

Inhaltsverzeichnis

Tips zu Eclipse.....	1
1. Eclipse-Grundbegriffe: Ansicht (view), Editor, Perspektive.....	1
2. Ein Hallo-Programm entwickeln und ausführen.....	2
3. Ein paar Einstellungen ändern.....	3
3.1. Zeilen-Nummern und Druckgrenze (print margin) etc. anzeigen lassen:.....	3
3.2. Die Formatierung von Java Quelldateien festlegen.....	3
4. Tastenkürzel (key bindings).....	4
4.1. Vordefinierte Tastenkürzel.....	4
4.2. Eigene Tastenkürzel definieren.....	5
5. Schablonen (templates).....	5
5.1. Schablonen anwenden.....	5
5.2. Schablonen definieren.....	6
6. Programme umstrukturieren (refactoring).....	7
7. Blitzänderungen (quick fixes).....	7
8. Das Help-Menü.....	9
9. Eine Java-Quelldatei in ein Eclipse-Projekt importieren.....	9
10. Ein Java-Archiv (d.h. eine .jar-Datei) zu einem Projekt hinzufügen.....	9
11. Ein Eclipse-Projekt auf einen anderen Rechner übertragen.....	10
12. Eine JUnit-Testklasse compilieren und ausführen lassen.....	10
13. Quelldateien von Standardklassen anzeigen lassen.....	11
14. Die XML-Beispiele in ein Eclipse-Projekt übernehmen.....	12
15. Pfadnamen und URIs.....	13
16. Suchen und Ersetzen mit regulären Ausdrücken und Fangmustern.....	14
17. Mehr über Eclipse lernen.....	16

Tips zu Eclipse

Dies sind die ersten Zeilen einer geplanten 6-bändigen, vollständigen Beschreibung der grafischen Benutzeroberfläche von Eclipse :-).

Eclipse ist ein Programm, dessen Leistungsumfang nicht starr festgelegt ist, sondern durch das Hinzufügen von so genannten **Plugins** erweitert und modifiziert werden kann. Als Basis enthält Eclipse eine Entwicklungsumgebung für Java-Programme und eine Entwicklungsumgebung für Eclipse-Plugins. Eclipse läuft auf verschiedenen Plattformen, z.B. unter Linux, Mac OS und Windows.

Im Internet werden zahlreiche Kombinationen aus Eclipse und bestimmten Plugins angeboten. Für die Entwicklung von Java-Programmen ist zur Zeit (Ende 2009) die Kombination namens **Galileo** besonders empfehlenswert. Die Tips in diesem Papier beziehen sich auf diese Kombination.

Wenn man Eclipse (zum ersten Mal) startet, wird man nach einem **workspace-Verzeichnis** gefragt, in dem das Eclipse-System Unterverzeichnisse anlegen und Dateien ablegen kann. Im SWE-Labor sollte man ein Verzeichnis auf der Z:-Partition angeben, z.B. Z:\JavaEWS (EWS soll hier an Eclipse Work Space erinnern).

1. Eclipse-Grundbegriffe: Ansicht (view), Editor, Perspektive

Eine **Ansicht** (engl. view) ist ein *Fenster* (innerhalb des umfassenden Eclipse-Fensters), in dem Informationen *angezeigt* werden, aber vom Benutzer *nicht verändert* werden können.

Ein **Editor** (engl. editor) ist ein *Fenster* (innerhalb des umfassenden Eclipse-Fensters) in dem Informationen *angezeigt* werden und vom Benutzer *verändert* werden können.

Achtung: Mit einem Editor ist in Eclipse ein *Fenster* gemeint, hinter dem ein Editor-Programm aktiv ist, nicht das Editor-Programm selbst.

Eine **Perspektive** (engl. perspective) ist eine bestimmte *Kombination* von *Ansichten* und *Editoren*.

Die Perspektive namens **Java** ist besonders gut zum Entwickeln von Java-Programmen geeignet, die Perspektive namens **Plug-in Development** zum Entwickeln von Eclipse-Plugins, die Perspektive namens **JavaScript** zum Entwickeln von ... etc.

Im umfassenden Eclipse-Fenster wird jeweils *eine* Perspektive gezeigt. Der *Name* dieser aktuellen Perspektive steht im Rahmen des Eclipse-Fensters ganz oben links (zwischen dem Eclipse-Logo und dem ersten Bindestrich).

Wechseln kann man die aktuelle Perspektive mit Knöpfen, die ganz rechts oben im Eclipse-Fenster angebracht sind, oder mit dem Menübefehl:

Window / Change Perspective

Eine Ansicht (engl. a view) kann man *verschwinden lassen*, indem man auf das Kreuzchen hinter ihrem Namen klickt. Mit dem folgenden Menübefehl kann man sie dann *wieder sichtbar machen*:

Window / Show View

2. Ein Hallo-Programm entwickeln und ausführen

Schritt 1: Stellen Sie sicher, dass die Perspektive namens **Java** gezeigt wird (siehe vorigen Abschnitt).

Schritt 2: Erstellen Sie ein Java-Projekt mit dem Befehl: **File / new / Java Project**

Im Fenster **Create a Java Project** braucht man nur einen *Namen* für das Projekt festzulegen, z.B. `Projekt01`. Danach sollte man auf **Next** klicken.

Im Fenster **Java Settings** auf **Finish** klicken.

Das neue `Projekt01` sollte jetzt in der **Package Explorer** - Ansicht (links im Eclipse-Fenster) angezeigt werden. Es sieht aus wie ein Ordner (mit einem +-Kästchen davor).

Schritt 3: Erstellen Sie innerhalb des Projekts eine Klasse:

Im **Package Explorer** das Projekt `Projekt01` auswählen.

Dann: **File / New / Class**

Im Fenster **Java Class**

hinter **Package** einen Paketnamen eintragen, z.B. `paket01`

hinter **Name** einen Klassennamen eintragen, z.B. `HauptKlasse`

vor `public void main(String[] args)` ein Häkchen machen und auf **Finish** klicken.

Dadurch sollte eine Quelldatei namens `HauptKlasse.java` erzeugt und in einem Editor (in der Mitte des Eclipse-Fensters) angezeigt werden. Ergänzen Sie die `main`-Methode dieser Klasse um einen Befehl wie etwa `System.out.println("Hallo!");`.

Schritt 4: Lassen Sie das Programm ausführen mit dem Befehl: **Run / Run as / Java Application**

Die Ausgabe des Programms sollte in der **Console**-Ansicht (im Eclipse-Fenster ganz unten) erscheinen.

Anmerkung: Während Sie Java-Quellcode eingeben, wird dieser Code *automatisch* und *inkrementell* ("Stückchen für Stückchen") kompiliert. In den meisten Fällen ist der inkrementelle Java-Compiler fertig, wenn Sie mit dem Eintippen des Quellcodes fertig sind.

3. Ein paar Einstellungen ändern

Konvention:

+ **Editors** soll bedeuten: Auf das Pluszeichen + vor **Editors** klicken.

