehlerhaften Stelle im Quelltext zu springen, siehe dazu auch den Tipp 15).

2. Wenn man die Befehle Suchen nach..., Ersetzen... oder In Dateien Suchen... benutzt. Diese drei Befehle stehen im Menü Suchen und können mit den Tastenkürzeln F5, F8 bzw. STRG-F5 aufgerufen werden.

In diesem Tipp geht es nur um 2.

Der TextPad kann wahlweise mit einer nicht-standardisierten Varianten von regulären Ausdrücken arbeiten oder mit der POSIX-Varianten (POSIX steht für *Portable Operating System Interface for UNIX*). Einige der Beispiele in diesem Tipp funktionieren nur, wenn Sie mit dem folgenden Befehl die POSIX-Variante einschalten:

Menü Konfiguration, Einstellungen, Bearbeiten
Vor Reguläre Ausdrücke nach POSIX-Syntax verwenden ein Häkchen machen.

Ohne dieses Häkchen arbeitet der TextPad mit der nicht-standardisierten Varianten von regulären Ausdrücken.

Um ein Auto benutzen zu können, muss man erst eine Weile üben. Aber wenn man mal fahren kann, ist man häufig schneller, als wenn man zu Fuß geht. Ganz Ähnliches gilt für reguläre Ausdrücke.

Zu jedem der drei Befehle Suchen nach..., Ersetzen... und In Dateien Suchen... gehört ein *spezielles Fenster*. In jedem dieser Fenster kann man vor Regulärer Ausdruck ein Häkchen machen. Es hat folgende Wirkung: Was immer man im Eingabefeld Suchen nach: eingibt wird als regulärer Ausdruck interpretiert (z.B. haben die Zeichen * und ? dann eine besondere Bedeutung). Und beim Befehl Ersetzen... wird dann auch der Inhalt des Eingabefeldes Ersetzen durch: als regulärer Ausdruck interpretiert.

20.1. Nach einem regulären Ausdruck suchen

Was bedeuten reguläre Ausdrücke, die man in das Eingabefeld Suchen nach... eingegeben hat (nachdem man vor Regulärer Ausdruck ein Häkchen gemacht hat)? Ein paar Beispiele:

Regulärer Ausdruck	Welche Zeichenketten (ZK) findet man damit?	Beispiele für solche ZK
`Hallo`	Nur die ZK `Hallo`	`Hallo`

`Hallo..`	Alle ZK der Länge 7, die mit `Hallo` beginnen.
	Ein Punkt `.` steht für *ein beliebiges Zeichen* (ausser Zeilenwechsel!).
		`Hallo01`
		`HalloX7`
		`Hallo!+`

`Oh*`	Alle ZK mit `O` am Anfang und beliebig vielen `h` dahinter. Ein Stern `*` bedeutet 0-oder-mehr-von-dem-was-davor steht	`O`
		`Oh`
		`Ohhhhhhhhh`

`Ta(ta)*`	Alle ZK mit `Ta` am Anfang und beliebig oft `ta` dahinter. Die Klammern `( )` haben hier ein spezielle Bedeutung.	`Ta`
		`Tata`
		`Tatatatata`

`Oh+`	Alle ZK mit `O` am Anfang und mindestens einem `h` dahinter. Ein Pluszeichen `+` bedeutet 1-oder-mehr-von-dem-was-davor-steht.	`Oh`
		`Ohh`
		`Ohhhhhhhhh`

`Ta(ta)+`	Alle ZK mit `Ta` am Anfang und mindestens einmal `ta` dahinter. Die Klammern `( )` haben hier ein spezielle Bedeutung.	`Tata`
		`Tatata`
		`Tatatatata`

`Oh?`	Die zwei ZK mit `O` am Anfang und 0-oder-einem `h` dahinter. Ein Fragezeichen `?` bedeutet 0-oder-einmal-von-dem-was-davor-steht.	`O` `Oh`
`Ta(ta)?`	Die zwei ZK mit `Ta` am Anfang und 0-oder-einem `ta` dahinter. Die Klammern `( )` haben hier ein spezielle Bedeutung.	`Ta` `Tata`
`Oh{2,}`	Alle ZK mit `O` am Anfang und mindestens 2 mal `h` dahinter.	`Ohh` `Ohhhhhhh`
`Ta(ta){2,}`	Alle ZK mit `Ta` am Anfang und mindestens 2 mal `ta` dahinter.	`Tatata` `Tatatatatata`

Regulärer Ausdruck	Welche Zeichenketten (ZK) findet man damit?	Beispiele für solche ZK

