# Nutzungshinweise für BlueJ

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES

| | |
|---|---|
| **Thema:** | Nutzung der Entwicklungsumgebung BlueJ |
| **Autoren:** | Prof. Dr. Stephan Kleuker |
| **Version / Datum:** | 2.2 / 12.8.2021 |
| **Empfänger:** | Teilnehmer der Lehrveranstaltungen im Bereich Programmierung der Studiengänge Informatik |

# 0 KleukersSEU

Um Konflikte mit anderen Software-Paketen zu vermeiden, bietet Prof. Dr. Kleuker Teilnehmern seiner Veranstaltungen ein Verzeichnis an, dass alle in seinen Veranstaltungen benötigte Software beinhaltet. Dieses Verzeichnis ist auf den Hochschulrechnern installiert und kann auf eigene Rechner oder einfach einen USB-Stick kopiert werden. Es ist dabei zu beachten, dass Einstellungen, die im Nutzerkonto gespeichert werden, auf jedem Rechner neu einzurichten sind.

Weitere Hinweise und das Download-Paket können http://home.edvsz.hs-osnabrueck.de/skleuker/kleukersSEU/index.html entnommen werden.

# 1 Vorbemerkungen

Diese Anleitung fasst einige Tipps und Tricks zum Umgang mit BlueJ in den einführenden Lehrveranstaltungen zur Programmierung zusammen. Dieser Text ist nicht als vollständiges Manual anzusehen. In der Lehrveranstaltung, insbesondere in den Praktika wird für konkrete Aufgaben auf einzelne Kapitel dieses Texts verwiesen.

BlueJ ist ein recht mächtiges Werkzeug, das allerdings dann sehr einfach zu bedienen ist, wenn die typischen Arbeitsabläufe genutzt werden, die fast kein Detailwissen über BlueJ erfordern. BlueJ ist deshalb sehr gut für Anfänger geeignet. Jeder Nutzer ist aufgefordert, sich selbst intensiver mit BlueJ zu beschäftigen, da dies einige Arbeitsschritte noch vereinfachen kann.

Diese Hinweise fokussieren auf die Microsoft-Version, fast alle Informationen können aber auch auf die anderen Versionen übertragen werden. Damit BlueJ funktioniert, ist vorher Java, genauer ein JDK ab der Version 6, zu installieren. Bei der Installation ist u. a. die korrekte Setzung der Pfadvariablen (PATH) zu beachten, in die das bin-Verzeichnis der Java-Installation einzutragen ist, was im nachfolgenden Kapitel beschrieben wird.

Im Folgenden bezieht sich die textuelle Beschreibung immer auf das *nachfolgende* Bild. Bei einzelnen Bildern können nach dem Bild noch Kommentare ergänzt werden. Wird in den Bildern die Position des Mauszeigers gezeigt, weist diese auf die auszuführende Aktion hin. *Am Rande sei bemerkt, dass dies kein akademischer Text ist und keine akademische Schreibweise genutzt wird, bei der u. a. alle Abbildungen zu beschriften sind und jede Abbildung explizit im Fließtext erwähnt werden muss. Weiterhin hat keine intensive Rechrschreibkontrolle stattgefunden.*

Generell benötigen Anfänger sehr wenige Befehle, um sinnvoll mit BlueJ arbeiten zu können. Bei Zeit und Interesse ist jeder aufgefordert, sich z. B. durch das Lesen der Tutorials in den Help-Files intensiver zu informieren. Im Text steht häufiger ein „z. B.“, um anzudeuten, dass es verschiedene Wege gibt, die gleiche Aktion durchzuführen.

Diese Hinweise enthalten Beschreibungen, wie das Werkzeug genutzt werden kann. Generell sollte ein Entwickler sich immer fragen, warum etwas gemacht wird. Diese Fragen werden getrennt in den Vorlesungen zur Programmierung behandelt.

Dieser Text lebt weiterhin von Kommentaren der Nutzer; wenn Sie Anmerkungen haben, sei auch nur ein einfacher Rechtschrei*p*fehler, schicken Sie diese an einen der Autoren.

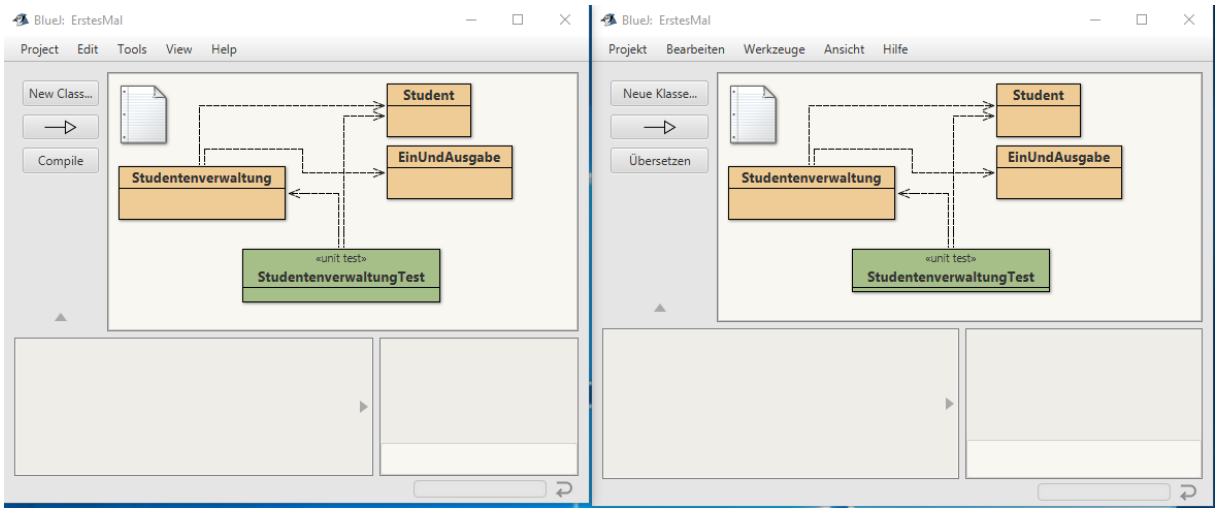
Weitere Informationen zu BlueJ können natürlich von den zugehörigen Web-Seiten geladen werden. Hilfreich ist dabei auch die Seite http://www.bluej.org/doc/documentation.html wobei sich die Informationen meist auf ältere Versionen von BlueJ beziehen, was aber (im Wesentlichen) kein Problem ist. Am Anfang sind das „BlueJ Tutorial“ und das „BlueJ Reference Manual“ mit recht ähnlichen Inhalten sehr hilfreich.


Weitere Informationen sind natürlich googlebar, dabei wird die Suche nach BlueJ zusammen mit filetype:pdf vorgeschlagen, wodurch PDF-Dokumente mit BlueJ im Inhalt gefunden werden. Das Ergebnis enthält einige mit diesen Hinweisen vergleichbare Dokumente, die ebenfalls zum Selbststudium geprüft werden können. Natürlich sind einige Besonderheiten dieses Dokuments, wie die absichtliche Wiederholung einiger Schritte ohne die Nutzung nerviger Querverweise und das Eingehen auf Probleme und Lösungsmöglichkeiten, in anderen

Dokumenten nicht (oder nur teilweise) enthalten. An den erhaltenen Suchergebnissen kann auch die hohe Verbreitung von BlueJ an Universitäten, Hochschulen und sonstigen Bildungseinrichtungen abgelesen werden.


Wird filetype:ppt genutzt, ist u. a. weiteres Lehrmaterial auffindbar.

Viele Informationen und Abbildungen beziehen sich auf die BlueJ-Version 4.1.2, dabei sind fast alle Vorgehensweisen auf ältere und wahrscheinlich neuere Versionen, zumindest 4.2.1, übertragbar. Die betrachtete Version hat noch kleine Defizite nach der Umstrukturierung der Software, bei der u. a. die Swing-Oberfläche durch eine JavaFX-Oberfläche ersetzt wurde. Diese Defizite sind im Text angemerkt und könnten in neueren Versionen behoben sein. Ein Defizit ist die nicht vollständige Übersetzung der Menü-Punkte ins Deutsche. Da die Positionen der Menü-Punkte identisch sind, sollte dies kleine Probleme machen, auch an den Stellen, an denen statt der deutschen die englische Version in diesem Text beschrieben wird.

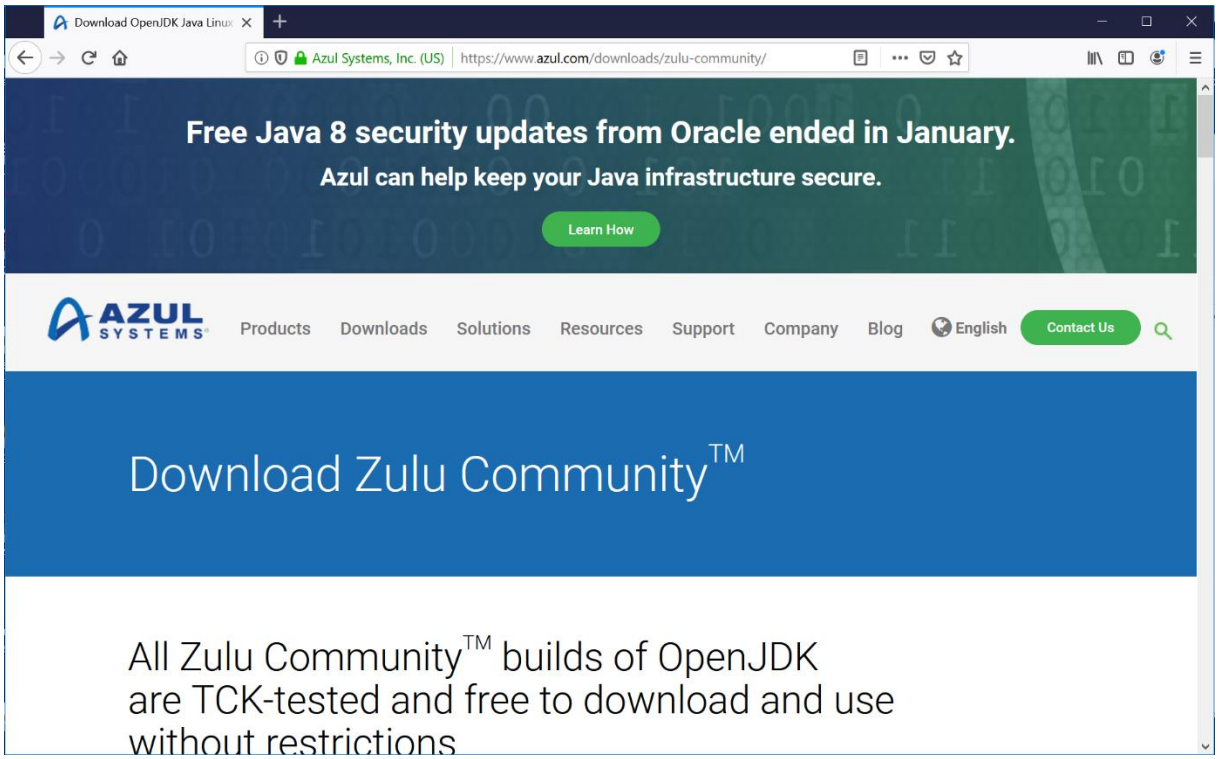



Abschließend sei angemerkt, dass in diesen Hinweisen die männliche Form von Personen und Substantiven vereinfachend und kompakter gewählt wird, auch wenn natürlich immer auch die weibliche und diverse Form von Personen und Substantiven mit gemeint ist. Eine notwendige Überarbeitung dieses Textes ist geplant.

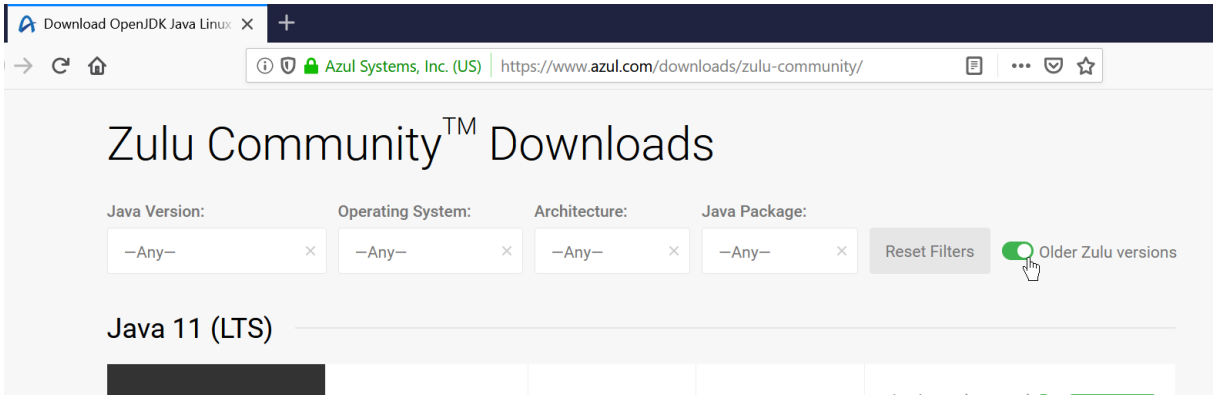
# 2 Installation von Java

Vorbemerkung: Wie alle Programmiersprachen entwickelt sich Java weiter, dabei werden meist neue Programmbibliotheken ergänzt oder hinzugefügt, die Funktionalität anbieten, die bisher in Java noch nicht vorhanden war. Weiterhin werden Befehle ergänzt, die es erfahrenen Programmierern erleichtern, etwas kürze Programme zu schreiben. Bis einschließlich der Version 8 liefen (praktisch) alle in anderen Versionen von Java geschriebenen Programme auch in dieser Version. Dies ist ab der Version 9 nicht mehr ganz der Fall, was einige Vorteile (modularere Software-Entwicklung, Auslieferung ohne vollständige Java Run Time) hat, aber langfristig kritische strategische Auswirkungen haben kann. Andere Programmiersprachen wie PHP und einige JavaScript-Frameworks haben deutlich massiver gegen diese Abwärtskompatibilität verstoßen und nicht wesentlich in ihrer Beliebtheit eingebüßt. Trotzdem ist zu erwarten, dass viele Projekte auch weit über 2019 hinaus auf Java 8 basieren. Weiterhin bleibt anzumerken, dass bis Java 8 zwischen den Wechsel der Hauptversionsnummer, z. B. zwischen Java 7 und Java 8 oft mehrere Jahre lagen, nach einer neuen Release-Strategie diese Nummer aber durchaus mehrfach im Jahr verändert werden kann. Für Programmieranfänger sei angemerkt, dass alles Lehrmaterial welches ab Java 7 (teilweise Java 5) entstanden ist, relativ problemlos weiter genutzt werden kann.
Eine weitere Verwirrung um Java wurde in 2018 von Oracle initiiert, indem versucht wird kommerziell direkt mit der Sprache Geld zu verdienen, ein über 20 Jahre unnötiger Ansatz. Genauer müssen kommerzielle Kunden für die Nutzung der Java Virtual Machine (JVM) zahlen, die zur Ausführung von Java-Programmen benötigt wird. Für Studierende und Entwickler bleibt die Nutzung (zur Zeit ?) frei. Da der Java Code Open Source ist, kann jeder sein eigenes Java übersetzen und frei zur Verfügung stellen. Dies machen einige Anbieter, deren Java frei nutzbar auch für kommerzielle Nutzer bleibt. Um diesen Ansatz zu unterstützen wird hier eine solche Java-Variante installiert. Wichtig ist, dass die daraus resultierenden Programme unabhängig von der genutzten Java-Installation sind.