Editors soll bedeuten: Auf das Wort **Editors** klicken (nicht auf das Pluszeichen davor)

3.1. Zeilen-Nummern und Druckgrenze (print margin) etc. anzeigen lassen:

Window / Preferences / + General / + Editors / Texteditors

Im Fenster **Text Editors** vor **Show line numbers** ein Häkchen machen.

Die Zeilen-Nummern werden dann nicht nur auf dem Bildschirm angezeigt, sondern auch gedruckt (wenn man eine Quelldatei mit **File / print** ausdrucken lässt).

Nach **Print margin column**: die Zahl 75 eingeben. Dadurch erscheint in allen Editoren ("Editor-Fenstern") eine Druckgrenze, d.h. ein senkrechter Strich zwischen Spalte 75 und 76. Wenn man beim Eingeben von Text diesen Strich nicht "übertritt", passen die Zeilen auch beim **Ausdrucken** auf eine Zeile (und müssen nicht vom Drucker umgebrochen werden). Im SWE-Labor kann man statt 75 sogar 78 eingeben.

Vor **Insert spaces for tabs** ein Häkchen machen. Dadurch wird sichergestellt, dass (nicht nur der Eclipse-Java-Texteditor sondern) *jeder Editor* die Einrückung des Quelltextes richtig darstellt.

Nach **Displayed tab width**: die Zahl 3 eintragen.

Achtung: Diese Einstellung hat Auswirkungen z.B. auf `.txt`-Dateien. Dagegen wird die Einrücktiefe von `.java`-Dateien durch eine spezielle **Formatter**-Komponente und ein **Profil** festgelegt (siehe nächsten Abschnitt).

3.2. Die Formatierung von Java Quelldateien festlegen

Wie Java-Quelldateien (`.java`-Dateien) vom Eclipse-Java-Editor formatiert werden sollen, wird durch so genannte **Profile** (engl. profiles) beschrieben. Ein paar solcher Profile (namens **Eclipse (built-in)**, **Java Conventions (built-in)**, ...) sind vorgegeben. Diese built-in-Profil kann der Benutzer *ansehen* und *aktivieren*, aber *nicht verändern*. Er kann aber weitere Profile erstellen, indem er ein schon vorhandenes Profil kopiert und abändert, etwa so:

Window / Preferences / + Java / + Code Style / Formatter / New

Im Fenster **New Formatter** unter **Profile name**: einen Namen für das neue Profil eintragen (z.B. `meinJavaProfil01`), unter **Initialize settings with the following profile** das zu kopierende Profil auswählen und auf **OK** klicken.

Normalerweise wird dadurch automatisch der Profil-Editor geöffnet, damit man das neue Profil seinen eigenen Wünschen anpassen kann. Der Profil-Editor bietet einem zahlreiche übereinander liegende Fenster-mit-Reitern zur Auswahl an (**Indentation**, **Braces**, **White Space**, **Blank Lines**, ...). Rechts daneben werden in einem **Preview**-Fenster die Auswirkungen der aktuellen Einstellungen auf bestimmte Code-Beispiele sofort angezeigt (was sehr angenehm und nützlich ist!).

Im Fenster mit dem Reiter **Indentation** kann man die Einrücktiefe (**Indentation size**:) und die Breite eines-Tab-Zeichens (**Tab size**:) beide auf 3 stellen. Als **Tab policy**: sollte man **Spaces only** auswählen.

Sehr empfehlenswert ist auch ein Häkchen vor **Align fields in columns**. Prüfen Sie im **Preview**-Fenster, ob Ihnen die Wirkung gefällt.

Im Fenster mit dem Reiter **New Lines** kann man alle Häkchen in der obersten Zeilengruppe (von **in empty class body** bis **at end of file**) entfernen. Dadurch wird der Quelltext etwas kompakter formatiert und man kann auf dem Bildschirm mehr davon sehen, ohne zu blättern (engl. scroll).

4. Tastenkürzel (key bindings)

Für viele Eclipse-Befehle gibt es Tastenkürzel und der Benutzer kann vorhandene Kürzel ändern oder löschen und neue Kürzel hinzufügen.

4.1. Vordefinierte Tastenkürzel

Es folgt hier eine kleine Auswahl besonders interessanter, vordefinierter Tastenkürzel:

Tastenkürzel	Wirkung	Befehlsname
Ctrl+Shift+L	Zeigt eine Liste der momentan verfügbaren Tastenkürzel an.	Key Assist
Ctrl+/ /	Ausgewählte Zeilen auskommentieren (mit //) bzw. Auskommentierung wieder aufheben.	Toggle Comment
Ctrl+Shift+M	Wenn der Cursor im Namen einer Klasse steht wird ein passender <code>import</code> -Befehl eingefügt (falls nötig und möglich).	Add Import
Ctrl+Shift+O	Organisiert die <code>import</code> -Befehle der aktuellen Datei (entfernt unnötige, fügt fehlende hinzu, sortiert sie etc.).	Organize Imports
Ctrl+F	Öffnet den Find/Replace-Dialog (Suchen und Ersetzen).	Find / Replace
Ctrl+K	Sucht den ausgewählten Text vorwärts.	Find Next
Ctrl+Shift+K	Sucht den ausgewählten Text rückwärts.	Find Previous
Ctrl+J	Startet eine <i>inkrementelle Suche</i> vorwärts (während man den Suchtext eintippt, wird schon gesucht).	Incremental Find Next
Ctrl+Shift+J	Startet eine <i>inkrementelle Suche</i> rückwärts (während man den Suchtext eintippt, wird schon gesucht).	Incremental Find Previous
Ctrl+Space	Öffnet einen "Was-darf-man-hier-einfügen?"-Dialog.	Content Assist
Ctrl+Shift+Space	Wenn der Cursor zwischen den runden Klammern () eines Methodenaufrufs steht, werden die Typen der Parameter angezeigt.	Parameter Hints
Alt-C	Schneidet die aktuelle Zeile aus. Wurde vom Benutzer definiert (siehe oben Abschnitt 4.).	Cut Line
Ctrl+Alt+↓	Fügt eine Kopie der aktuellen Zeile (<i>nach</i> der aktuellen Zeile) ein.	Editing Text
Ctrl+F11	Führt das aktuelle Programm aus.	Run

In die freien Zeilen können Sie Ihre persönlichen Favoriten eintragen. Eine umfangreiche *Liste von populären Tastenkürzeln* wird einem angezeigt, wenn man den Menüpunkt **Help / Help Content** auswählt und dort nach `List of [key bindings]` sucht.

4.2. Eigene Tastenkürzel definieren

Leider gibt es keinen Eclipse-Befehl, mit dem man die aktuelle Zeile (die, in der der Cursor gerade steht) löschen kann (ohne sie vorher umständlich auswählen zu müssen). Immerhin gibt es einen Befehl namens `Cut Line`, der die aktuelle Zeile *ausschneidet* (d.h. in die Ablage kopiert und aus dem Text entfernt). Als Beispiel soll für diesen Befehl das Tastenkürzel `Alt+C` definiert werden:

Window / Preferences / + General / Keys

Im großen Auswahlfenster wählt man die Zeile, die mit dem Text `Cut Line` beginnt (in der Spalte **Command**). Dann gibt man das neue Tastenkürzel `Alt+C` in das Textfeld nach **Binding**: ein.