`Oh{,4}`	Alle ZK mit `O` am Anfang und höchstens 4 mal `h` dahinter.	`O`
		`Oh`
		`Ohhhh`

`Ta(ta){,4}`	Alle ZK mit `Ta` am Anfang und höchstens 4 mal `ta` dahinter.	`Ta`
		`Tata`
		`Tatatatata`

`Oh{2,4}`	Alle ZK mit `O` am Anfang und mindestens 2, höchstens 4 mal `h` dahinter.	`Ohh`
		`Ohhh`
		`Ohhhh`

`Ta(ta){2,4}`	Alle ZK mit `Ta` am Anfang und mindestens 2, höchstens 4 mal `ta` dahinter.	`Tatata`
		`Tatatata`
		`Tatatatata`

`\(...\)`	Alle ZK, die aus drei Zeichen in runden Klammern bestehen. Die runden Klammern haben hier (wegen der Rückwärtsschrägstriche \ davor) *keine* spezielle Bedeutung)	`(ABC)`
		`(123)`
		`(\/\)`

`1.2`	Alle ZK der Länge 3 mit `1` am Anfang und `2` am Ende (und einem beliebigen Zeichen dazwischen). Der Punkt `.` hat hier die spezielle Bedeutung *irgendein Zeichen*.	`1A2`
		`1.2`
		`1+2`

`1\.2`	Nur die ZK `1.2`. Der Punkt . hat hier (wegen des Rückwärtsschrägstrichs \ davor) *keine* spezielle Bedeutung	`1.2`

`\\.\\`	Alle ZK, bei denen ein beliebiges Zeichen zwischen zwei Rückwärtsschrägstrichen steht. Der jeweils zweite Rückwärtsschrägstrich \ hat, wegen des Rückwärtsschrägstrichs \ davor, *keine* spezielle Bedeutung.	`\A\`
		`\7\`
		`\\\`

`^abc`	Die ZK `abc`, aber nur am *Anfang* einer Zeile. Das Zeichen Zirkumflex ^ bedeutet hier Zeilenanfang.	`abc`
`abc$`	Die ZK `abc`, aber nur am *Ende* einer Zeile. Das Zeichen Dollar $ bedeutet Zeilenende.	`abc`
`^abc$`	Die ZK `abc`, aber nur, wenn sie *allein* auf einer Zeile steht.	`abc`
`^$`	Damit findet man eine Leerzeile (aber nur *eine*! Mehrere Leerzeilen wie im übernächsten Beispiel!)	
`\n\n`	Damit findet man zwei Zeilenwechsel (n wie *newline*) ohne andere Zeichen dazwischen (d.h. eine Leerzeile)	
`\n\n\n`	Damit findet man drei Zeilenwechsel ohne andere Zeichen dazwischen (d.h. zwei Leerzeilen hintereinander)	
`[za?9!]`	Eines der Zeichen in den eckigen Klammern	`z` `!`

`[^ya?8!]`	Irgendein Zeichen welches *nicht* durch `[za?9!]` beschrieben wird. Ein Zirkumflex ^ als erstes Zeichen zwischen eckigen Klammern bedeutet *nicht* oder *alle ausser*.	`x`
		`z`
		`7`

`[a-e]`	Eines der Zeichen zwischen a und e (einschließlich)	`a`
		`b`
		`e`

Regulärer Ausdruck	Welche Zeichenketten (ZK) findet man damit?	Beispiele für solche ZK
`[^a-e]`	Irgendein Zeichen welches *nicht* durch `[a-e]` beschrieben wird.	`f` `8`

`[a-z][0-9]`	Einen kleinen Buchstaben gefolgt von einer Ziffer	`a0`
		`k5`
		`x9`

`[a-zA-Z0-Z]`	Einen (kleinen oder großen) Buchstaben oder eine Ziffer	`a`
		`B`
		`6`

`[A-Z]+[0-9]`	Einen oder mehrere Grossbuchstaben gefolgt von einer Ziffer	`A7` `MMZE3`
`abc|12`	`abc` oder `12`	`abc` `12`
`ab|b1|x|yy`	`ab` oder `b1` oder `x` oder `yy`	`ab` `x`
`abc|[0-9]`	`abc` oder eine Ziffer	`abc` `7`

Reguläre Ausdrücke mit vordefinierten Zeichenklassen:

Regulärer Ausdruck	Welche Zeichenketten (ZK) findet man damit?	Beispiele für solche ZK
`[[:alpha:]]`	Einen beliebigen (kleinen oder großen) Buchstaben	`A` `b`
`[[:digit:]]`	Eine Ziffer	`0` `9`