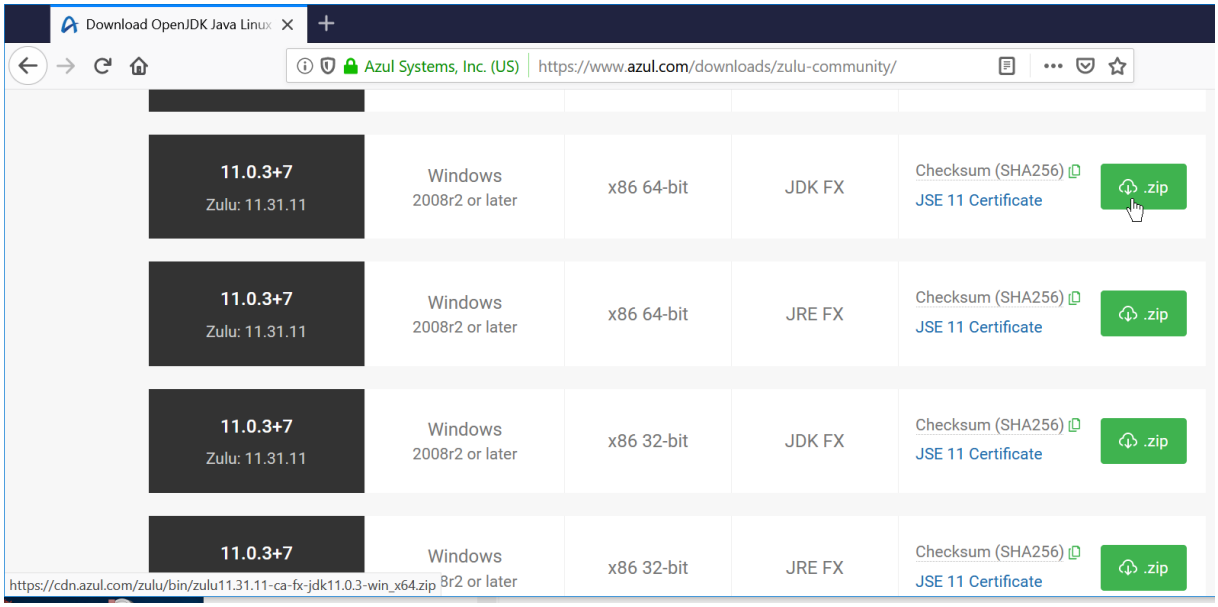
Java wird direkt von der Seite von Azul Systems https://www.azul.com/downloads/zulu-community/ heruntergeladen, auf der man meist reicht weit nach unten scrollen muss.



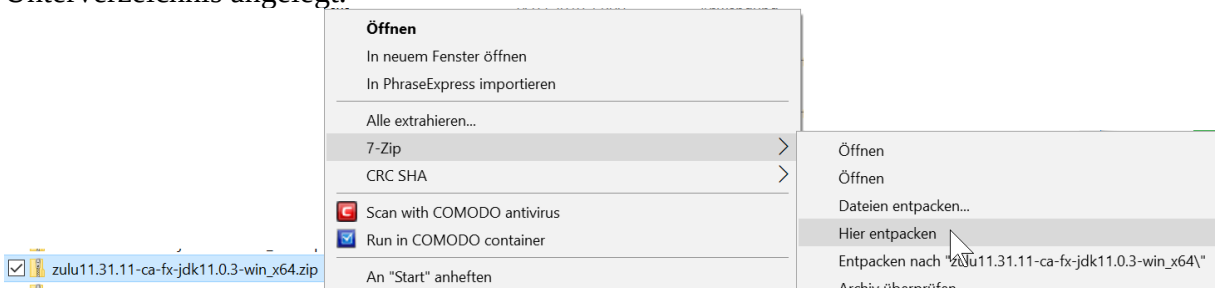

Es wird eine LTS (Long Term Service)-Version, hier 11, genutzt. Die Installation für andere Varianten verläuft identisch. Genauer soll eine Version mit bereits eingebauter Unterstützung von Java FX (jetzt JFX) geladen werden. Java FX ist eine Java-Bibliothek zur Entwicklung graphischer Oberflächen, die bis Java 8 Teil der Standard-Distribution war und bei anderen Versionen etwas aufwändig nachinstallieret werden muss. Java FX wird u. a. bei BlueJ und als Hilfsmittel in einigen Praktikumsaufgaben genutzt. Da Java FX nicht in allen Versionen von Zulu gepflegt wird, muss gegebenenfalls nach einer etwas älteren Version gesucht werden. Bei der betrachteten Webseiten-Variante ist dazu der Schieber „Older Zulu versions“ nach rechts zu setzen.



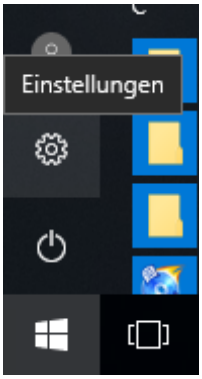

Es wird eine Java 11 LTS-Version mit JFX gesucht und heruntergeladen. Die Web-Seite zeigt auch, dass hier noch eine Version für 32-Bit-Systeme (x86 32-bit) angeboten wird. Eventuell funktioniert noch folgender Link: https://cdn.azul.com/zulu/bin/zulu11.31.11-ca-fx-jdk11.0.3-win_x64.zip oder aktueller: https://cdn.azul.com/zulu/bin/zulu16.32.15-ca-fx-jdk16.0.2-win_x64.zip. Wichtig ist, dass ein JDK, ein Java Development Kit, heruntergeladen wird, da nur damit die eigene Entwicklung möglich ist. Mit einer JRE, einer Java Runtime Environment, können nur fertige Java-Programme ausgeführt werden.



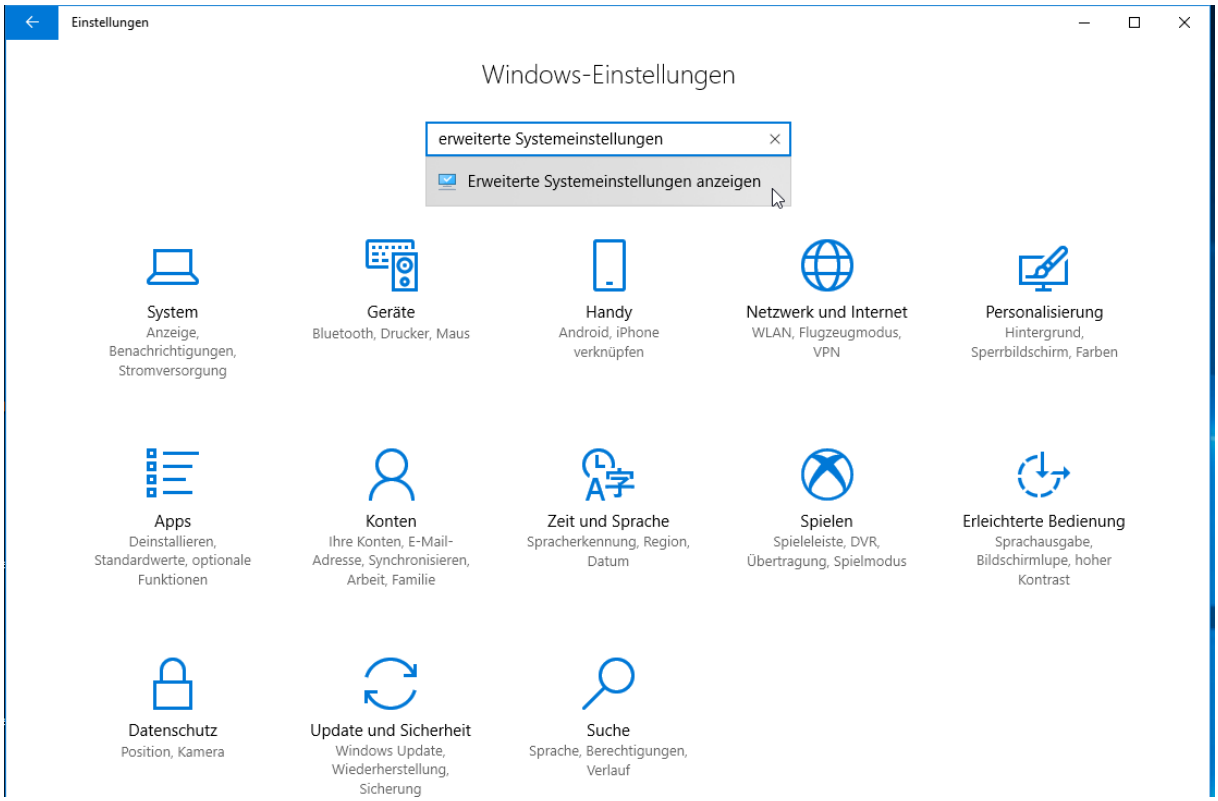

Die Zip-Datei kann an einem fast beliebigen Ort mit Schreibrechten mit „Extract here“ ausgepackt werden. Der Ort ist fast beliebig, da er wie in den meisten Fällen sinnvol, kein Leerzeichen oder Sonderzeichen im Namen enthalten sollte. Es wird automatisch ein Unterverzeichnis angelegt.



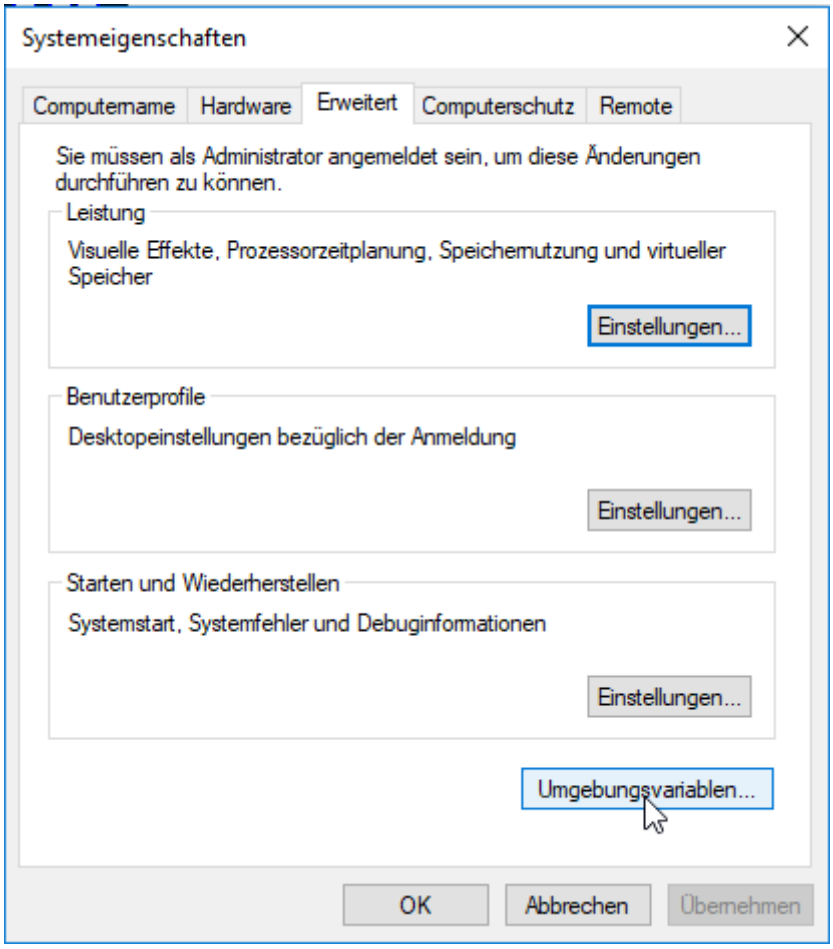

Zum Abschluss müssen einige Systemvariablen auf das Installationsverzeichnis gesetzt werden. Bei Windows 10 beginnt ein Weg mit einem Klick auf dem Start-Icon und der Auswahl „Einstellungen“.