Soll das neue Tastenkürzel nur beim Editieren von Java-Quellprogrammen wirksam sein, kann man im **When**-Fenster die Alternative **Editing Java Source** auswählen. Wählt man statt dessen die Alternative **In Windows**, ist das neue Tastenkürzel *immer* wirksam.

Hinweis: Wenn man mit dem Befehl `Cut Line` (oder mit dem neu definierten Tastenkürzel `Alt+C`) *mehrere* Zeilen unmittelbar nacheinander ausschneidet, stehen danach *alle* diese Zeilen in der Ablage (und nicht nur die *letzte*).

5. Schablonen (templates)

Eine Schablone (engl. a template) ist ein mit einem Namen versehener Text (z.B. ein Java-Befehl oder ein häufig verwendeter Kommentar wie `// Copyright by Angelika Meyerbeer` oder ...). Eine Schablone kann aus wenigen Zeichen, einer ganzen Zeile oder aus mehreren Zeilen bestehen. Eine Schablone kann Variablen enthalten, die beim Anwenden der Schablone (siehe nächsten Abschnitt) durch Texte ersetzt werden oder andere Effekte bewirken.

Beispiel-01: Die Schablone namens `main` ist wie folgt definiert:

```
public static void main(String[] args) {
    ${cursor}
}
```

Beispiel-02: Die Schablone namens `sysout` ist wie folgt definiert:

```
System.out.println("${word_selection}");${cursor}
```

Das **Templates-Fenster**, welches *alle* zur Zeit vorhandenen Schablonen-Definitionen anzeigt, können Sie mit dem folgenden Befehl öffnen:

Window / Preferences / + Java / + Editor / Templates

In diesem **Templates-Fenster** können Sie bereits vorhandene Definitionen ansehen, verändern und löschen oder neue Definitionen einfügen.

5.1. Schablonen anwenden

Beispiel-01: Die Schablone `main` anwenden

Positionieren Sie den Cursor an einer Stelle, an der Sie eine `main`-Methode einfügen *möchten und dürfen* (also *innerhalb* einer Klassenvereinbarung, aber *ausserhalb* aller Methodenvereinbarungen).

Geben Sie `main` ein und dann `Ctrl+Space`.

Ein Fenster mit mehreren Alternativen darin wird geöffnet. Wählen Sie die Alternative `main - main method` (mit den Tasten `↓` bzw. `↑` und `Ctrl` oder durch Doppelklicken).

Eine `main`-Methode (mit noch leerem Rumpf) wird eingefügt und anschließend steht der Cursor im Rumpf dieser Methode (da wo in der `main`-Schablone die Variable `${cursor}` steht).

Beispiel-02: Die Schablone `sysout` anwenden (*ohne* Textauswahl)

Positionieren Sie den Cursor an einer Stelle, an der Sie einen Befehl der Form `System.out.println(...)`; einfügen *möchten und dürfen* (z.B. innerhalb einer Methodenvereinbarung).

Geben Sie `sysout` ein und dann `Ctrl+Space`.

Der gewünschte Befehl wird eingefügt und anschliessend steht der Cursor zwischen den runden Klammern nach `println`.

Beispiel-03: Die Schablone `sysout` anwenden (*mit* Textauswahl)

Positionieren Sie den Cursor an einer Stelle, an der Sie den Befehl `System.out.println("sum: " + sum)`; einfügen *möchten und dürfen* (z.B. innerhalb der Vereinbarung eines Konstruktors).

Geben Sie (an dieser Stelle) den gewünschten aktuellen Parameter `"sum: " + sum` ein und wählen Sie ihn aus (mit der Maus oder mit der Umschalttaste und den Pfeil-Tasten `←` bzw. `→`).

Geben Sie *zweimal* `Ctrl+Space` ein.

In einem kleinen Fenster werden Ihnen mehrere Alternativen angeboten (aber nur noch *Schablonen*, anderen Alternativen wurden durch die *zweite* Eingabe von `Ctrl+Space` entfernt).

Wählen Sie die Alternative `sysout - print to standard out` aus (mit den Tasten `↓` bzw. `↑` und `Ctrl` oder durch Doppelklicken)

Der gewünschte Befehl wird eingefügt und anschließend steht der Cursor unmittelbar hinter dem aktuellen Parameter (so dass Sie ihn noch ergänzen können, z.B. durch die Eingabe von `+ " Euro"`). Mit einem Druck auf die `Tab`-Taste können Sie den Cursor hinter das abschließende Semikolon des eingefügten Befehls bewegen.

In der Eclipse-Distribution Ganymede sind etwa 80 Schablonen vordefiniert.

5.2. Schablonen definieren

Angenommen, wir wollen häufiger eine Minuszeile wie die folgende in unsere Java-Quelltexte einfügen (z.B. um Methoden optisch deutlich voneinander zu trennen):

```
// -----
```

Eine solche Minuszeile "von Hand" einzugeben ist ziemlich mühsam. Deshalb soll hier als Beispiel eine Schablone namens `mz` mit einer solchen Minuszeile als Bedeutung definiert werden:

Dazu öffnen wir das Templates-Fenster mit dem Befehl

Window / Preferences / + Java / + Editor / Templates

Dort klicken wir auf den Knopf `New ...`. Das `New Template` - Fenster wird geöffnet.

Hinter **Name**: geben wir den Namen `mz` ein.

Hinter **Description**: geben wir eine Beschreibung ein, z.B. `Minus-Zeile`.

Im Textfeld **Pattern** geben wir die Minuszeile und `${cursor}` ein, etwa so:

```
// -----
${cursor}
```

Durch die Variable `${cursor}` wird sichergestellt, dass nach dem Einfügen einer solchen Minuszeile der Cursor genau unter dem ersten Schrägstrich positioniert wird, auch wenn die Minuszeile (um eine oder mehr Stufen) eingerückt ist.

6. Programme umstrukturieren (refactoring)

Angenommen, Sie haben ein großes Programm geschrieben, welches 100 Klassen umfasst. In mehreren davon haben Sie Methoden namens `check` vereinbart. Diese Methoden heißen zwar gleich, leisten aber Verschiedenes. Jetzt möchten Sie eine dieser Methoden in `pruefeBenutzerEingaben` umbenennen. Dann ist es mühsam und fehlerträchtig, mit dem Suchen-und-Ersetzen-Befehl eines Editors in vielen Dateien des aktuellen Projekts genau *die richtigen Vorkommen* des alten Namens durch den neuen Namen zu ersetzen. Leicht passiert es, dass man `check` zu oft oder zu wenig oft durch `pruefeBenutzerEingaben` ersetzt.

Mit den Umstrukturierungsbefehlen (refactoring commands) von Eclipse lässt sich das Problem relativ leicht, schnell und mit geringer Fehlerwahrscheinlichkeit lösen, etwa so:

1. Wählen Sie ein *beliebiges* Vorkommen *des* Namens, den Sie ändern wollen.
2. **Refactor / Rename ...** (oder Alt+Shift+R)
3. Geben Sie den neuen Namen ein und schließen Sie mit Return ab.