`[[:alnum:]]`	Einen Buchstaben oder eine Ziffer	`a`
		`B`
		`6`

`[[:space:]]`	Ein Leerzeichen, Tab-Zeichen, vertikales Tab-Zeichen oder Seitenvorschubzeichen	*diese Zeichen sind transparent*
`[[:cntrl:]]`	Ein Delete-Zeichen oder ein Zeichen, dessen Unicode zwischen 0 und 31 (einschließlich) liegt.	*diese Zeichen sind transparent*

`[[:alpha:][:digit]]`	Einen Buchstaben oder eine Ziffer	`a`
		`B`
		`6`

`[[:alpha:]][[:digit]]`	Einen Buchstaben gefolgt von einer Ziffer	`a7` `B0`
`[[:alpha:]]{6}`	Genau 6 Buchstaben	`ABCdef` `xXxxXx`

`[[:digit:]]{2,5}`	Mindestens 2, höchstens 5 Ziffern	`01`
		`947`
		`12345`

Regulärer Ausdruck	Welche Zeichenketten (ZK) findet man damit?	Beispiele für solche ZK
`[[:alpha:]]{3}[[:digit:]]{2}`	Genau 3 Buchstaben gefolgt von genau 2 Ziffern	`abc01` `Aaa99`

20.2. Reguläre Ausdrücke und Register

Drei Beispiele für Fortgeschrittene:

Regulärer Ausdruck	Welche Zeichenketten (ZK) findet man damit?	Beispiele für solche ZK

`(.+)\1`	Eine beliebige, nichtleere Zeichenkette (`(.+)`) gefolgt von einer Kopie davon (`\1`)	`XX`
		`abcabc`
		`12XYZ12XYZ`

`([a-z])[0-9]\1`	Eine Ziffer (`[0-9]`), vor ihr ein Kleinbuchstabe (`[a-z]`) und hinter ihr eine Kopi des Kleinbuchstabens (`\1`).	`a7a` `x0x`

`(.)(.)(.)\3\2\1`	Eine beliebige ZK der Länge 3 (`(.)(.)(.)`) gefolgt von ihrem Spiegelbild (`\3\2\1`), d.h. ein Palindrom der Länge 6.	`123321`
		`AbccbA`
		`******`

Erläuterungen zu den drei Beispielen (RE steht für regulärer Ausdruck):

Beispiel 1:

Der RE `.+` passt auf eine beliebige, nicht-leere Zeichenkette.
Der RE `(.+)` passt auf eine beliebige, nicht-leere Zeichenkette und kopiert sie in das Register 1.
Der RE `\1` bezeichnet den Inhalt von Register 1.

Beispiel 2:

Der RE `[a-z]` passt auf eine beliebigen Kleinbuchstaben.
Der RE `([a-z])` passt auf eine beliebigen Kleinbuchstaben und kopiert ihn in das Register 1.
Der RE `\1` bezeichnet den Inhalt von Register 1.

Beispiel 3:

Der RE `.` bezeichnet ein beliebiges Zeichen.
Der erste RE `(.)` bezeichnet ein beliebiges Zeichen und kopiert es in das Register 1.
Der zweite RE `(.)` bezeichnet ein beliebiges Zeichen und kopiert es in das Register 2.
Der dritte RE `(.)` bezeichnet ein beliebiges Zeichen und kopiert es in das Register 3.
Der RE `\3` bezeichnet den Inhalt von Register 3.
Der RE `\2` bezeichnet den Inhalt von Register 2.
Der RE `\1` bezeichnet den Inhalt von Register 1.

Regel 1: Wenn man einen regulären Ausdruck `re` in runde Klammern einschließt, wird die Zeichenkette, auf die `re` paßt und die man aufgrund von `re` gefunden hat, in das nächste freie Register kopiert (und dieses Register ist damit belegt und nicht mehr frei).

Regel 2: Insgesamt gibt es 9 solche Register.

Regel 3: Die Namen `\1`, `\2`, ..., `\9` bezeichnen die Inhalte dieser Register.