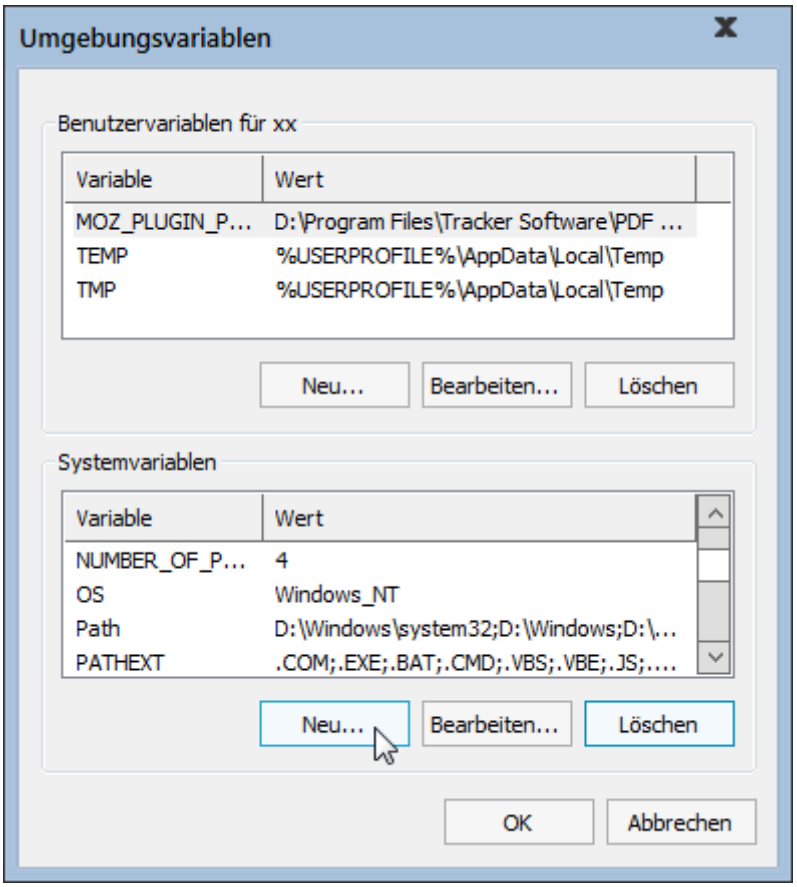

Im oberen Suchfenster wird „erweiterte Systemeinstellungen“ eingeben und darunter dann der Vorschlag mit einem Links-Klick ausgewählt.



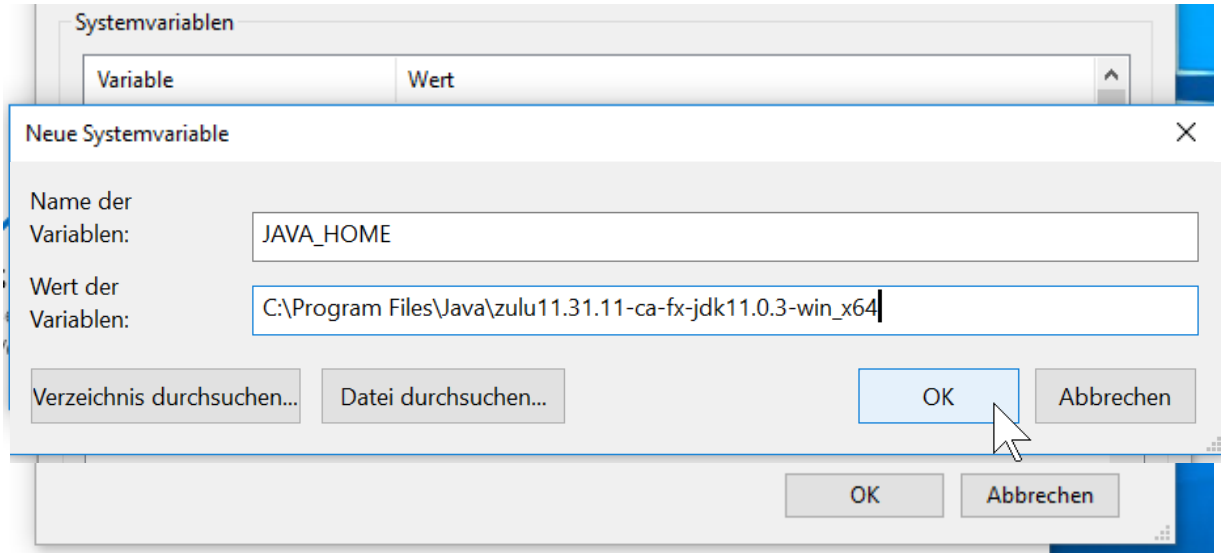

Es wird auf „Umgebungsvariablen“ geklickt.



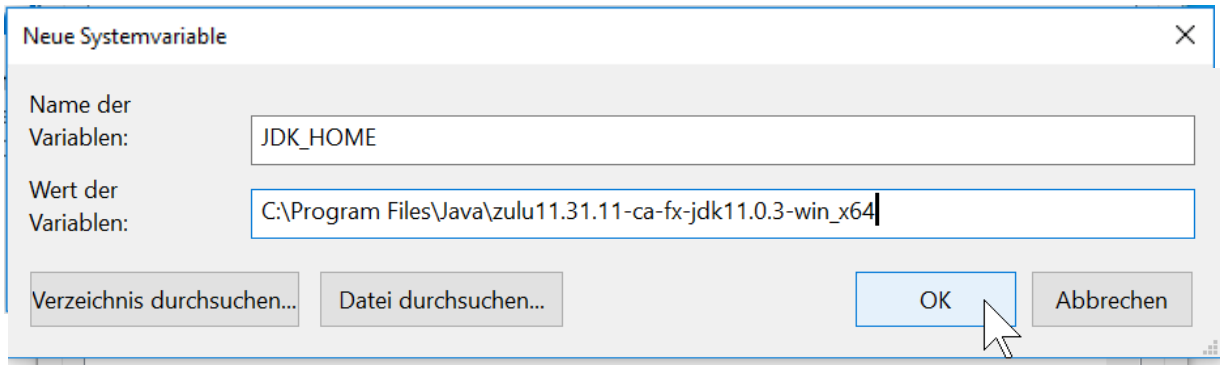

Administratoren des Systems nutzen den Knopf „Neu...“ unter Systemvariablen, andere Nutzer den Knopf „Neu…“ unter Benutzervariablen.



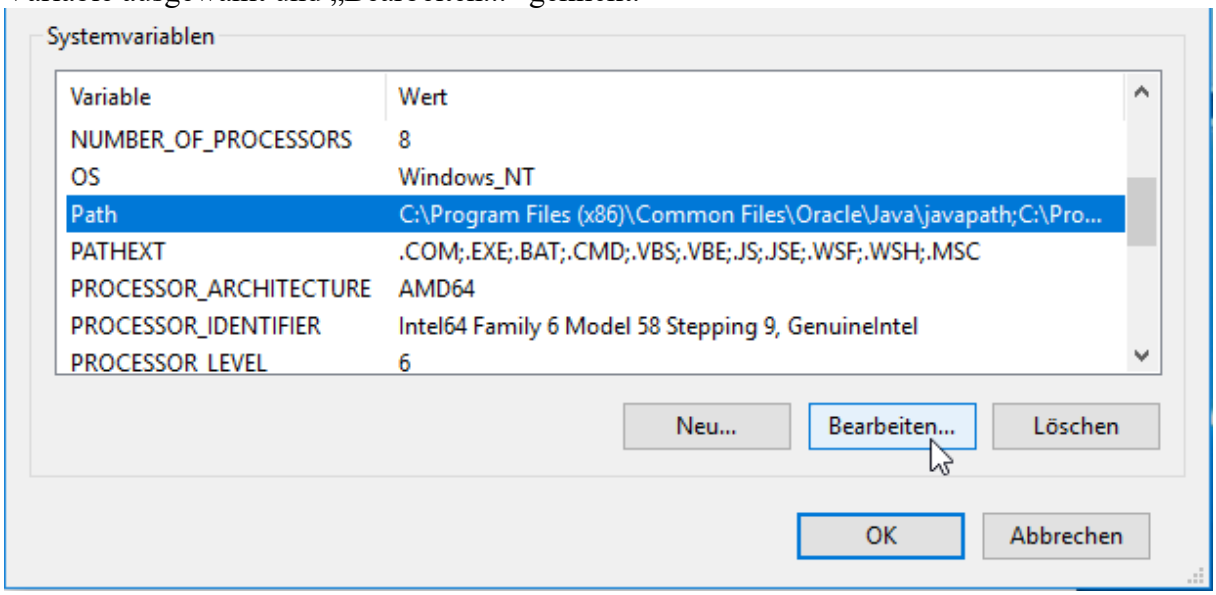

Hier werden der Variablenname sowie der Pfad zum installierten JDK eingetragen und die neue Variable mit „OK“ bestätigt. Es ist zu beachten, dass wahrscheinlich eine aktuellere Variante des JDK installiert wurde und so der letzte Ordnernamen etwas anders aussehen wird.



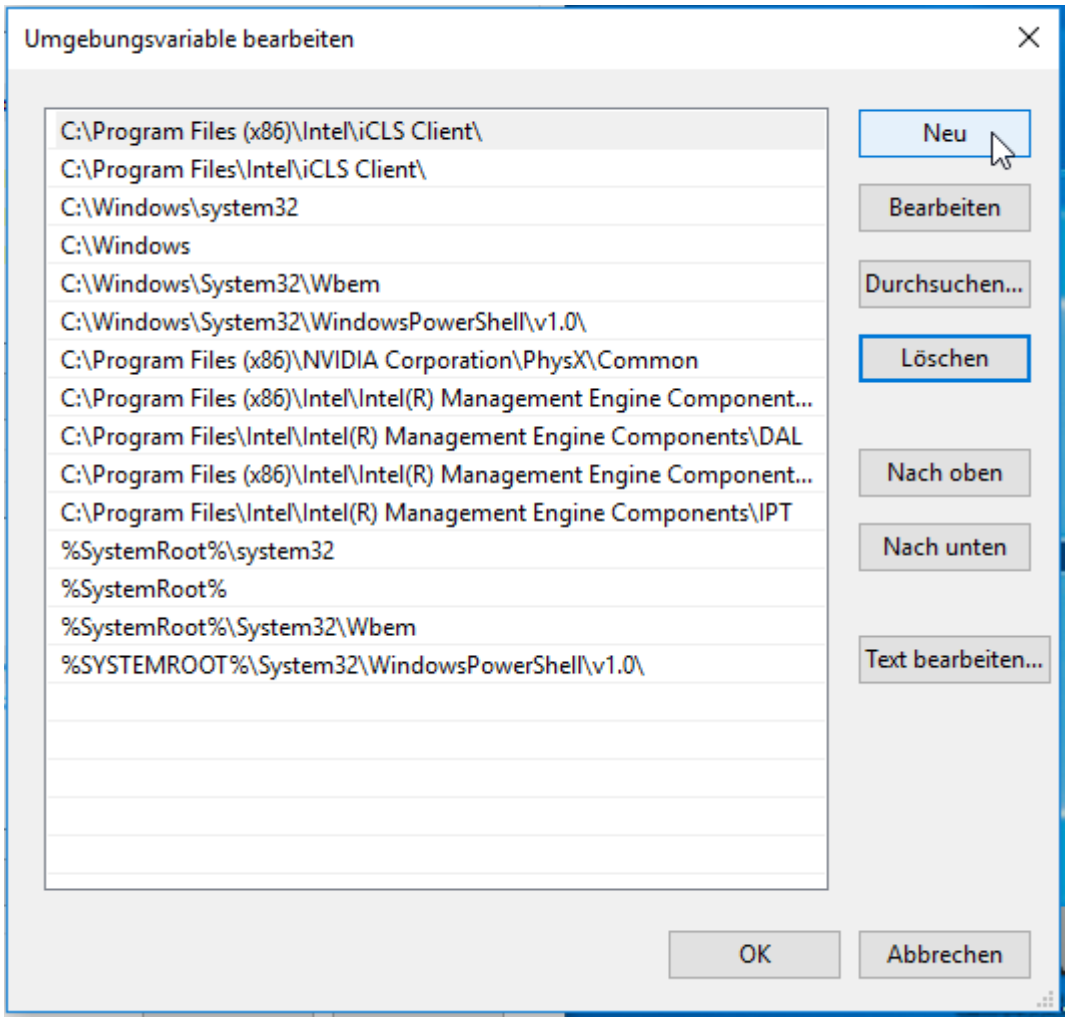

Weiterhin ist es sinnvoll, Variablen JDK_HOME und JRE_HOME mit genau dem gleichen Wert anzulegen.




Abschließend muss Java in die PATH-Variable eingetragen werden. Dazu wird die Path-Variable ausgewählt und „Bearbeiten...“ geklickt.

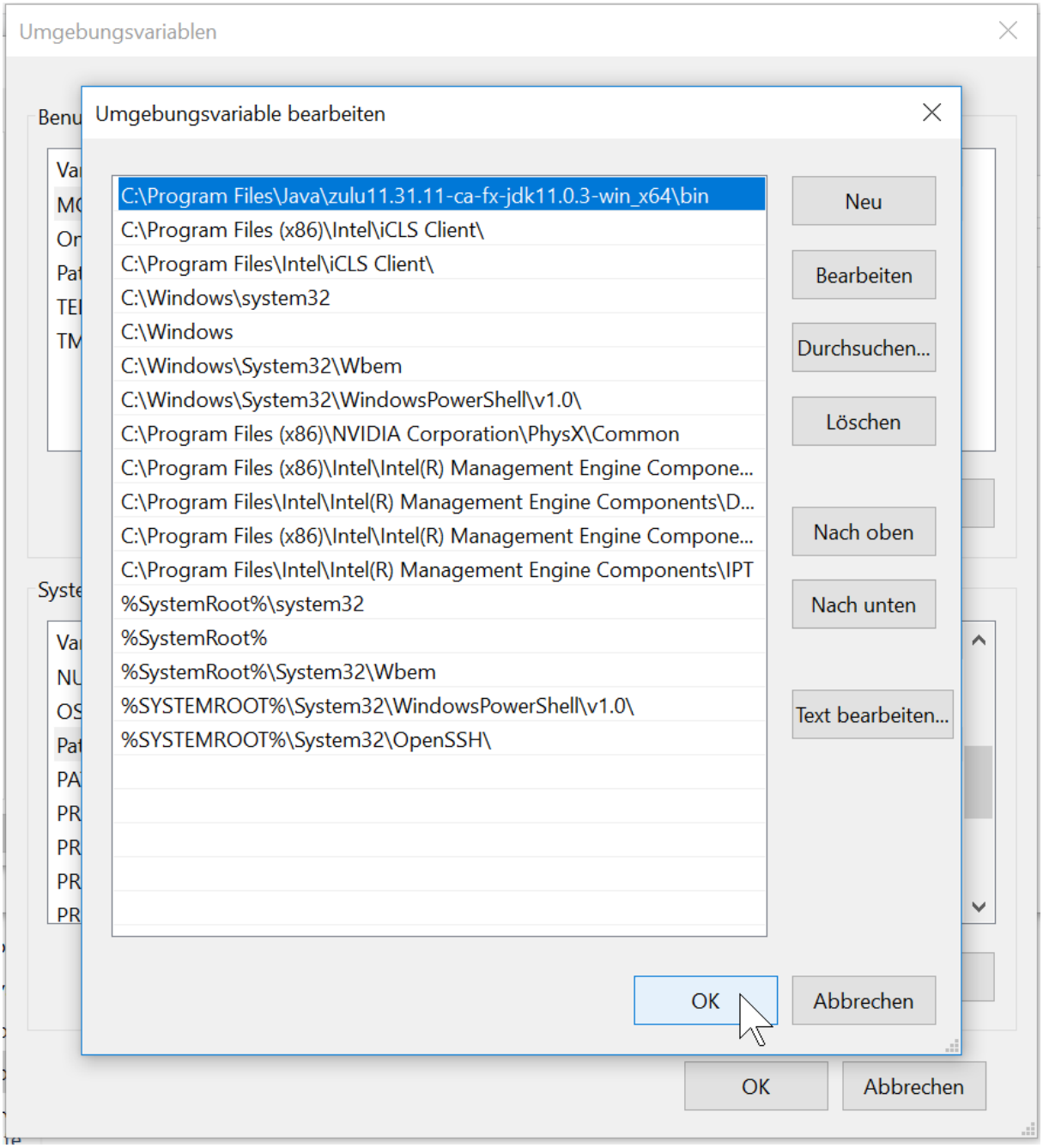



Es wird auf „Neu“ oben geklickt.