Dadurch werden nicht *alle* Vorkommen des alten Namens durch den neuen Namen ersetzt, sondern nur "die richtigen Vorkommen", die "zusammengehören".

Rename ... ist ein besonders einfacher (und gerade deshalb sehr nützlicher) Umstrukturierungsbefehl. Im Menü **Refactor** findet man aber noch viele weitere verwandte Befehle. Einige davon können ziemlich komplizierte Änderung durchführen (z.B. eine Methode aus *einer* Klasse in eine *andere* Klasse verschieben), erfordern aber auch gewisse Vorbereitungen (möglicherweise muss man in der Hilfe nach dem Stichwort `Refactor Actions` suchen und mehrere Abschnitte sorgfältig durchlesen). Angeblich kann man mit den **Refactor**-Befehlen auch Blei in Gold umwandeln, vermutlich aber erst nach einigem Üben.

7. Blitzänderungen (quick fixes)

Wenn der inkrementelle Eclipse-Java-Compiler in einem Quellprogramm einen Fehler findet, versieht er die betreffende Zeile sofort mit einem "roten Pickel". Piekelt man den Pickel mit dem Mauszeiger an (ohne zu klicken), erscheint die zugehörige Fehlermeldung (ähnlich wie ein tool tip).

Wenn man den roten Pickel *anklickt* (oder den Cursor in die betreffende Zeile bringt und Ctrl+1 eingibt) erscheint ein kleines Fenster, in dem (in aller Regel) verschiedene Blitzänderungen (engl. quick fixes) zur Beseitigung des Fehlers vorgeschlagen werden. Es empfiehlt sich, diese Vorschläge häufig genauer anzusehen und sich die Situationen zu merken, in denen sie nützlich waren. Häufig kann man sich Tipparbeit sparen, indem man absichtlich etwas Unvollständiges oder Falsches eingibt und diese Eingabe dann durch eine bestimmte Blitzänderung automatisch ergänzen oder korrigieren lässt.

Beispiel-01: Den Rückgabotyp `void` automatisch ergänzen lassen

Geben Sie den Anfang einer Prozedurvereinbarung ein und "vergessen" Sie dabei, den Rückgabotyp `void` anzugeben, etwa so:

```
public print(String s)
```

Am Anfang der Zeile erscheint ein roter Pickel und der falsche Text wird rot unterstrichen. Geben Sie Ctrl+1 ein (während der Cursor sich noch im rot unterstrichenen Bereich befindet) oder bewegen Sie den Mauszeiger über den falschen Text (ohne zu klicken). In einem kleinen Fenster werden Ihnen mehrere Blitzänderungen angeboten. Wählen Sie die Alternative `Set method return type to 'void'` (mit den Tasten ↓ bzw. ↑ und Ctrl oder durch Doppelklicken). Dadurch wird der fehlende Rückgabotyp automatisch ergänzt.

Dieses Beispiel erspart einem nicht viel Arbeit, zeigt aber den grundsätzlichen Ablauf einer Blitzänderung. Das folgende Beispiel ist (in bestimmten Situationen) etwas nützlicher.

Beispiel-02: Ein Methoden-Skelett automatisch einfügen lassen.

Angenommen, Sie brauchen in Ihrem Programm eine Funktion der folgenden Form:

```
public int machWas(int nummer, String text) {  
    ...  
}
```

Statt zuerst die *Vereinbarung* der Funktion einzutippen und danach erst die benötigten *Aufrufe* zu programmieren, kann man in Eclipse auch genau andersherum vorgehen: Zuerst gibt man (an einer geeigneten Stelle) einen *Aufruf* der Funktion ein, z.B. so:

```
int erg = machWas(17, "ABC");
```

Diesen Befehl versteht der (inkrementelle) Compiler sofort mit einem Fehler-Pickel, weil die Funktion `machWas` noch nicht vereinbart ist.

Geben Sie **Ctrl+1** ein. Ein Fenster mit Blitzänderungen wird geöffnet. Wählen Sie die Alternative `Create method 'machWas(int, String)'` (mit den Tasten **↓** bzw. **↑** und **Ctrl** oder durch **Doppelklicken**). Dadurch wird folgendes Methoden-Skelett (engl. `method stub`) in Ihr Programm eingefügt:

```
private int machWas(int i, String string) {  
    // TODO Auto-generated method stub  
    return 0;  
}
```

Mehrere Worte in diesem Skelett (`private`, `int`, `machWas`, ...) sind einzeln von schwarzen Rechtecken umgeben. Das bedeutet, dass Sie mit der **Tab**-Taste von Wort zu Wort springen und einige davon ändern können, z.B. `private` zu `public`, `i` zu `nummer` und `string` zu `text`. Mit **Shift+Tab** können Sie auch rückwärts springen.

Der "Trick" in diesem Beispiel: Man gibt zuerst einen *falschen Befehl* ein und lässt ihn dann mit einer Blitzänderung *korrigieren*.

Man kann sich Blitzänderungen aber auch anbieten lassen, *ohne* zuerst einen Fehler einzugeben.

Beispiel-02: Eine Befehlsfolge zu einer Methode "veredeln"

Angenommen, Sie haben eine Methode geschrieben, deren Rumpf viel zu kompliziert geworden ist. Jetzt wollen Sie einen Teil des Rumpfes in eine separate Methode auslagern.

Wählen Sie die auszulagernden Befehle aus und geben Sie **Ctrl+1** ein.

Ein Fenster mit Blitzänderungen erscheint.

Wählen Sie die Alternative `Extract to method` (mit den Tasten **↓** bzw. **↑** und **Ctrl** oder durch **Doppelklicken**).

Die ausgewählten Befehle werden in eine separate Methode namens `extracted` ausgelagert und durch einen Aufruf dieser Methode ersetzt. Falls nötig, wird diese Methode automatisch mit bestimmten Parametern versehen, die von den ausgelagerten Befehlen benötigt werden.

Der Name `extracted` der neuen Methode erscheint an zwei Stellen: In der Vereinbarung der Methode und da, wo sie (im Rumpf der alten Methode) aufgerufen wird. Man braucht den Namen aber nur an *einer* Stelle zu ändern, die andere Stelle wird dann automatisch angepasst.

Der "Trick" in diesem Beispiel: Man gibt erst den *Rumpf* einer Methode ein und lässt daraus per Blitzänderung eine *Methode* machen.

8. Das Help-Menü

Probieren Sie möglichst bald die beiden Befehle

Help / Help Contents und
Help / Search

aus und machen Sie sich mit den Unterschieden vertraut.

9. Eine Java-Quelldatei in ein Eclipse-Projekt importieren

Beispiel: In einem Verzeichnis namens `D:\meineDateien\quellen\` steht eine Datei namens `Hallo.java`. Diese Datei wollen Sie (als Quelldatei) in ein Eclipse-Projekt namens `Pro01` importieren.