20.3. Durch einen regulären Ausdruck ersetzen

Das Fenster für den Befehl Ersetzen... kann man durch Menü Suchen, Ersetzen... oder F8 öffnen. Wenn man in diesem Fenster vor Regulärer Ausdruck ein Häkchen gemacht hat, wird (nicht nur der Inhalt des Eingabefeldes Suchen nach:, sonder auch) der Inhalt des Eingabefeldes Ersetzen durch: als *regulärer Ausdruck* (RE) interpretiert. Hier ein paar Beispiele, was man dadurch erreichen kann:

Beispiel 1: Pluszeichen entfernen, aber nur solche, die *vor einer Ganzzahl* stehen:

Suchen nach: `\+([0-9]+)`
Ersetzen durch: `\1`

`\+` steht für ein normales Pluszeichen (welches *nicht* 1-oder-mehr-von-dem-was-davor-steht bedeutet). Der RE `\+([0-9]+)` findet ein Pluszeichen vor einer (nicht-leeren) Ziffernfolge. Weil (nur) die Ziffernfolge in runden Klammern steht, wird (nur) sie in das Register 1 geschrieben.

Der RE `\1` bezeichnet den Inhalt von Register 1 (d.h. die gefundene Ziffernfolge ohne Pluszeichen).

Beispiel 2: Ganzzahlen mit einem Pluszeichen versehen (aber nur solche, die noch kein Vorzeichen haben)

Suchen nach: `([^+-0-9])([0-9]+)`
Ersetzen durch: `\1\+\2`

Der RE `([^+-0-9])` findet irgendein Zeichen, welches kein Vorzeichen und keine Ziffer ist, und schreibt es in das Register 1. Der Teilausdruck `0-9` ist nötig, weil sonst z.B. `-123` umgewandelt wird in `-1+23`.

Der RE `([0-9]+)` findet eine (nicht-leere) Ziffernfolge und schreibt sie in das Register 2.

Der RE `\1\+\2` bezeichnet den Inhalt von Register 1, gefolgt von einem Pluszeichen, gefolgt vom Inhalt von Register 2 (d.h. gefolgt von der Ziffernfolge).

Hinweis: Der Suchen-nach-RE ist noch nicht perfekt. Er findet keine Zahlen, die ganz am Anfang einer Zeile stehen (weil *vor* diesen Zahlen nichts oder nur ein Zeilenwechsel steht, der nicht als normales Zeichen zählt). Können Sie den Ausdruck perfektionieren (z.B. mit einem Oder-Strich `|` und ...)?

Beispiel 3: Hinter jedem öffnenden HTML-Tag einen entsprechenden schließenden Tag einfügen

Suchen nach: `<([^/]*)>`
Ersetzen durch: `<\1></\1>`

Der RE `<([^/]*)>` findet einen beliebigen öffnenden HTML-Tag und schreibt seinen Namen (ohne die spitzen Klammern drumrum) in das Register 1.

Der RE `<\1></\1>` beschreibt den gefundenen Tag (`<\1>`) gefolgt von einem entsprechenden schließenden Tag (`</\1>`).

Beispiel 4: Zwei hintereinanderstehende Zeichenketten miteinander vertauschen

Suchen nach: `([0-9]+)([A-Za-z]+)`
Ersetzen durch: `\2\1`

Der RE `([0-9]+)` findet eine beliebige, nicht leere *Ziffernfolge* und schreibt sie in das Register 1.

Der RE `([A-Za-z]+)` findet eine beliebige, nicht-leere *Buchstabenfolge* und schreibt sie in das Register 2.

Der RE `\2\1` bezeichnet die gefundene *Buchstabenfolge* gefolgt von der *Ziffernfolge*.

Beispiel 5: `Hallo` durch `Hello` ersetzen, aber nur in `//`-Kommentaren

Suchen nach: `(//.*)Hallo`
Ersetzen durch: `\1Hello`

Der RE `(//.*)Hallo` findet einen `//`-Kommentar, in dem `Hallo` vorkommt und schreibt ihn (ohne das `Hallo`) in das Register 1.

Der RE `\1Hello` bezeichnet einen ganz ähnlichen Kommentar wie den gefundenen, aber mit `Hello` statt `Hallo`.

Hinweis: Auch hier ist der Suchen-nach-RE noch nicht perfekt. Wenn `Hallo` in einem Zeilenende-Kommentar *mehrmals* vorkommt, wird nur *das letzte Vorkommen* durch `Hello` ersetzt. Können Sie den Ausdruck perfektionieren? Dem Autor dieses Papiers ist keine Lösung für dieses Problem bekannt und er würde sich über einen Hinweis auf eine Lösung besonders freuen.

Beispiel 6: Zeilen-Nummern in eine Textdatei schreiben

Suchen nach: `^`
Ersetzen durch: `\i`

Der RE `^` findet den Anfang einer Zeile.