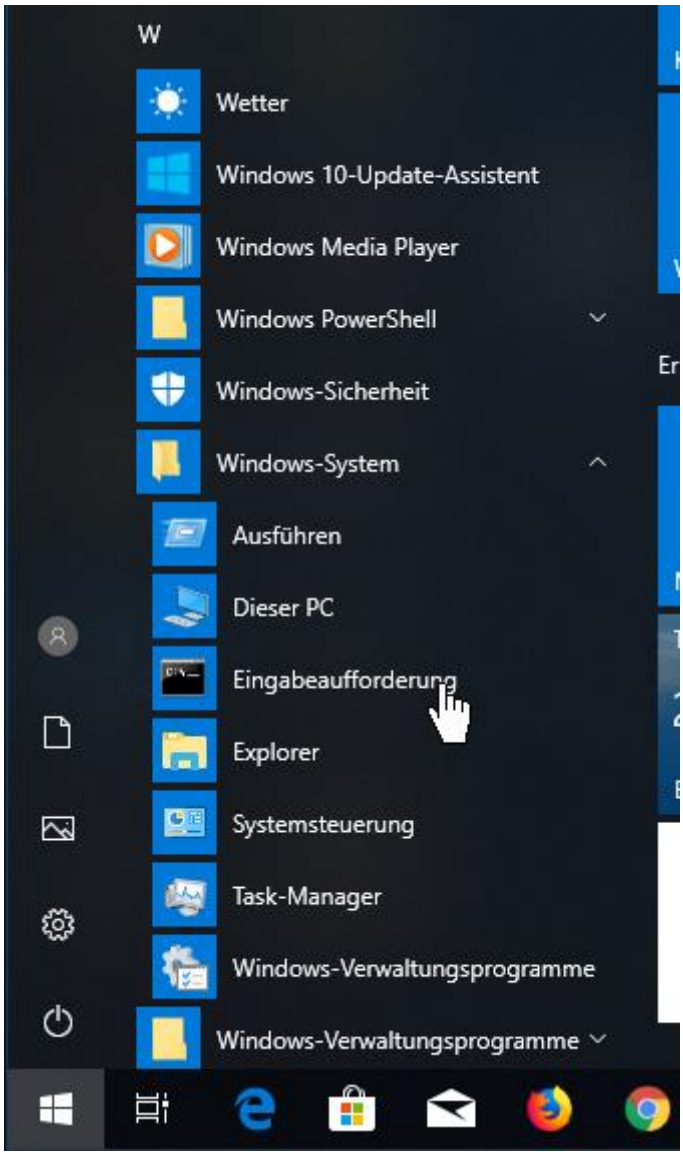



Es wird der Pfad zum bin-Verzeichnis `C:\Program Files\Java\zulu11.31.11-ca-fx-jdk11.0.3-win_x64\bin` eingetragen.

Danach kann der ausgewählte Pfad durch mehrfaches Klicken von „Nach oben“ in die obere Zeile geschoben werden, was allerdings für die Nutzung nicht relevant ist. Die Eintragung wird jeweils mit „OK“ bestätigt.




Zur Überprüfung der Installation kann ein Konsolenfenster genutzt werden. Dies ist z. B. über einen Klick auf das Windows-Icon links-unten, dem Scrollen zum Buchstaben „W“ bei den installierten Programmen, dem Aufklappen von „Windows-System“ und der Auswahl von „Eingabeaufforderung“ erreichbar.



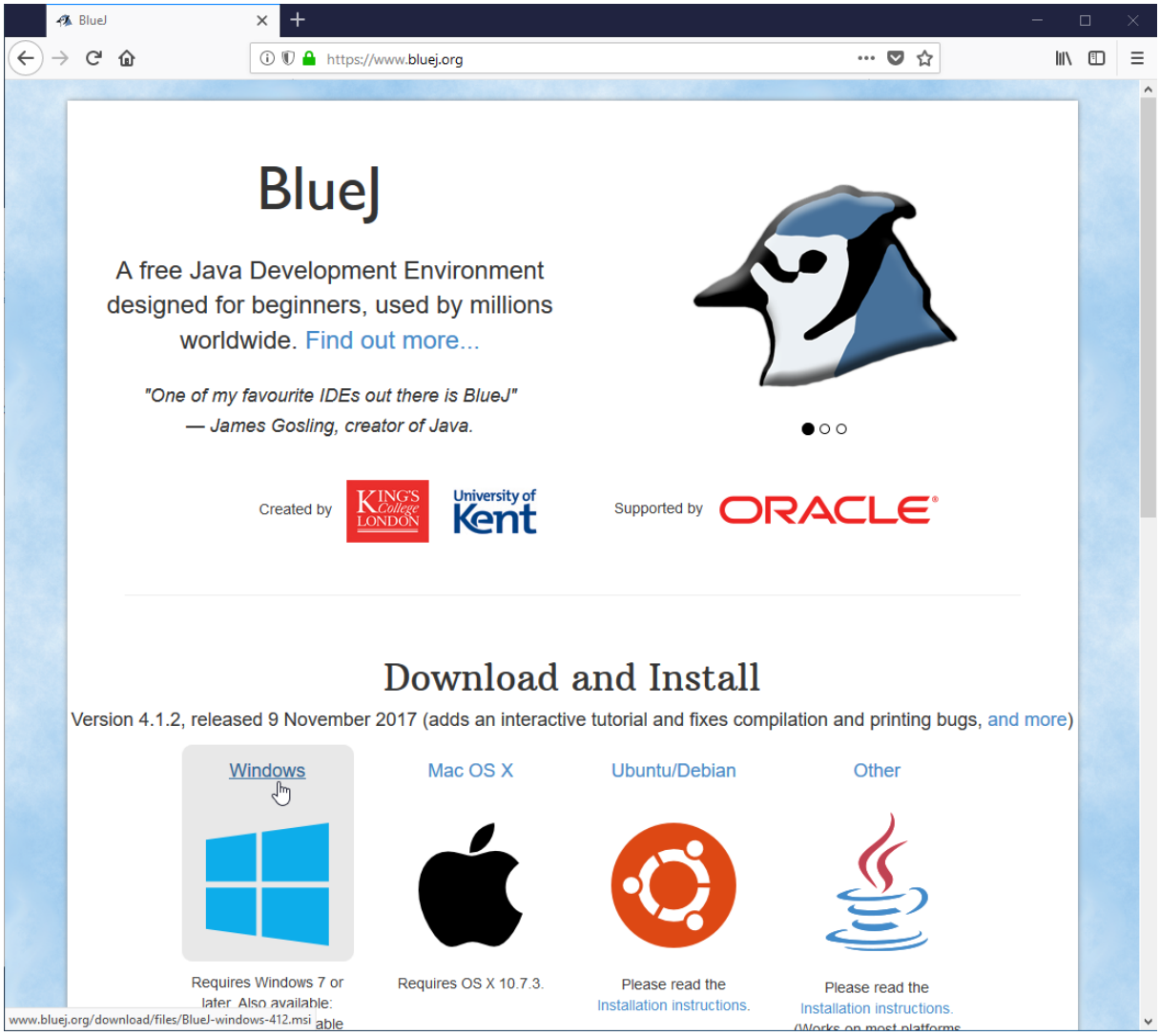

In dem Fenster wird `javac -version` eingegeben, was zur angezeigten Ausgabe, evtl. mit höherer Versionsnummer, führen muss. Java ist dann korrekt installiert.

```
Administrator: Eingabeaufforderung
Microsoft Windows [Version 10.0.17134.885]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\x>javac -version
javac 11.0.3

C:\Users\x>_
```

# 3 Installation von BlueJ

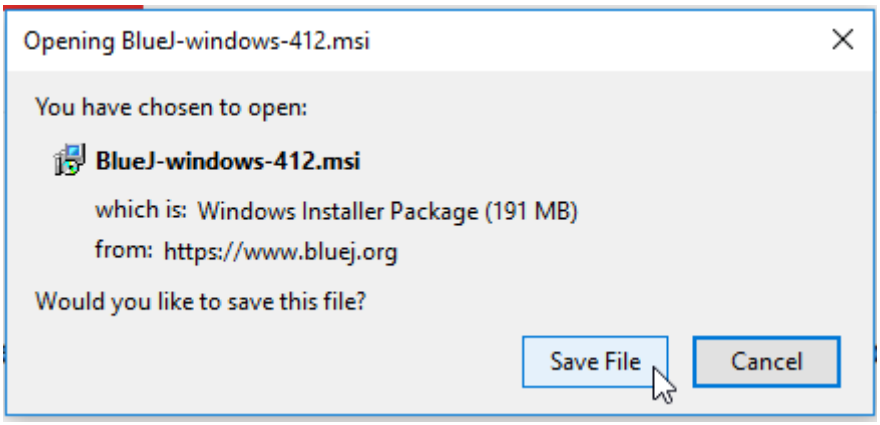
Der folgende Installationsprozess beschreibt eine Installationsmöglichkeit unter Windows, da dies die in der Hochschule angebotene Variante ist. Einige Abbildungen beziehen sich auf Version 4.1.2, sind aber genauso für nachfolgende Versionen, zumindest 4.2.1 und 5.0.1, nutzbar. Da BlueJ eine eigene Java-Installation beinhaltet, könnte auf die Java-Installation verzichtet werden, wenn sonst kein mit Java entwickeltes Programm genutzt wird.

Das Programm kann unter https://www.bluej.org/ heruntergeladen werden, es wird auf den Windows-Link geklickt.

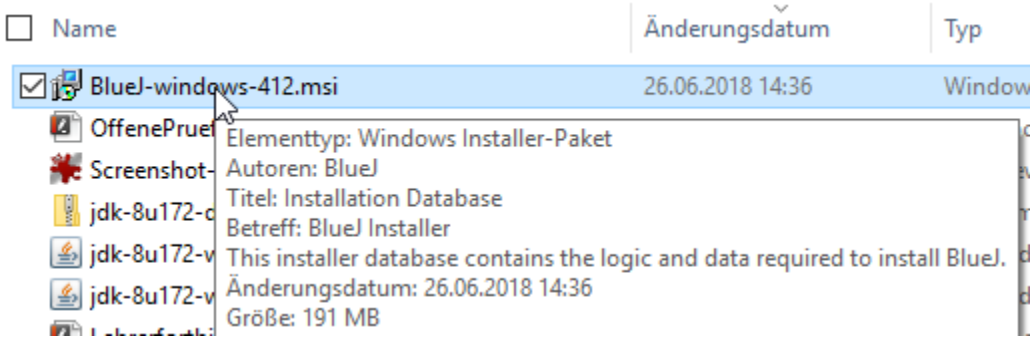



Unterhalb des Links befindet sich ein Link zu einer Standalone-Version, die nur ausgepasckt werden muss und sich so z. B. für den portablen Einsatz eignet. Dieses spannende Thema, das für eine Erstinstallation allerdings irrelevant ist, wird in „8 Portable Version von BlueJ“ getrennt behandelt.

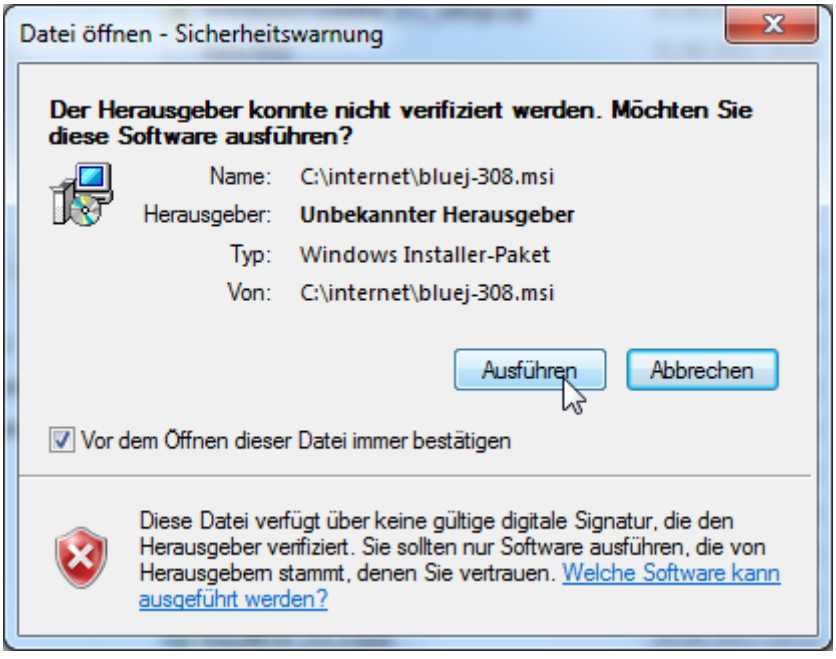
Der Link zur Datei wird angeklickt und die zugehörige Datei heruntergeladen.