Öffnen Sie das Kontextmenü des Ordners `src` des Projektes `Pro01` (im **Package Explorer** auf `src` rechtsklicken).

Wählen Sie darin **Import ...**. Dadurch wird das Fenster **Import-Select** ("der Import-Wizard") geöffnet.

+ **General / File System**

Next (oder Doppelklick auf **File System**). Dadurch wird das Fenster **Import-File System** geöffnet.

In das Textfeld hinter **From Directory:** das Verzeichnis `D:\meineDateien\quellen` *eintragen* (oder nach klicken auf **Browse ... auswählen**). Der Name `quellen` sollte im linken der beiden mittelgroßen Fenster erscheinen. Auf den Namen `quellen` klicken. Im rechten mittelgroßen Fenster sollten alle in Frage kommenden `.java`-Dateien mit einem Auswahlknopf davor erscheinen.

Die Datei `Hallo.java` auswählen (durch Häkchen im Auswahlknopf).

Stellen Sie sicher, dass im Textfeld nach **Into Folder:** `Pro01/src` steht (sonst eintragen). **Finish**.

Die Datei `Hallo01.java` sollte jetzt in einen Unterordner von `Pro01/src` kopiert werden (der Unterordner hängt vom Paket ab, zu dem `Hallo01.java` gehört. Das namenlose Paket hat in Eclipse den Namen **default package**, die anderen Pakete heißen so, wie sie heißen :-).

10. Ein Java-Archiv (d.h. eine .jar-Datei) zu einem Projekt hinzufügen

Die Abkürzung *jar* steht für *Java-Archiv*. Ein solches Archiv kann Klassen (`.class`-Dateien) und andere Dateien enthalten; es muss ein sog. *Manifest* (eine Art Inhaltsverzeichnis) enthalten. Bevor man seinen Inhalt in einem Eclipse-Projekt benutzen kann, muss man das Archiv zum Buildpath des Projekts hinzufügen.

Ein Java-Archiv namens `jdom.jar`, welches im Verzeichnis `D:\jars` abgespeichert wurde, können Sie wie folgt zum Buildpath eines Eclipse-Projekt namens `Pro01` hinzufügen:

Wählen Sie im **Package Explorer** das Archiv `Pro01`.

Dann: **Project, Properties**. Das Fenster **Properties for Pro01** sollte sich öffnen.

Klicken Sie ganz links in diesem Fenster auf **Java Build Path** und dann auf den Reiter **Libraries**.

Rechts im Fenster erscheinen Knöpfe mit den Aufschriften **Add JARs ...**, **Add external JARs ...** etc. Der oberste Knopf ist für den Fall gedacht, dass die einzufügende `.jar`-Datei sich bereits in einem Ordner des Projekts befindet. Da in unserem Beispiel die `.jar`-Datei außerhalb des Projekts (im Verzeichnis `D:\jars`) steht, klicken Sie auf den zweitobersten Knopf (**Add external JARs ...**).

Das Fenster **JAR Selection** sollte aufgehen.

Navigieren Sie mit diesem Fenster zum Verzeichnis `D:\jars`, wählen Sie dort die Datei `jdom.jar` und klicken Sie auf den Knopf **Öffnen**. Dadurch wird die Datei `jdom.jar` nicht geöffnet, sondern zum Buildpath des aktuellen Projekts hinzugefügt.

Ab jetzt können Sie den Inhalt des Archivs `jdom.jar` benutzen und z.B. die folgenden Befehle in eine Quelldatei des Projekts `Pr01` einfügen:

```
1 import org.jdom.Document;  
2 ...  
3 class Otto {  
4     Document doc1 = new Document();  
5     ...
```

Die Klasse `org.jdom.Document` befindet sich im Java-Archiv `jdom.jar`.

Anmerkung: Ein Java-Archiv (eine .jar-Datei) ist eigentlich ein ZIP-Archiv mit einem Java-Manifest darin (und einem Namen der mit .jar endet statt mit .zip). Jedes Entpackerprogramm für ZIP-Archive kann auch .jar-Dateien entpacken und einem zeigen, "was drin ist".

11. Ein Eclipse-Projekt auf einen anderen Rechner übertragen

Angenommen, Sie haben Eclipse auf zwei Rechnern (R1 und R2) installiert und auf dem Rechner R1 ein Projekt `Pr01` entwickelt. Jetzt wollen Sie dieses *Projekt* ("mit allem Drum und Dran") mit Hilfe eines USB-Speicher-Sticks auf den Rechner R2 *übertragen*. Hier wird angenommen, dass der Stick auf dem Rechner R1 (automatisch) mit dem Laufwerksbuchstaben `I:` und auf R2 mit `J:` verbunden wird.

Stecken Sie den Stick in den Rechner R1.

Öffnen Sie das Kontextmenü des Projekts `Pr01` (indem Sie im **Package Explorer** auf `Pr01` rechtsklicken). Wählen Sie **Export ...**. Dadurch wird das Fenster **Export-Select** geöffnet.

+ **General / File System**. Dadurch wird das **Export - File system** - Fenster geöffnet.

Machen Sie ein Häkchen bei dem zu exportierenden Projekt-Ordner `Pr01`.

Geben Sie im Textfeld hinter **To directory:** das Verzeichnis `I:\` ein (oder wählen Sie das Verzeichnis nach einem Klick auf **Browse:**).

Finish. Der Projekt-Ordner `Pr01` wird in das angegebene Verzeichnis `I:\` kopiert (die Kopie hat somit den Pfad `I:\Pr01`).

Übertragen Sie den Stick vom Rechner R1 zum Rechner R2.

Erzeugen Sie auf dem Rechner R2 ein neues Projekt namens `Pr01`.

Öffnen Sie das Kontextmenü des neuen Projekts (indem Sie im **Package Explorer**, auf `Pr01` rechtsklicken). Wählen Sie **Import ...**. Dadurch wird das **Import-Select** - Fenster geöffnet.

+ **General / File System**. Dadurch wird das **Import - File System** - Fenster geöffnet.

Geben Sie im Textfeld hinter **From directory:** den Ordner `J:\Pr01` ein (oder wählen Sie das Verzeichnis nach einem Klick auf **Browse:**).

Machen Sie (im linken mittelgroßen Fenster) ein Häkchen vor dem Ordner `Pr01`.

Finish.

Dadurch wird ein Dialog-Fenster **Question** geöffnet mit der Frage:

Overwrite '.classpath' in folder Pr01? Wählen Sie **No To All** als Antwort.

Jetzt sollte das Projekt vom Stick auf den Rechner R2 kopiert werden.

12. Eine JUnit-Testklasse compilieren und ausführen lassen

Angenommen, Sie wollen in einem Eclipse-Projekt `Jut01` eine JUnit-Testklasse entwickeln. Dann müssen Sie den so genannten **Build Path** (die Liste aller Ordner und .jar-Dateien, in denen der Compiler nach Klassen sucht, die ihm noch fehlen) so ändern, dass er auch die benötigten JUnit-Klassen enthält.