Der RE `\i` hat eine ganz spezielle Bedeutung. Er wird durch die jeweils nächste Zahl aus der Folge `1`, `2`, `3`, ... ersetzt. Hinter dem `\i` sollte man noch ein Leerzeichen und/oder einen Doppelpunkt `:` oder etwas ähnliches einfügen, damit die Zeilen-Nummern nicht unmittelbar vor dem Text der Zeilen stehen.

Anmerkung: Dieses Beispiel scheint nur zu funktionieren, wenn man den Knopf Alle ersetzen benutzt (statt der Knöpfe Ersetzen oder Nächstes Ersetzen). Vermutlich ein Fehler im TextPad.

Beispiel 6: Eine Nummerierung, die durcheinandergeraten ist, erneuern.

Angenommen, Sie haben in einem Dokument Abbildungen nummeriert. Die Nummern sehen etwa so aus: `Abb. 1`, `Abb. 2`, `Abb. 3`, ... Jetzt sind diese Nummern durcheinandergeraten: Ein paar Nummern fehlen, andere stehen nicht in der richtigen Reihenfolge. Sie wollen die Abbildungen neu nummerieren, mit 100 beginnend und in 10-er-Schritten (`Abb. 100`, `Abb. 110`, `Abb. 120`, ...). Das geht etwa so:

Suchen nach: `Abb. [0-9]*`
Ersetzen durch: `Abb. \i(100, 10)`

Der RE `Abb. [0-9]*` findet eine Abbildungs-Nummer (auch solche, bei denen nach `Abb.` *keine* Zahl steht).

Der RE `\i(100, 10)` wird durch die jeweils nächste Zahl aus der Folge `100`, `110`, `120`, ... ersetzt.

Anmerkung: Auch dieses Beispiel funktioniert nur dann richtig, wenn man den Knopf Alle ersetzen benutzt. Benutzt man statt dessen die Knöpfe Ersetzen oder Nächstes Ersetzen wird nur die Anfangszahl eingesetzt (d.h. all Abbildungen haben dann die Nummer 100).

Weitere Informationen zu diesem Thema findet man in der TextPad-Hilfe unter dem Stichwort reguläre Ausdrücke.

20.4. Reguläre Ausdrücke vernünftig editieren

Selbst auf einem 30-Zoll-Plasmabildschirm sind die Fenster der Befehle Suchen nach... und Ersetzen... ziemlich klein und die Eingabfelder Suchen nach: und Ersetzen durch: in diesen Fenstern sind noch viel kleiner. Statt einen regulären Ausdruck direkt in einem dieser unübersichtlichen Eingabefelder zu editieren, kann man auch ein neues Dokument anlegen (z.B. mit dem Tastenkürzel STRG-N), den regulären Ausdruck in einer Zeile dieses Dokuments editieren und den Ausdruck erst dann (über die Ablage) in das entsprechende Eingabefeld kopieren (mit STRG-C und STRG-V).

Reguläre Ausdrücke, die man später wiederverwenden will, kann man auch in eine Text-Datei (deren Namen mit der Erweiterung `.txt` endet) schreiben. Und wenn man kleine Schriften nur mit Mühe lesen kann, sollte man Text-Dateien mit einem etwas größeren Font anzeigen lassen, etwa so:

Menü Konfiguration, Einstellungen, +Dokumentenklassen, +Text, Schriftart
Unter Schriftgrad eine geeignete Schriftgröße auswählen.

21. Im Blockmodus Textblöcke löschen, kopieren und einfügen

Der TextPad kennt zwei Modi (oder: Betriebsarten): Den *Normalmodus* und den *Blockmodus*. Im Blockmodus kann man in einem Text *rechteckige Texblöcke* auswählen und dann *löschen* oder ausschneiden oder in die Ablage *kopieren* oder aus der Ablage an beliebige Stellen des Textes *einfügen*.

21.1. Ein (hoffentlich) motivierendes Beispiel

Angenommen, Sie wollen in einem Java-Programm (z.B. in einem Testprogramm) die Ergebnisse bestimmer Funktionsaufrufe "gut kommentiert" ausgeben, etwa mit Befehlen wie den folgenden:

```
1    printf("Math.sin(0.5 * Math.PI): %6.2f%n", Math.sin(0.5 * Math.PI));
2    printf("Math.cos(0.5 * Math.PI): %6.2f%n", Math.cos(0.5 * Math.PI));
3    printf("Math.sin(1.0 * Math.PI): %6.2f%n", Math.sin(1.0 * Math.PI));
4    printf("Math.cos(1.0 * Math.PI): %6.2f%n", Math.cos(1.0 * Math.PI));
5    printf("Math.sin(1.5 * Math.PI): %6.2f%n", Math.sin(1.5 * Math.PI));
6    printf("Math.cos(1.5 * Math.PI): %6.2f%n", Math.cos(1.5 * Math.PI));
```

In diesem Text wurden zwei rechteckige Blöcke fett hervorgehoben um leichter erkennbar zu machen, dass sie *genau übereinstimmen*. Wenn man diese Zeilen einfach so abtippt, würde man also an vielen Stellen zweimal das Gleiche tippen.