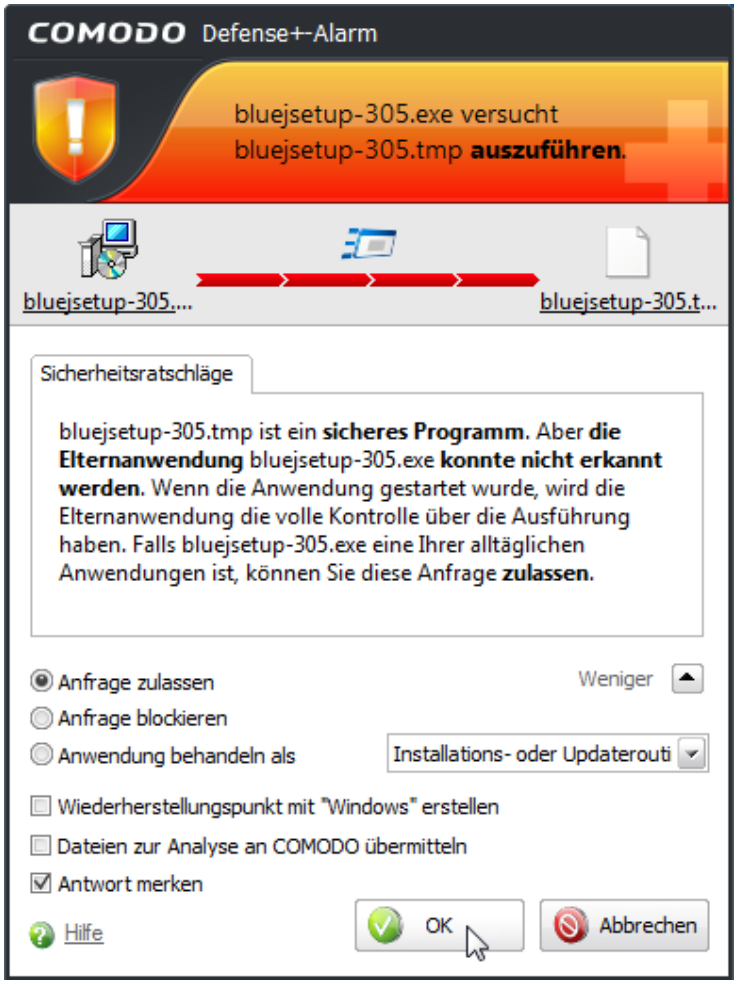

Die Installation startet über einen Doppelklick auf der heruntergeladenen Datei.



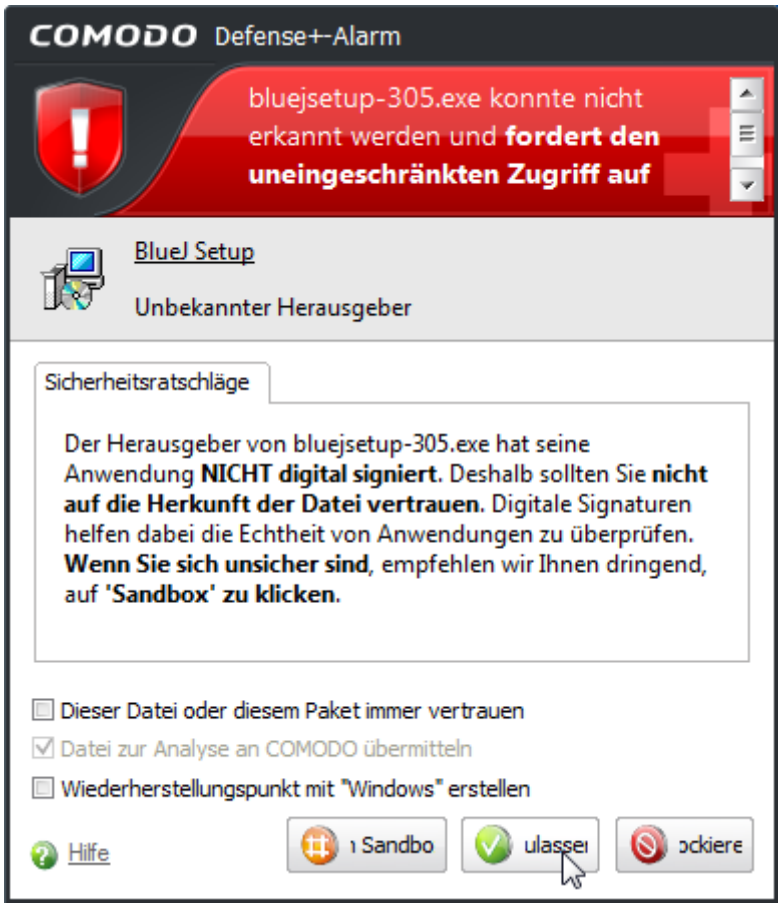

Bei der Installation könnte sich die Firewall oder ein anderes Schutzprogramm melden, bei denen dann die Erlaubnis zur Installation erteilt werden muss. Eine Beispielmeldung kann wie folgt aussehen, hängst natürlich von der genutzten Software und deren Einstellungen ab.



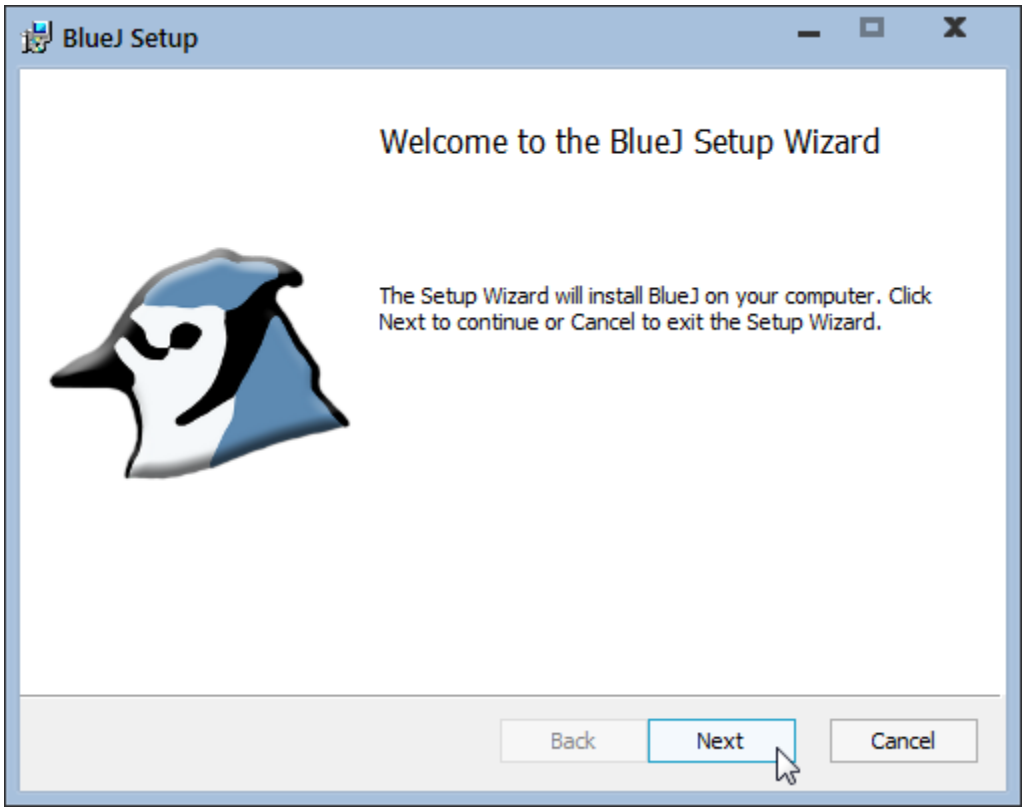
Auch bei der Firewall muss gegebenenfalls eine Installationserlaubnis eingetragen werden.



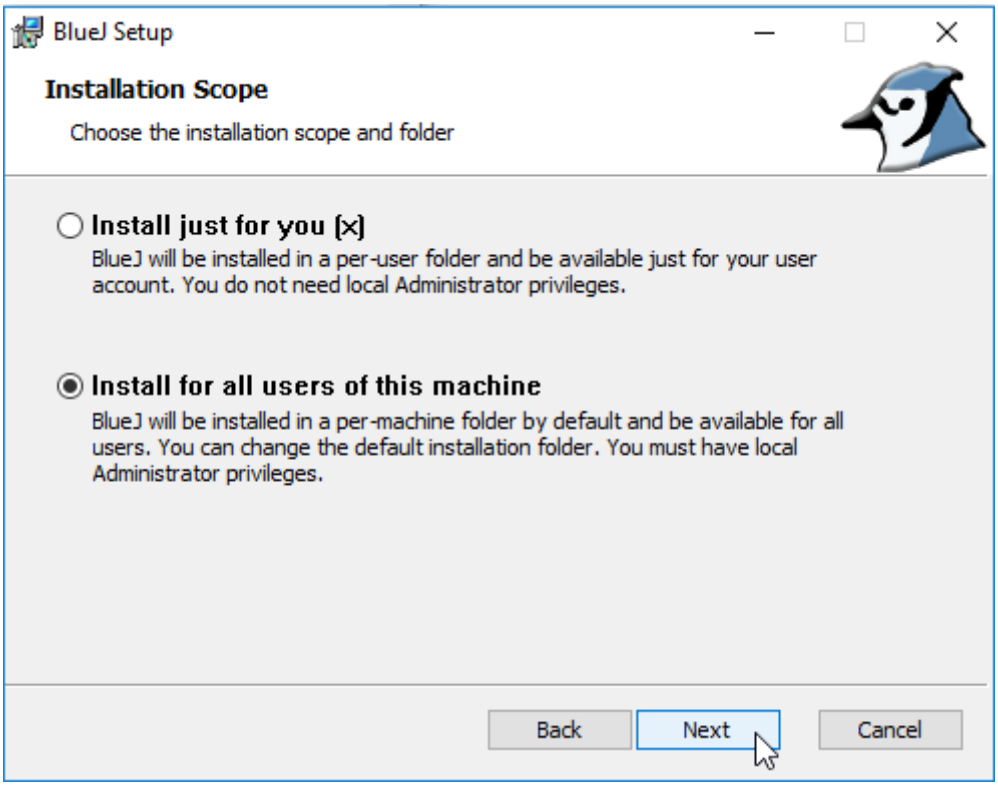

Die hier privat genutzte Firewall benötigt eine weitere Bestätigung, dass das Programm im normalen Modus installiert werden soll.



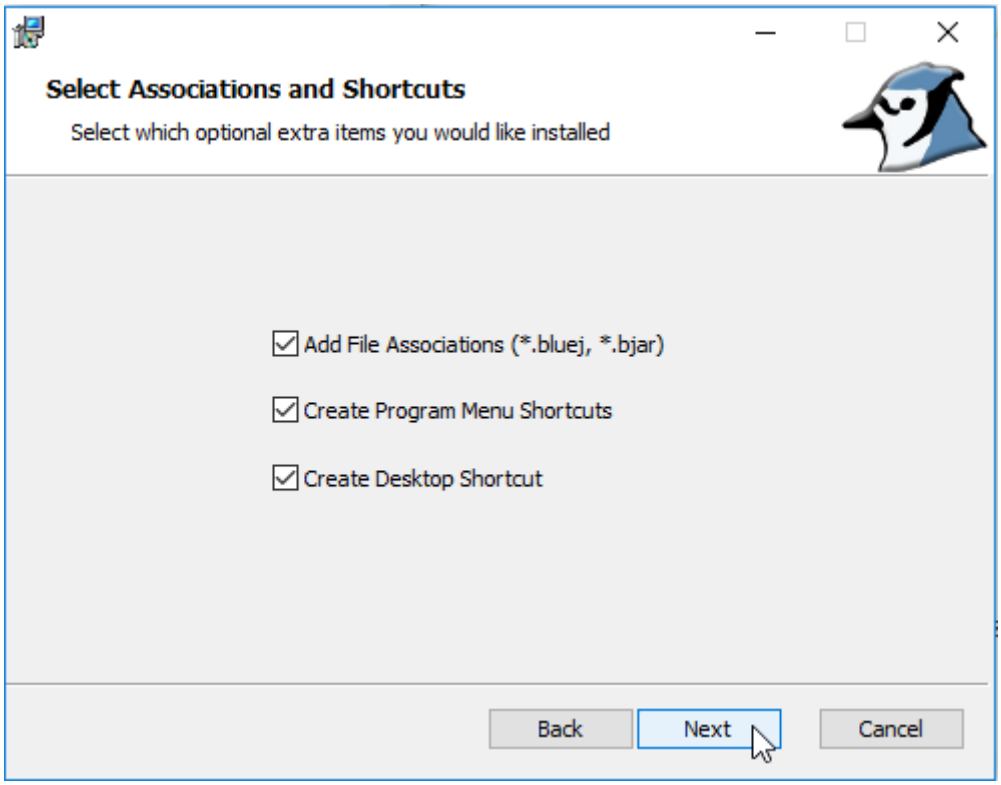

Die Startmeldung wird gelesen und mit „Next“ der Installationsprozess fortgesetzt.