Öffnen Sie das Kontextmenü des Projekts `Jut01` (durch Rechtsklicken auf den Projektnamen, im **Package Explorer**, im Eclipse-Fenster ganz links).

Wählen Sie darin **Build Path / Add Libraries ...** . Dadurch wird das **Add Library**-Fenster geöffnet.

Doppelklicken Sie darin auf **JUnit** (oder wählen Sie **JUnit** und klicken Sie auf **Next >**). Dadurch wird das Fenster **Add Library-JUnit Library** geöffnet.

Wählen Sie im Fenster hinter **JUnit library version:** entweder **JUnit 3** oder **JUnit 4** (je nachdem, welche JUnit-Version Sie verwenden wollen).

Finish. Jetzt sollte der Compiler alle Klassen der gewählten JUnit-Version finden und nicht mehr melden: "Can't resolve ... to a type name".

Hinweis 1 zu den JUnit-Versionen:

Eine **JUnit 3** Testklasse beginnt typischerweise etwa wie folgt:

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class StringBuilderJut01 extends TestCase {
    ...
}
```

Eine **JUnit 4** Testklasse beginnt typischerweise etwa wie folgt:

```
import static org.junit.Assert.assertEquals;
import junit.framework.JUnit4TestAdapter;

import org.junit.Before;
import org.junit.Test;

public class StringBuilderJut02 {
    ...
}
```

Hinweis 2 zu den JUnit-Versionen:

Die beiden Versionen von JUnit (3 und 4) "vertragen sich miteinander". D.h. es ist möglich, innerhalb *eines* Projektes Testklassen *beider* Versionen nebeneinander zu entwickeln.

Ausführen lassen können Sie eine JUnit-Klasse (egal ob Version 3 oder 4) mit folgendem Kommando:

Run / Run as / JUnit Test

oder durch Eingabe von **Alt+Shift+X, T**.

13. Quelldateien von Standardklassen anzeigen lassen

Wenn Sie den Namen einer Klasse kennen (oder zumindest wissen, wie er *anfängt*), können Sie sich den *Quelltext* der Klasse anzeigen lassen. Das entsprechende Fenster **Open Type** lässt sich auf drei verschiedene Weisen öffnen:

1. Durch einen Klick auf einen kleinen Knopf links unter dem **Window**-Menü (der Knopf hat den Tool-Tip-Text **Open Type Ctrl+Shift+T**).
2. Durch Eingabe von **Ctrl+Shift+T** .
3. Mit dem Menübefehl **Navigate / Open Type ...**

Im Fenster **Open Type** geben Sie im oberen (einzeiligen) Textfeld den Namen der Klasse ein. Während Sie noch tippen, wird Ihnen im unteren (mehrzeiligen) Textfeld eine Liste von Klassennamen angeboten, die zu Ihren Eingaben passe. Die Liste wird kürzer, je mehr Zeichen Sie eingeben. Wenn Sie

in dieser Liste einen Namen auswählen und auf Return drücken (oder auf den Namen doppelklicken) wird der Quelltext der Klasse (in einem Editor) angezeigt.

Das gilt nicht nur für Klassen, die Sie in irgendeinem Projekt (im aktuellen Workspace) entwickelt haben, sondern auch für Klassen der Java-Standardbibliothek, z.B. `String` oder `ArrayList` etc.

14. Die XML-Beispiele in ein Eclipse-Projekt übernehmen

Das Archiv `DateienFuerPR2.zip` (auf der Netzseite public.beuth-hochschule.de/~grude) enthält ein Verzeichnis XML. Darin befinden sich mehrere *Beispiel-Programme* (`.java`-Dateien) zum Thema XML und zahlreiche *Daten-Dateien*, die von den Beispiel-Programmen eingelesen werden sollen. Die Beispielprogramme verlassen sich darauf, dass (wenn sie ausgeführt werden) alle benötigten Daten-Dateien im *aktuellen Arbeitsverzeichnis* liegen. Ausserdem können einige der Beispielprogramme nur dann kompiliert und ausgeführt werden, wenn bestimmte Java-Archive (`.jar`-Dateien) zugänglich (d.h. in der Umgebungsvariablen `CLASSPATH` eingetragen) sind.

Im folgenden wird beschrieben, wie man diese Beispiel-Programme und Daten-Dateien in ein Eclipse-Projekt übernehmen kann. Dabei wird angenommen, dass das Archiv `DateienFuerPR2.zip` entpackt wurde, so dass die zu übernehmenden Beispiel-Programme und Daten-Dateien in einem Verzeichnis `...\DateienFuerPR2\XML` liegen.

Schritt 1: In Eclipse ein neues Projekt erstellen.

Menü **File, New, Java Project** wählen. Im Fenster **New Java Project** oben neben **Project Name** den Namen des Projekts eintragen. Auf **Finish** klicken. Für das neue Projekt wird automatisch ein Verzeichnis namens `src` erzeugt, in dem man alle Java-Quelldateien (`.java`-Dateien) ablegen sollte.

Im folgenden wird angenommen, dass dieses neue Projekt `XmlProjekt01` heisst.

Schritt 2: Im `XmlProjekt01` ein Verzeichnis für die Daten-Dateien anlegen:

Öffnen Sie das **Kontext-Menü** des Projekts (indem Sie im **Package Explorer** auf `XmlProjekt01` rechtsklicken) und wählen Sie **New, Folder**. Im Fenster **New Folder** den Namen des neuen Verzeichnisses eingeben. Hier wird angenommen, dass das neue Verzeichnis `DatenDateien` heisst.

Schritt 3: Die Beispiel-Programme in das Verzeichnis `src` importieren:

Öffnen Sie das **Kontext-Menü** des Verzeichnisses `src` (indem Sie im **Package Explorer** auf `src` rechtsklicken) und wählen Sie **Import ...**. Im **Import** - Fenster + **General / File System** wählen und auf **Next >** klicken. Dann auf **Browse ...** klicken und zu dem Verzeichnis navigieren, in dem die zu importierenden Dateien stehen (hier wird angenommen: `...\DateienFuerPR2\XML`). Alle Dateien in diesem Verzeichnis werden angezeigt. Nur die Java-Quelldateien (`.java`-Dateien) auswählen (aber keine vergessen!) und auf **Finish** klicken. Dadurch werden die ausgewählten Dateien in das Verzeichnis `src` kopiert.

Einige der kopierten Quelldateien werden (vom inkrementellen Java-Compiler) sofort mit roten Fehler-Pickeln versehen. Die werden (hoffentlich) nach dem **Schritt 5** (siehe unten) verschwinden.

Schritt 4: Die Daten-Dateien in das Verzeichnis `DatenDateien` importieren:

Öffnen Sie das **Kontext-Menü** des Verzeichnisses `DatenDateien` (indem Sie im **Package Explorer** auf `DatenDateien` rechtsklicken) und wählen Sie **Import ...**. Im **Import** Fenster + **General / File System** wählen und auf **Next >** klicken. Dann auf **Browse ...** klicken und zum Verzeichnis navigieren, in dem die zu importierenden Dateien stehen (hier wird angenommen: `...\DateienFuerPR2\XML`). Alle Dateien in diesem Verzeichnis werden angezeigt. Nur die Daten-Dateien (mit den Namenserverweiterungen `.xml`-, `.dtd`-, `.xsd`-, `.txt`) auswählen (aber keine vergessen!) und auf **Finish** klicken. Dadurch werden die ausgewählten Dateien in das Verzeichnis `DatenDateien` kopiert.

Schritt 5: Benötigte Java-Archive (.jar-Dateien) zum Projekt hinzufügen

Benötigt werden (von einigen der Beispiel-Programme zum Thema XML) die drei Java-Archive

`jdom.jar`,

`xercesImpl.jar` und

`xml-apis.jar`.

Im SWE-Labor findet man diese Archive im Verzeichnis `D:\jars`.

So fügt man die Archive zum `XmlProjekt01` hinzu: Menü **Project, Properties** wählen. Im Fenster **Properties for XmlProjekt01** in der linken Spalte **Java Build Path** wählen, dann etwa in der Mitte des Fenster den Reiter **Libraries** wählen und schließlich ganz rechts auf **Add External JARs ...** klicken. Im **JAR Selection** Fenster zum Verzeichnis `D:\jars` navigieren, die benötigten Archive auswählen und auf **Öffnen** klicken (keine Angst, die Archive werden dadurch nicht geöffnet, sondern kopiert :-).

Schritt 6: Das Verzeichnis `DatenDateien` als Arbeitsverzeichnis festlegen

Das aktuelle Arbeitsverzeichnis wird in einer **Run Configuration** festgelegt. Deshalb muss dieser Schritt für jedes *Programm* wiederholt werden. Hier wird er beispielhaft für das Programm `XmlJStd` beschrieben:

1. Das Programm `XmlJStd` ausführen lassen (Menü **Run, Run as, Java Application**).

Dadurch wird eine **Run Configuration** für das Programm `XmlJStd` erzeugt (falls es noch keine gibt). Das Programm `XmlJStd` wird jetzt Fehlermeldungen ausgeben, weil es mehrere `Daten-Dateien` (noch) nicht finden kann.

2. Menü **Run, Run Configurations ...** wählen. Im Fenster **Run Configurations** wird dann die **Run Configuration** des zuletzt ausgeführten Programms angezeigt.

3. Auf den Reiter (x) = **Arguments** klicken.

4. Ganz unten anstelle von **Default:** die **Alternative Other:** auswählen und daneben das gewünschte Arbeitsverzeichnis eintragen, z.B. so: `${workspace_loc:XmlProjekt01/DatenDateien}`

Wenn man das Programm `XmlJStd` jetzt erneut ausführen lässt (indem man auf **Run** klickt) sollte es alle benötigten `Daten-Dateien` finden und keine diesbezüglichen Fehlermeldungen mehr ausgeben.

Achtung: Die `Daten-Datei` `daten01E.xml` (E wie Error) enthält absichtlich Fehler und sollte auch jetzt noch Fehlermeldungen provozieren.

15. Pfadnamen und URIs

Viele Java-Methoden erwarten als Parameter einen String, der einen *Pfadnamen* enthält. In einem solchen Pfadnamen kann man (auch unter Windows) die einzelnen Teile durch Schrägstriche / voneinander trennen, z.B. so:

```
String pfad01 = "D:/VerzeichnisA/Datei05.txt";
```

Freunde von Rückwärtsschrägstrichen \ können (nur unter Windows) Pfadnamen auch so notieren:

```
String pfad02 = "D:\\VerzeichnisA\\Datei05.txt";
```

Einige Java-Methoden erwarten als Parameter einen String, der einen *URI* (Universal Resource Identifier) enthält. Falls dieser URI auf eine Datei auf dem lokalen Rechner zeigen soll, kann er z.B. so aussehen:

```
String uri01 = "file://localhost/D:/VerzeichnisA/Datei05.txt";
```

Manchmal darf bzw. muss man die Angabe wie folgt abkürzen:

```
String uri02 = "file:///D:/VerzeichnisA/Datei05.text";
String uri03 = ":///D:/VerzeichnisA/Datei05.text";
```

16. Suchen und Ersetzen mit regulären Ausdrücken und Fangmustern

Mit dem Find/Replace-Dialog kann man in einem Text bestimmte Strings *suchen* und durch andere Strings *ersetzen*. Öffnen kann man diesen Dialog mit dem Befehl

Edit / Find/Replace oder **Ctrl-F**

(der zweite Schrägstrich ist ein Teil des Menüpunkt-Namens). Das relativ kleine Dialogfenster kann man (durch Ziehen mit der Maus an einem der vier Ränder) vergrößern und verkleinern.

Im simpelsten Fall gibt man hinter **Find:** und **Replace with:** *konkrete Strings* an, weil man z.B. alle Vorkommen des konkreten Strings `Hallo` suchen und durch den konkreten String `hello` ersetzen möchte.

Wenn man im Find/Replace-Dialog vor **Regular expressions** ein Häkchen macht, werden die Eingaben hinter **Find:** nicht mehr als konkrete Strings sondern als *reguläre Ausdrücke* (kurz: als *Muster*) interpretiert. Ein Muster kann für *einen* konkreten String oder aber für *mehrere* konkrete Strings stehen. Es folgen drei typische Beispiele:

Muster (eingetragen nach Find:)	Steht für oder passt auf:
Hallo	Hallo
[A-Z]+	Alle nicht-leeren Folgen von Zeichen zwischen A und Z. Beispiele: A oder XYZ oder XXX oder SUMME . Gegenbeispiele: a oder A3 oder Ä.
[+-]?[0-9]	Alle nicht-leeren Folgen von Zeichen zwischen 0 und 9, mit oder ohne einem Vorzeichen + oder - davor. Beispiele: 7 oder -123 oder +123 oder 0 oder -000. Gegenbeispiele: a7 oder 123- oder 0+0.

Außerdem bewirkt das Häkchen vor **Regular expressions**, dass man hinter **Find:** sog. *Fangmuster* (engl. capturing groups) eintragen darf und dann hinter **Replace:** mit den Namen `\1`, `\2`, `\3`, ... die gefangenen Zeichenketten bezeichnen kann. Es folgt ein (hoffentlich) einfaches Beispiel, bei dem Namen der Form `Meier, Otto` oder `Acar, Murat` etc. durch `Otto Meier` bzw. `Murat Acar` etc. ersetzt werden sollen.

Ein Fangmuster (hinter **Find:**) besteht aus einem Paar runder Klammern mit einem Muster darin. Das folgende Find-Muster enthält zwei Fangmuster (und dazwischen die Zeichen `,`):

Find: `([A-Z][a-z]+), ([A-Z][a-z]+)`

Replace with: `\2 \1`

Wendet man dieses Find-Muster z.B. auf die Zeichenkette `Meier, Otto` an, so wird der Nachname `Meier` vom ersten Fangmuster gefangen und der Vorname `Otto` vom zweiten. Die beiden Zeichen `,` zwischen dem Nach- und dem Vornamen werden von *keinem* der Fangmuster gefangen.

Hinter **Replace with:** bezeichnet `\2` den Fang des zweiten Fangmusters und `\1` den Fang des ersten Fangmusters. Zwischen den Variablen `\2` und `\1` steht in diesem Beispiel noch ein Blank (damit `Meier, Otto` durch `Otto Meier` und nicht durch `OttoMeier` ersetzt wird).

Anmerkung: Im obigen Beispiel werden nur die 26 Zeichen zwischen A und Z als Großbuchstaben und die 26 Zeichen zwischen a und z als Kleinbuchstaben erkannt. Sollen auch Zeichen wie ä, ö, ü, Ä, Ö, Ü, ß, é, ê etc. als Buchstaben erkannt werden, sollte man anstelle von `[A-Z]` und `[a-z]` die folgenden Teilmuster verwendet:

`\p{L}` für einen beliebigen (großen oder kleinen) *Buchstaben* (L wie "letter")
`\p{Ll}` für einen beliebigen *kleinen Buchstaben* (Ll wie "letter, lowercase")

`\p{Lu}` für einen beliebigen *großen Buchstaben* (Lu wie "letter, uppercase")

Damit werden 708 Unicode-Zeichen als Großbuchstaben und 886 Zeichen als Kleinbuchstaben erkannt (feststellbar durch die Methoden `Character.isUpperCase` und `Character.isLowerCase`).

Aufgabe: Mit welchem Muster nach **Find:** und welchem Eintrag nach **Replace with:** kann man umgekehrt Namen der Form `Otto Meier` oder `Murat Acar` durch solche der Form `Meier, Otto` bzw. `Acar, Murat` ersetzen?