Mit dem TextPad kann man statt dessen so vorgehen: Man tippt die 6 Zeilen ein, lässt aber auf jeder Zeile die Daten, die zum rechten Block gehören (und die man gerade schon mal eingegeben hat) weg. Wenn man damit fertig ist, kopiert man den linken Block an die Stelle des rechten Blocks. Das ist (nur) im Blockmodus möglich und geht, nach ein bisschen Übung, ziemlich einfach und schnell.

21.2. Blockmodus-Grundlagen

In den *Blockmodus* versetzen kann man den TextPad auf zwei unterschiedliche Weisen:

1. Flüchtiger Blockmodus: Man drückt auf die Alt-Taste und hält sie gedrückt. Sobald man die Taste losläßt kehrt der TextPad aus dem Blockmodus wieder in den Normalmodus zurück.

2. Fester Blockmodus: Man gibt den Befehl Menü Konfiguration, Blockauswahl-Modus (oder STRG-Q B) ein. In diesem Fall kehrt der TextPad erst dann wieder in den Normalmodus zurück, wenn man den gleichen Befehl noch einmal eingibt.

Während der TextPad sich im Blockmodus befindet, kann man auf zwei unterschiedliche Weisen rechteckige Textblöcke *auswählen* (oder: *markieren*):

1. Auswählen mit der Maus: Bei gedrückter linker Maustaste den gewünschten Block von links oben nach rechts unten (oder umgekehrt von rechts unten nach links oben) "überstreichen".

2. Auswählen mit Tasten: Den Cursor links oben "vor das *erste* Zeichen des gewünschten Blocks" (oder umgekehrt: unten rechts "hinter das *letzte* Zeichen des gewünschten Blocks") positionieren. Spätestens jetzt den Blockmodus einschalten und mit den Cursortasten ←, ↑, → und ↓ den gewünschten Block auswählen.

21.3. Textblöcke bearbeiten: Ein paar Beispiele

Einen ausgewälten Block kann man genau wie eine ander Auswahl bearbeiten: Man kann ihn

löschen (mit der Entf-Taste oder der Back-Taste)
ausschneiden (mit STRG-X)
kopieren (mit STRG-C)
einfügen (mit STRG-V) .

Die folgenden Beispiele sind ein Versuch, die Wirkung wichtiger Blockbefehle zu veranschaulichen.

Beispiel-01: Textblöcke löschen

```
++++++++++++++++++++        ++++++++++++++++++++        ++++++++++++++++++++
++AAA++++++++++++++++        ++AAA++++++++++++++++        ++++++++++++++++++
++AAA++++BBBBBB+++++        ++AAA+++++++++              ++++++++++
++AAA++++BBBBBB+++++        ++AAA+++++++++              ++++++++++
++AAA++++BBBBBB+++++        ++AAA+++++++++              ++++++++++
+++++++++BBBBBB+++++        +++++++++++++++             +++++++++++++++
+++++++++BBBBBB+++++        +++++++++++++++             +++++++++++++++
```

Text 1: Mit A- und B-Block Text 2: Ohne den B-Block ... Text 3: ... und ohne A-Block

Der Text 1 besteht aus 7 gleich langen Zeilen, die zu einem großen Teil Pluszeichen enthalten. Nur zwei rechteckig Blöcke enthalten anstelle der Pluszeichen As bzw. Bs.

Wenn man den B-Block (auswählt und) *löscht*, erhält man den Text 2. Wenn man darin den A-Block (auswählt und) *löscht* erhält man Text 3. Der Text 1 hat einen glatten rechten Rand. Der Text 3 hat rechts einen Flatterrand.