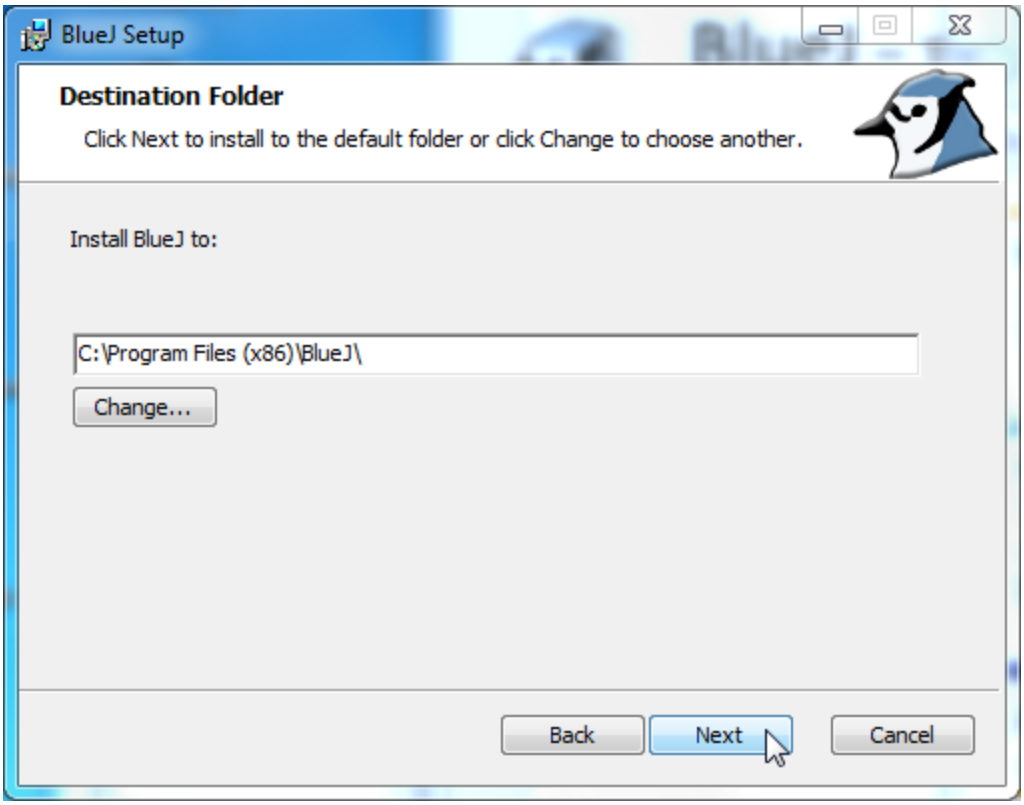

Die Installationsart kann beliebig gewählt und dann „Next“ geklickt werden.



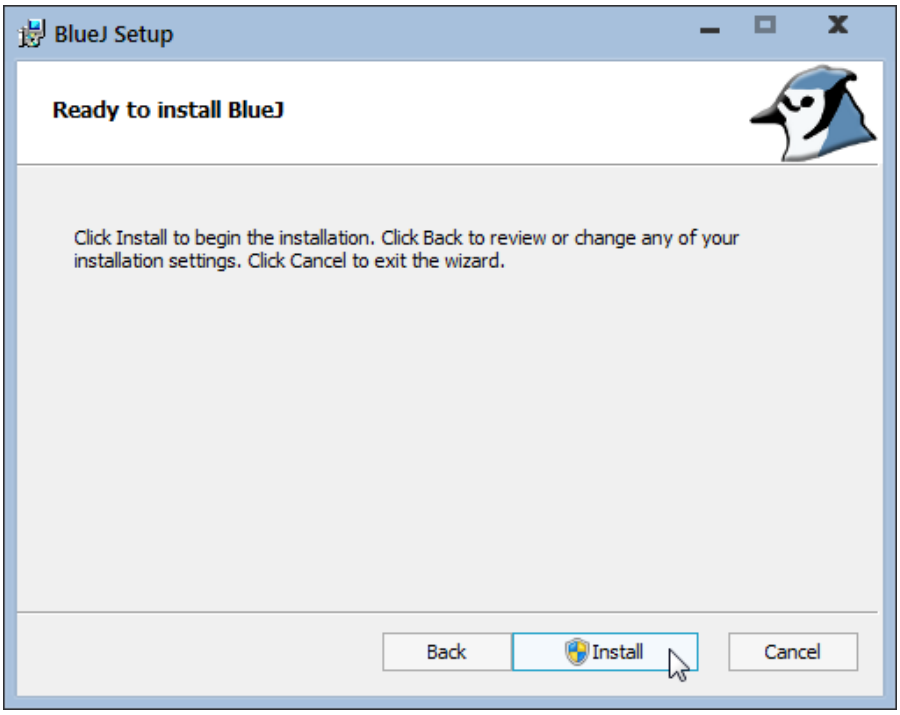
Die erste Einstellung führt zu einer Verknüpfung von zu BlueJ passenden Dateien mit BlueJ, so dass bei einem Doppelklick auf Dateien mit dieser Endung diese Dateien mit BlueJ geöffnet werden. Der Haken sollte deshalb stehen bleiben. Die anderen Haken können nach eigenen Wünschen gesetzt oder gelöscht werden. Es wird „Next“ geklickt.



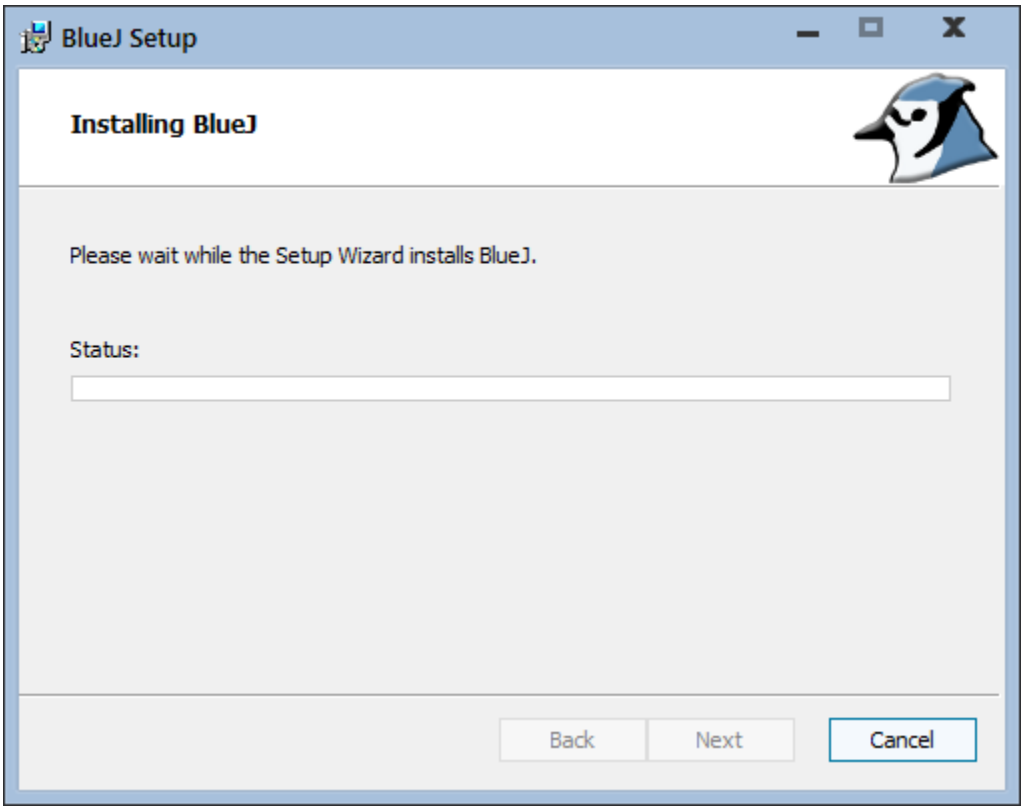

Man wird dann aufgefordert, ein Verzeichnis anzugeben, in dem BlueJ installiert werden soll, meist ist das vorgeschlagene Verzeichnis sinnvoll. Danach wird „Next“ geklickt.



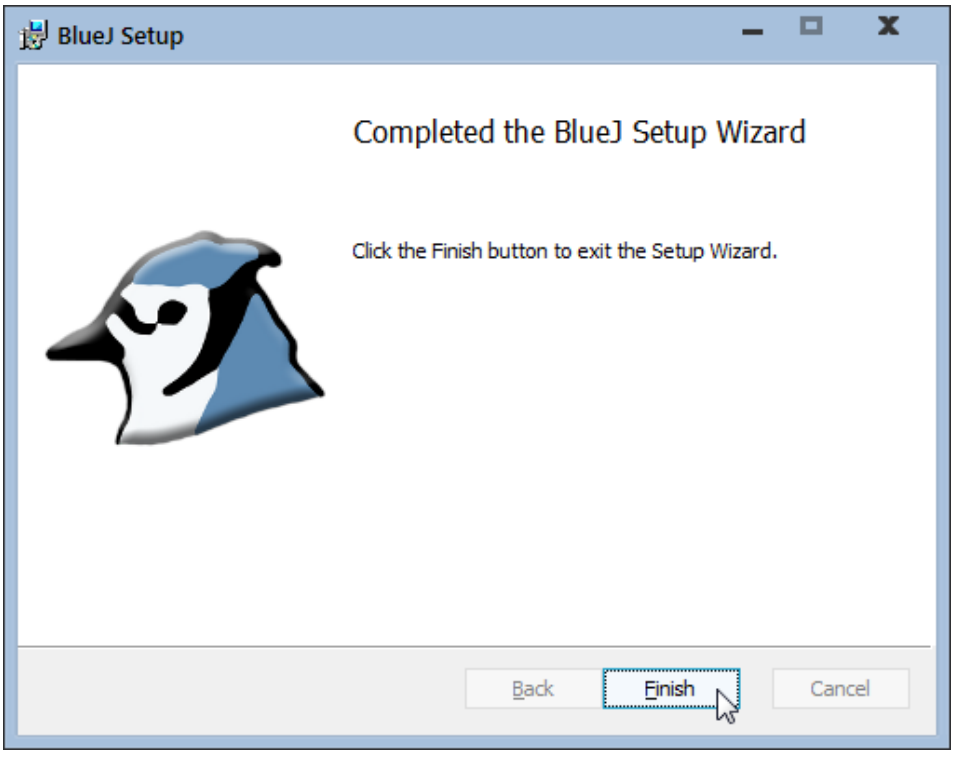

Danach kann die eigentliche Installation mit Klicken des „Install“-Knopfs begonnen werden. Falls die Installation nicht als Administrator durchgeführt wird, muss gegebenenfalls die Installation erlaubt werden.



Die Installation dauert etwas, gegebenenfalls sind Freigaben bei einem lokal installierten Systemüberwachungsprogramm notwendig.




Man wird über die abgeschlossene Installation informiert, die mit einem Klick auf „Finish“ beendet wird.



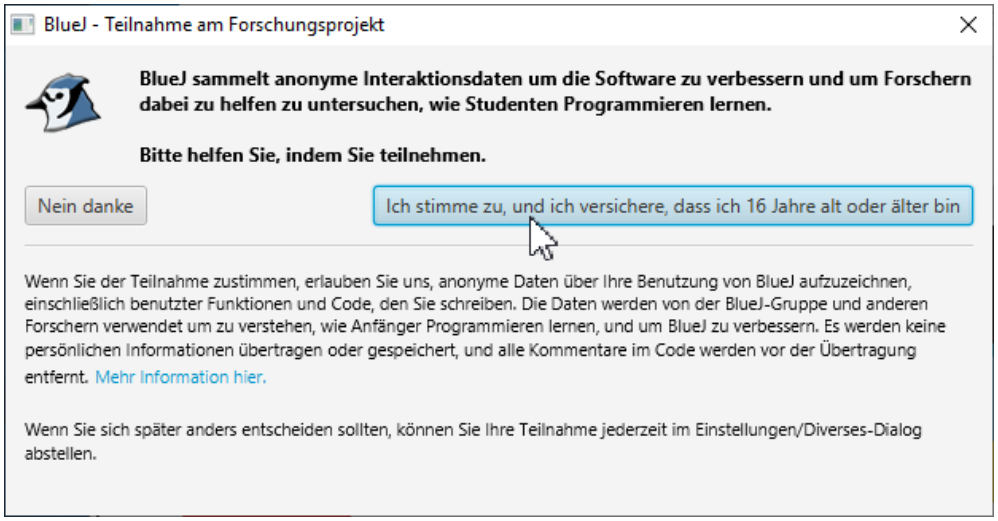
Nach erfolgreicher Installation erscheint, insofern es bei den Einstellungen zugelassen wurde, auf dem Desktop folgendes Icon.

# 4 Erste Nutzung von BlueJ

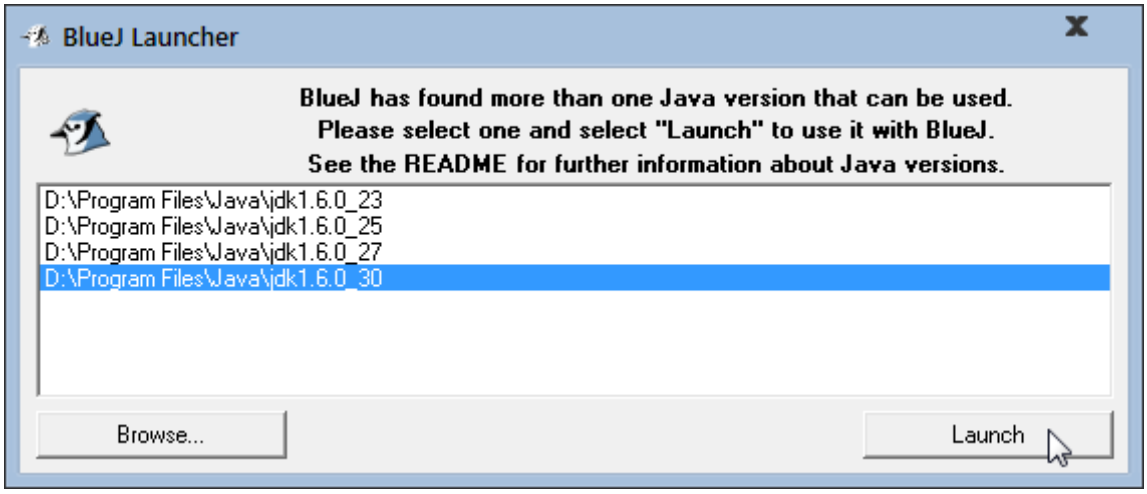
## *4.1 Anlegen eines Projektes*

Bevor mit BlueJ gearbeitet werden kann, ist auch zum Experimentieren zunächst ein Projekt anzulegen. Dazu wird BlueJ z. B. durch einen Doppelklick auf das Icon gestartet.