Eine *vollständige Beschreibung* der regulären Ausdrücke (Muster), die man hinter **Find:** eintragen darf, findet man in der *Online-Dokumentation der Java-Klasse* `Pattern` (denn Eclipse ist in Java programmiert).

Eine knappe Zusammenfassung der Dokumentation bekommt man auch, wenn man im Find/Replace-Dialog den Cursor im Eingabe-Textfeld hinter **Find:** (bzw. hinter **Replace with:**) positioniert und `Ctrl+Space` drückt.

Es folgen hier noch ein paar Beispiele. Die doppelten Anführungszeichen " dienen hier nur dazu, Blanks (Leerzeichen, spaces) erkennbar zu machen. Sie müssen weggelassen werden, wenn man die Ausdrücke hinter **Find:** bzw. **Replace with:** eingibt:

Beispiel 1: An jede Ziffernfolge den Text `Euro` anhängen:

Find: `"(\d+)"` **Replace with:** `"\1 Euro"`

Anmerkung: `\d` (wie digit) bedeutet das Gleiche wie `[0-9]`.

Beispiel 2: An jede Buchstabenfolge ein Komma anhängen:

Find: `"(\p{Alpha}+)"` **Replace with:** `"\1, "`

Anmerkung: `\p{Alpha}` bedeutet das Gleiche wie `[A-Za-z]`. Achtung: Die Umlaute Ä, Ö und Ü liegen nicht zwischen A und Z, und für ä, ö, ü und ß gilt Entsprechendes.

Beispiel 3: Jede nicht-leere Folge von Blanks (spaces) und Tab-Zeichen durch ein Blank ersetzen:

Find: `"[\ \t]+"` **Replace with:** `" "`

Beispiel 4: Jedes `double`-Literal der Form `0.0` oder `12.3456` etc., wird durch ein entsprechendes `float`-Literal (mit `F` dahinter) ersetzen.

Find: `"(\d+\.\d+)"` **Replace with:** `"\1F"`

Anmerkung: Der Punkt `.` hat normalerweise eine spezielle Bedeutung ("irgendein Zeichen"). Die wird hier mit dem Rückwärtsschrästrich `\` beseitigt und `\.` steht nur für einen Punkt.

Die folgende Lösung erlaubt auch Literale wie `.0` oder `.123` oder `0.` oder `123.:`

Find: `"(\d+\.\d*|\d*\.\d+)"` **Replace with:** `"\1F"`

Anmerkung: Der Stern `*` bedeutet: Null oder mehr Mal. `\d*` bedeutet somit: Null oder mehr Ziffern. Der senkrechte Strich `|` bedeutet *oder* (er trennt Alternativen voneinander).

Reguläre Ausdrücke editieren: Besonders beim Eingeben von regulären Ausdrücken sollte man jedes Zeichen (auch Punkte und Kommas) deutlich erkennen können. Wenn einem der Font im Find/Replace-Dialog "zu klein und fitzelig" erscheint, kann man ihn mit folgendem Befehl ändern (z.B. von 10 Punkten auf 12 Punkte vergrößern):

Window / Preferences / + General / + Appearance / Colors and Fonts / Dialog Font

Eine andere Technik besteht darin, dass man reguläre Ausdrücke in einer speziellen *Textdatei* (d.h. `.txt`-Datei) entwickelt und aufbewahrt (evtl. mit Kommentaren versehen) und sie bei Bedarf in den Find/Replace-Dialg rüberkopiert (mit `Ctrl-C` und `Ctrl-V`). Den Font für Text-Dateien kann man mit folgendem Befehl verändern:

Window / Preferences / + General / + Appearance / Colors and Fonts / Text Font

Besonders empfehlenswert sind für reguläre Ausdrücke Mono-Fonts (bei denen alle Zeichen gleich breit sind) wie z.B. *Courier*, *Courier New* oder *Lucida Console*.

Abschließende Anmerkung: Je früher und gründlicher man sich mit *regulären Ausdrücken* und *Fangmustern* vertraut macht, desto mehr langweilige Editier-Aufgaben kann man durch interessante Denksportaufgaben ersetzen. Statt z.B. von Hand hinter 17 Strings je ein Komma einzufügen, was höchstens 2 Minuten dauert, kann man sich auch fragen: "Mit welchem regulären Ausdruck kann ich diese Kommas automatisch einfügen lassen?", und mit dieser Frage kann man sich (wenn man zum ersten Mal darauf stößt) viel länger als 2 Minuten beschäftigen :-). Und wenn man ähnliche Probleme häufiger mit regulären Ausdrücken löst, kann man damit sogar Zeit sparen.

17. Mehr über Eclipse lernen

Mit dem Befehl **Help / Tips and Tricks ...** kann man einen Abschnitt der Eclipse-Hilfe öffnen, in dem interessante Tips und Tricks erklärt werden.

Unter der Adresse www.eclipse.org/resources/ findet man etwa 300 Artikel, Präsentationen, Demos, Bücher und Podcasts über bestimmte Aspekte von Eclipse. Die meisten sind auf English, aber einige auch auf Chinesisch, Koreanisch, Japanisch, Arabisch, Französisch (sehr beruhigend) und etwa 10 sogar auf Deutsch (wieso eigentlich? :-).

Nehmen Sie sich vor, jede Woche oder alle 14 Tage ein bisschen mehr über Eclipse zu lernen. Vielleicht können Sie dann schon in naher Zukunft aus Blei Gold machen (siehe dazu den Abschnitt 6.).