Beispiel-02: Textblöcke einfügen

```
++++++++++          ++++++++++          ++++++++++
++CC+++++++         +CCCCC++++++++      +CCCCC++++++++
++++++++++          ++CCC+++++++++      ++CCC+++++++++
+++++++DD++         ++CCC+++DD++++      ++CCC+++DDDD++++
++++++++++          ++++++++++          +++++++++DD++
++++++++++          ++++++++++          +++++++++DD++
++++++++++          ++++++++++          +++++++++DD++
```

Text 4: Ausgangstext Text 5: Mit einem C-Block Text 6: Mit C- und D-Block

Wenn man im Text 4 den Cursor zwischen die beiden Ces positioniert und dann einen3x3-Block von Ces einfügt, erhät man Text 5. Wenn man darin den Cursor zwischen die beiden Des positioniert und einen 2x4-Block mit Des einfügt, erhält man Text 6. Der Text 4 hat einen glatten rechten Rand. Der Text 6 hat rechts einen Flatterrand.

Beispiel-03: Rechteckige und andere Blöcke

```
    +    E          +E          +    F    +F
   ++    E         ++E         ++    F    ++F
  +++    E        +++E        +++    F    +++F
 ++++    E       ++++E       ++++    F    ++++F
+++++    E      +++++E      +++++    F    +++++F
```

Text 7 Text 8 Text 9 Text 10 Text 11 Text 12

Wenn man den Text 7 vor den Text 8 kopiert, erhält man Text 9. Offenbar ist Text 7 ein rechteckiger Block, der oberhalb der Diagonalen Leerzeichen enthält. Das ist kaum überraschend.

Wenn man den Text 10 vor den Text 11 kopiert erhält man Text 12. Daraus kann man schliessen: Text 10 ist *kein rechteckiger* Block, der oberhalb der Diagonalen Leerzeichen enthält, sondern ein *dreieckiger* Block, der oben kürzer ist als unten. Das ist möglicherweise überraschend (weil die Markierung eines Blocks *immer rechteckig* aussieht, auch wenn der Block in Wirklichkeit dreieckig ist oder einen rechten Flatterrand hat).

Wenn man nur Text 10 und Text 11 sieht, kann man nicht mit Sicherheit voraussagen, wie Text 12 aussehen wird. Denn man könnte den Text 10 oberhalb der Diagonalen auch mit Leerzeichen zu einem Rechteck ergänzen oder "flattern lassen". Das würde sein Aussehen nicht ändern, wohl aber das von Text 12 (dort würden die eFs dann untereinander stehen oder ebenfalls flattern, statt eine regelmäßige Treppe zu bilden).

Wenn das Ergebnis eines Block-Einfüge-Befehls nicht wie erwartet aussieht und einem falsch vorkommt, sollte man sich an dieses Beispiel erinnern oder es noch einmal ansehen. In solchen Fällen ist es meist auch hilfreich, die (normalerweise unsichtbaren) *Leerzeichen* und *Zeilenwechsel* sichtbar zu machen (z.B. indem man STRG-Q I eingibt oder auf den entsprechenden kleinen Knopf mit dem Paragraphen-Symbol ¶ darauf klickt).

Merke: Im Blockmodus kann man *rechteckige Textblöcke* bearbeiten, aber einige dieser Blöcke sind dreieckiger oder flatteriger als andere :-).