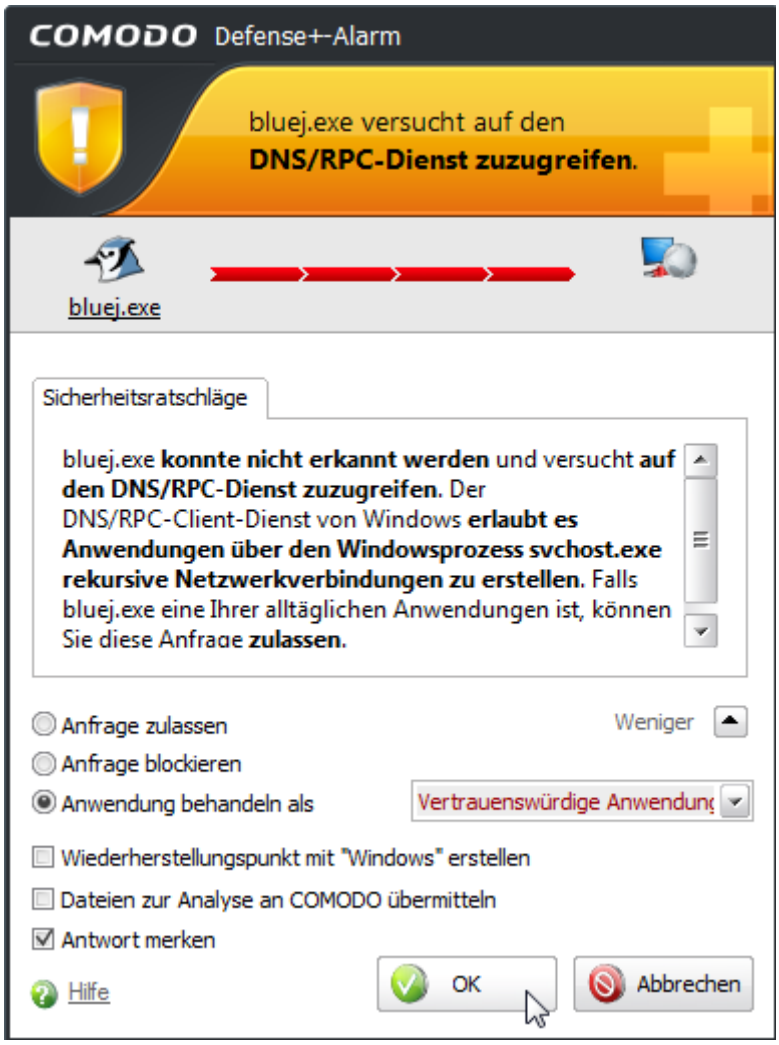

Die einmalig am Anfang gestellte Frage kann nach eigenen Interessen beantwortet werden.



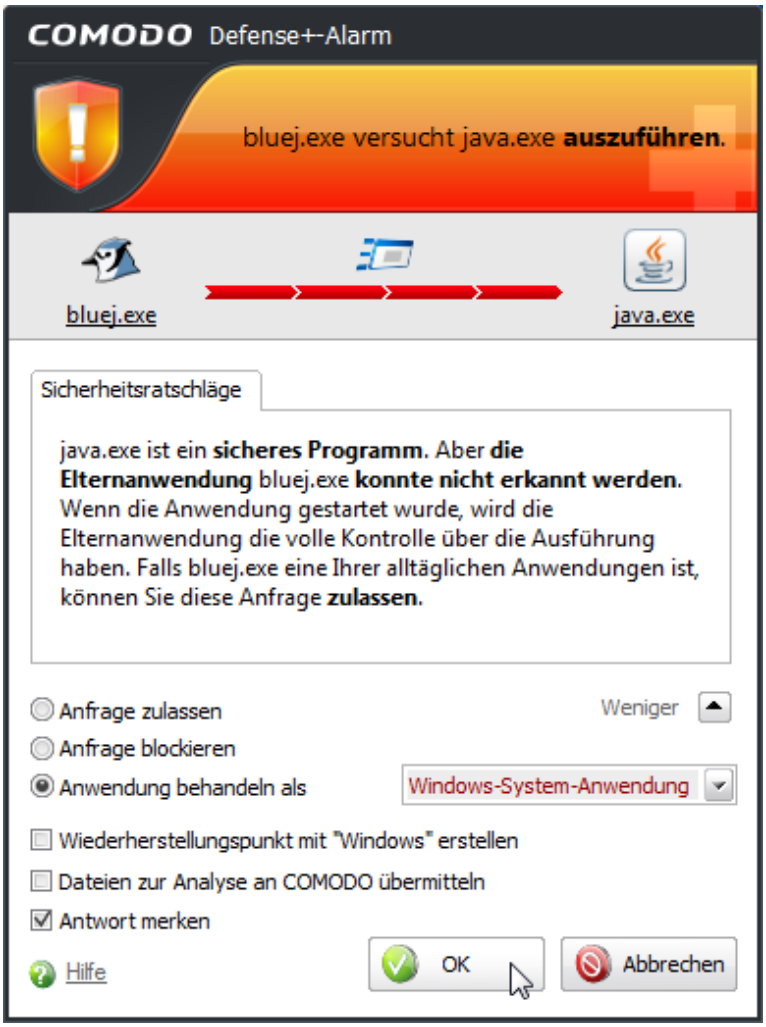

Beim ersten Start prüft BlueJ, ob eine Java-Installation gefunden wird. Sind mehrere vorhanden, muss jetzt eine, typischerweise die neueste, ausgewählt und „Launch“ geklickt werden. Ist nur genau eine passende Java-Version vorhanden, wird diese genommen und das folgende Fenster erscheint nicht.



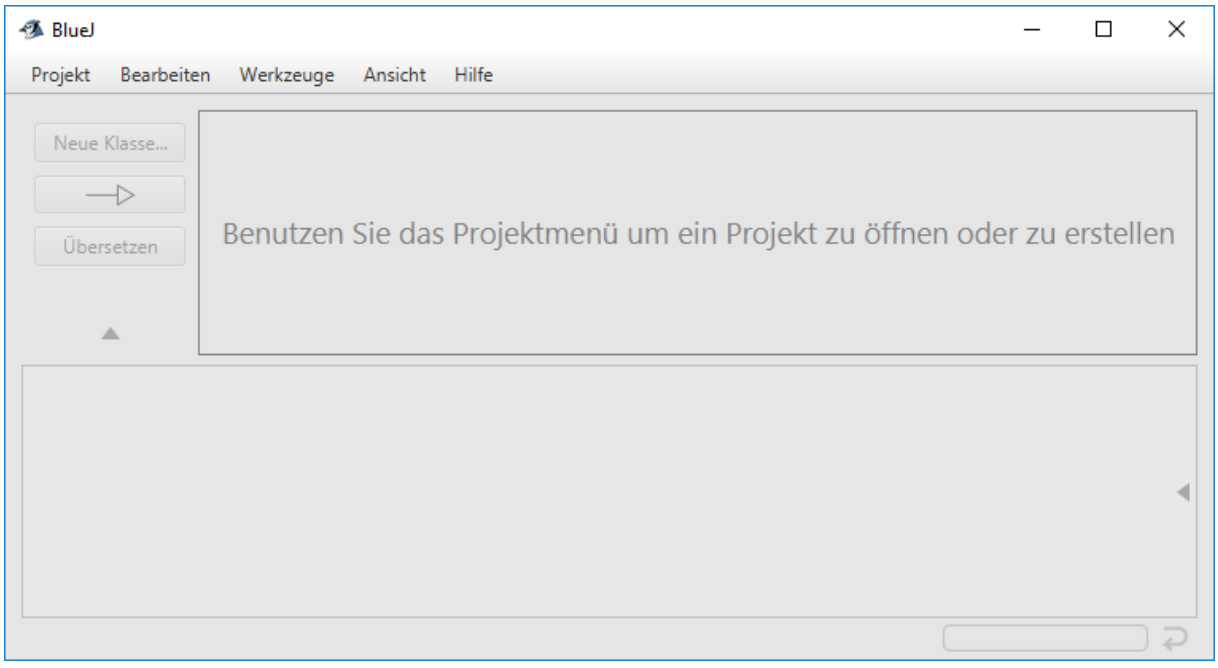
Abhangig von der Rechnerinstallation muss die Software zuerst von der Firewall die Erlaubnis zum Starten bekommen.



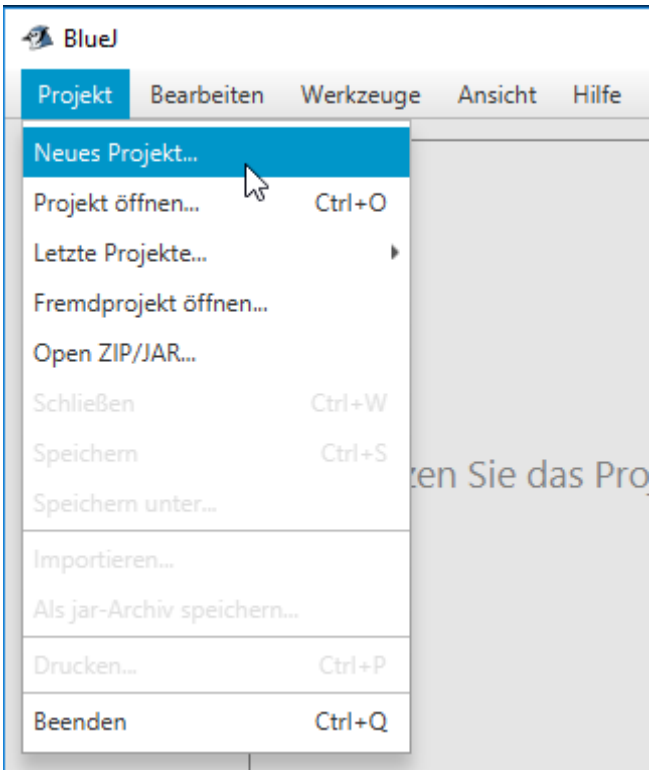

Weiterhin muss der Start von Java erlaubt werden, da BlueJ selbst in Java geschrieben ist und auf die Java-Installation zugreift.




Danach zeigt sich eine recht einfache, aufgeräumte Entwicklungsumgebung.

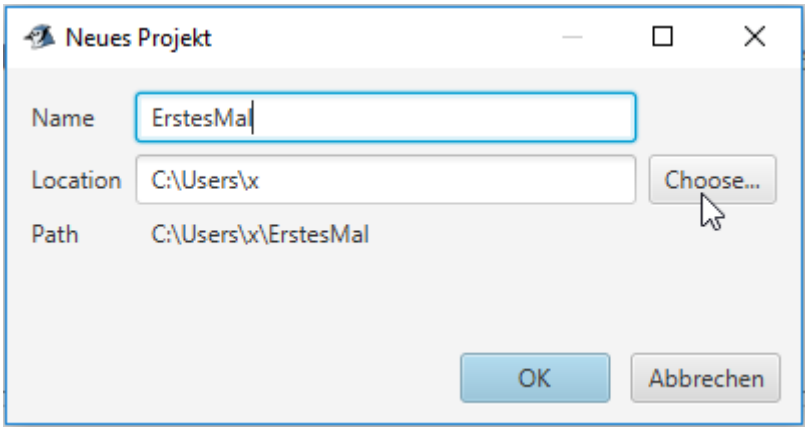


Die gesamte Entwicklung findet in BlueJ in getrennten Projekten statt, so dass z. B. pro Aufgabe ein eigenes Projekt anlegt wird. Ein neues Projekt entsteht über „Projekt -> Neues Projekt…“.

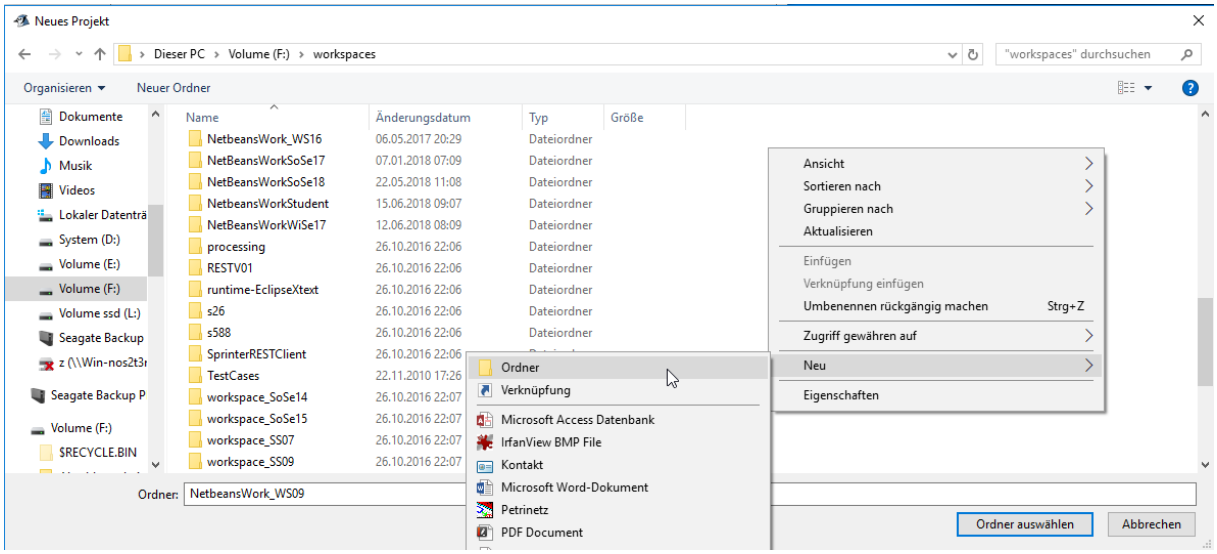



Spätestens jetzt ist zu planen, wie Projekte verwaltet werden sollen, da theoretisch an beliebigen Orten liegen können. Im konkreten Fall wird ein Ordner „BlueJWork“ angelegt, in dem sich alle Projekte, dann in eigenen Ordnern, befinden sollen. Der Ordner „BlueJWork“ kann vor der Nutzung oder bei der ersten Nutzung in BlueJ erstellt werden.

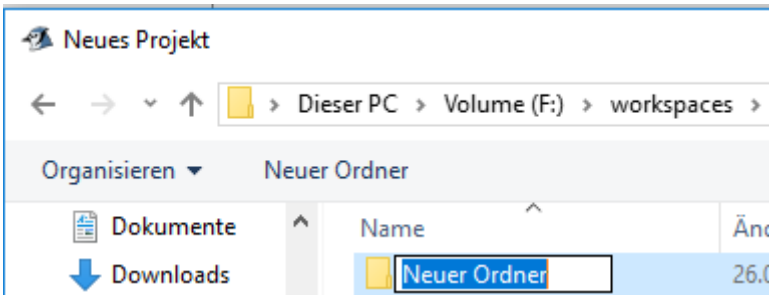
Zuerst wird ein eindeutiger und noch nicht vorhandener Projektname gewählt, hier „ErstesMal“. Dann wird die Auswahl des Speicherortes des Projektverzeichnisses mit einem Klick auf „Choose…“ begonnen.