22. Anhang A: Eine Datei mit TextPad-Bausteinen (eine .tcl-Datei)

Wie Sie die folgende Datei namens `java.tcl` in Ihre TextPad-Installation einbauen können, wird oben als Tipp 7. Textbausteine (eine .tcl-Datei) übertragen beschrieben.

```
!TCL=2, Dateiname java.tcl, von Ulrich Grude, Oktober 2010
!TITLE=Java
!SORT=N
!CHARSET=ANSI

!TEXT=Mehrzeilenkommentar
/* ------------------------------------------------------------------------

------------------------------------------------------------------------ */

!

!TEXT=Minuszeile
// ------------------------------------------------------------------------

!
!TEXT=Pluszeile
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

!

!TEXT=Klasse (mit main, pln)
// Datei XX.java
/* ------------------------------------------------------------------------
Dieses Programm XX leistet Folgendes:
------------------------------------------------------------------------ */

class XX {
   // ---------------------------------------------------------------------
   static public void main(String[] sonja) {
      pln("XX: Jetzt geht es los!");
      pln("XX: Das war's erstmal!");
   } // main
   // ---------------------------------------------------------------------
   // Eine Methode mit einem kurzen Namen:
   static void pln(Object ob) {System.out.println(ob);}
   // ---------------------------------------------------------------------
} // class XX

!

!TEXT=Klasse (mit main, printf)
// Datei XX.java
/* ------------------------------------------------------------------------
Dieses Programm XX leistet Folgendes:
------------------------------------------------------------------------ */

class XX {
   // ---------------------------------------------------------------------
   static public void main(String[] sonja) {
      printf("XX: Jetzt geht es los!%n");
      printf("XX: Das war's erstmal!%n");
   } // main
   // ---------------------------------------------------------------------
   // Eine Methode mit einem kurzen Namen:
   static void printf(String f, Object... v) {System.out.printf(f, v);}
   // ---------------------------------------------------------------------
} // class XX

!

!TEXT=Klasse (UebFF, ohne main)
```

```
// Datei UebFF.java
/* --------------------------------------------------------------------------
Eine Klasse, die eine einzige Klassenmethode (genauer: eine Funktion)
enthaelt.
-------------------------------------------------------------------------- */

public class UebFF {
   // ------------------------------------------------------------------------
   static public TT FF(PP) {

   } // FF
   // ------------------------------------------------------------------------
} // class UebFF

!

!TEXT=Klasse (UebFFJut, JUnit)
// Datei UebFFJut.java
/* --------------------------------------------------------------------------
Ein JUnit-Programm zum Testen der Funktion FF in der Klasse UebFF
(JUnit Version 3.8)
-------------------------------------------------------------------------- */
import junit.framework.Test;      // Eine Schnittstelle
import junit.framework.TestCase;  // Eine Test-Klasse
import junit.framework.TestSuite; // Eine Test-Klasse

public class UebFFJut extends TestCase {
   // ------------------------------------------------------------------------
   static public Test suite() {
      return new TestSuite(UebFFJut.class);
   } // suite
   // ------------------------------------------------------------------------
   public void test01() {

   } // test01
   // ------------------------------------------------------------------------
   public void test02() {

   } // test02
   // ------------------------------------------------------------------------
   public void test03() {

   } // test03
   // ------------------------------------------------------------------------
   public void test04() {

   } // test04
   // ------------------------------------------------------------------------
   public void test05() {

   } // test05
   // ------------------------------------------------------------------------
   static public void main(String[] _) {
      junit.awtui.TestRunner.run(UebFFJut.class);
   } // main
   // ------------------------------------------------------------------------
} // class UebFFJut

!

!TEXT=main (mit pln)
   // ------------------------------------------------------------------------
   static public void main(String[] sonja) {
      pln("XX: Jetzt geht es los!");
      pln("XX: Das war's erstmal!");
   } // main
   // ------------------------------------------------------------------------
   // Eine Methode mit einem kurzen Namen:
```

```
   static void pln(Object ob) {System.out.println(ob);}
   // ------------------------------------------------------------------------

!

!TEXT=main (mit printf)
   // ------------------------------------------------------------------------
   static public void main(String[] sonja) {
      printf("XX: Jetzt geht es los!%n");
      printf("XX: Das war's erstmal!%n");
   } // main
   // ------------------------------------------------------------------------
   // Eine Methode mit einem kurzen Namen:
   static void printf(String f, Object... v) {System.out.printf(f, v);}
   // ------------------------------------------------------------------------

!


!TEXT=stat-pub-void-Methode^
   static public void \^ (PP) \^ {
   } //

!

!TEXT=stat-pub-XXX-Methode
   static public XXX (PP) {
   } //

!

!TEXT=if-else
      if (BB) {
         AA
      } else {
         AA
      } // if

!

!TEXT=if-else-if
      if (BB) {
         AA
      } else if (BB) {
         AA
      } else if (BB) {
         AA
      } else {
         AA
      } // if

!

!TEXT=for (mit int i)
      for (int i=0; i<XX; i++) {
         AA
      } // for

!

!TEXT=for (ohne i)
      for (XX;YY;ZZ) {
         AA
      } // for

!

!TEXT=while (true)
```

```
      while (true) {
         AA
         if(BB) break;
         AA
      } // while

!

!TEXT=do-while
      do {
         AA
      } while (BB);

!

!TEXT=pln (1 Abkürzung)
   // Eine Methode mit einem kurzen Namen:
   static void pln(Object ob) {System.out.println(ob);}

!

!TEXT=pln (3 Abkürzungen)
   // Mehrere Methoden mit kurzen Namen:
   static void pln(Object ob) {System.out.println(ob);}
   static void p  (Object ob) {System.out.print  (ob);}
   static void pln()          {System.out.println();  }

!

!TEXT=printf (1 Abkürzung)
   // Eine Methode mit einem kurzen Namen:
   static void printf(String f, Object... v) {System.out.printf(f, v);}

!
```

Die vorige Zeile (die nur ein Ausrufezeichen `!` enthält) ist *die letzte Zeile* der Datei `java.tcl`