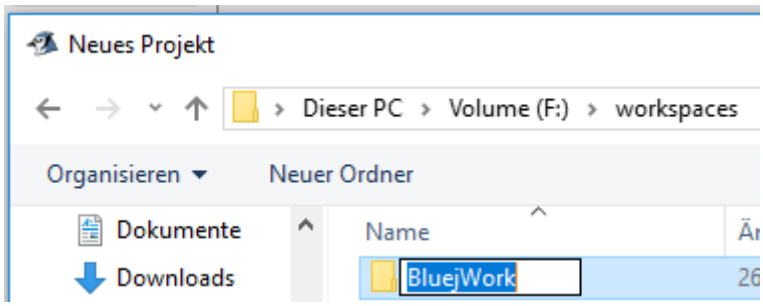

Im sich öffnenden Dateibrowser sollte zu einem Verzeichnis manövriert werden, in dem das Projekt anlegt wird. Dieses Verzeichnis ist auch direkt in diesem Fenster erstellbar, in dem z. B. im rechten Fensterteil einen Rechtsklick gemacht und „Neuer Ordner“ auswählt wird.



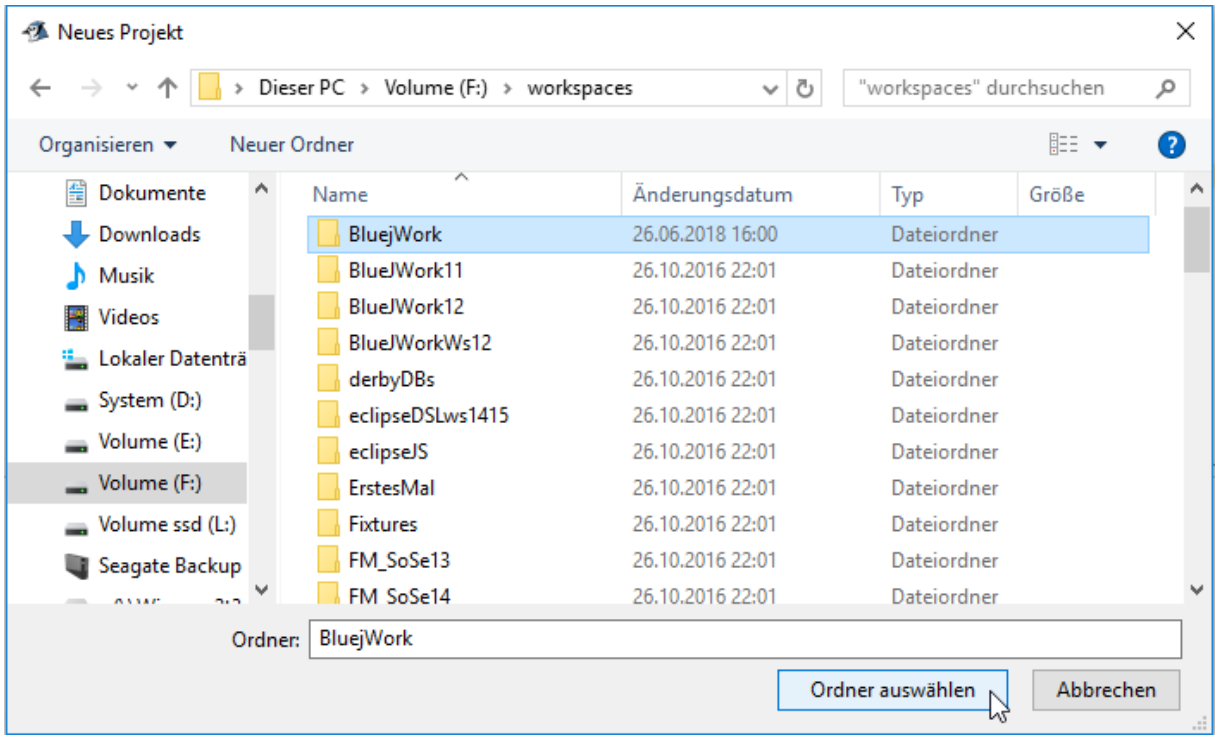

Der Name wird statt „Neuer Ordner“ eingetragen.



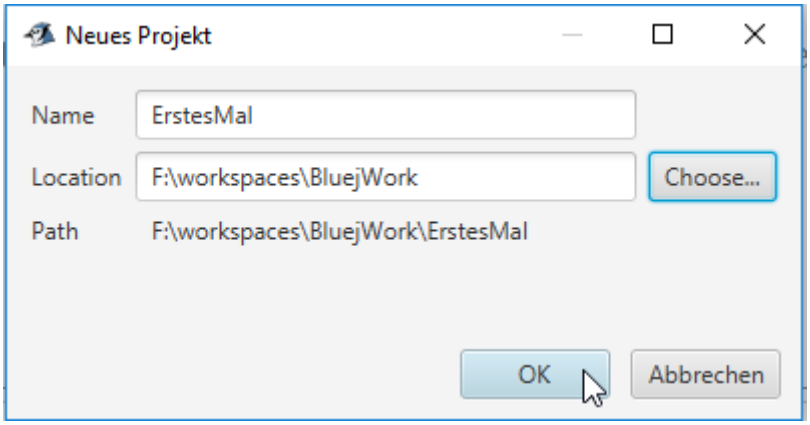

Der Ordner heißt im Beispiel „BlueJWork“.



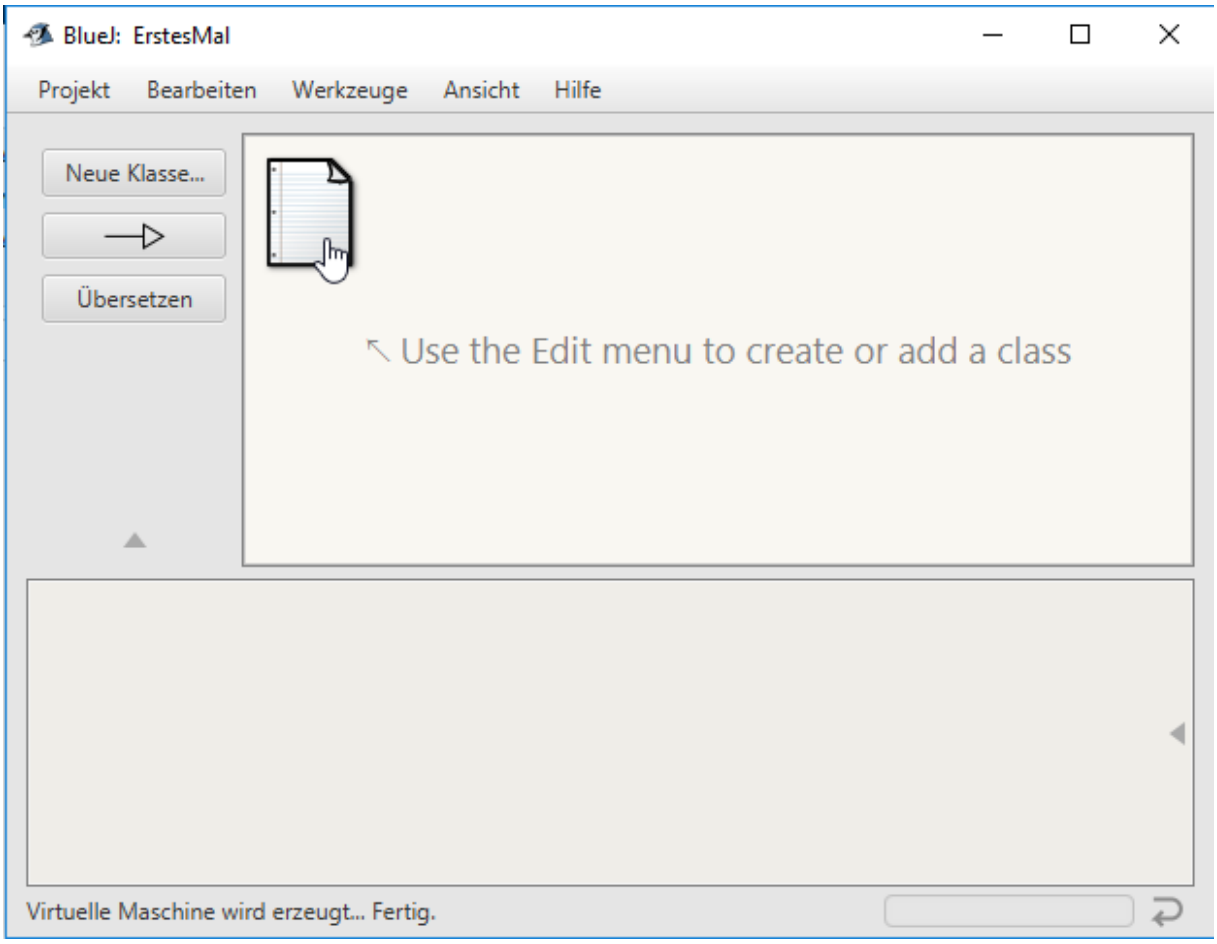

Ist der Ordner dann angelegt oder unter den existierenden Ordnern angeklickt, wird auf „Ordner auswählen“ geklickt.



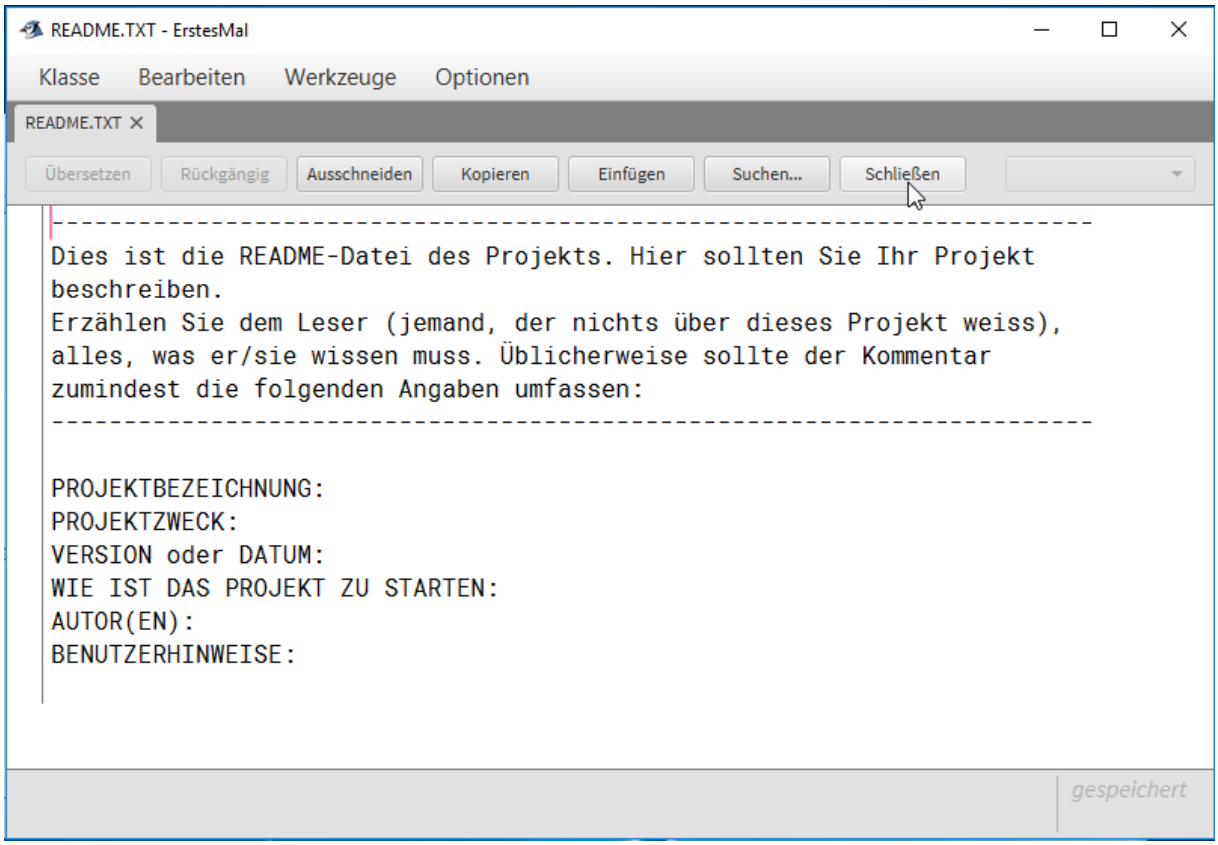

Die Projekterstellung wird mit einem Klick auf „OK“ abgeschlossen.




Das BlueJ-Fenster verändert seine Farbe, die Knöpfe auf der linken Seite werden nutzbar und ein Zettel liegt im Arbeitsbereich, der aus Interesse jetzt einmal mit einem Doppelklick angeklickt wird.

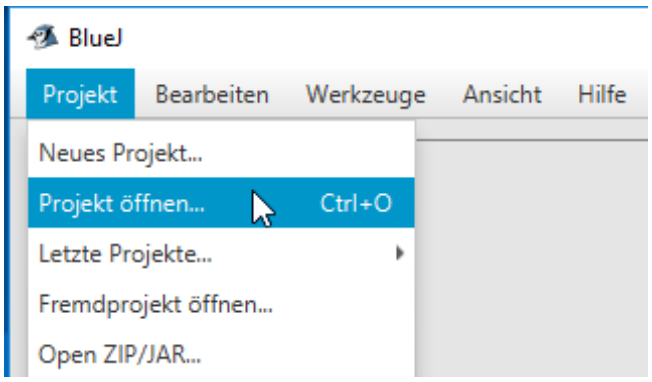



Es öffnet sich ein Editor mit der zum Projekt gehörenden Datei „README.TXT“ in der eine Projektbeschreibung eingetragen werden kann. Der Editor kann über den Knopf „Close“ verlassen werden.

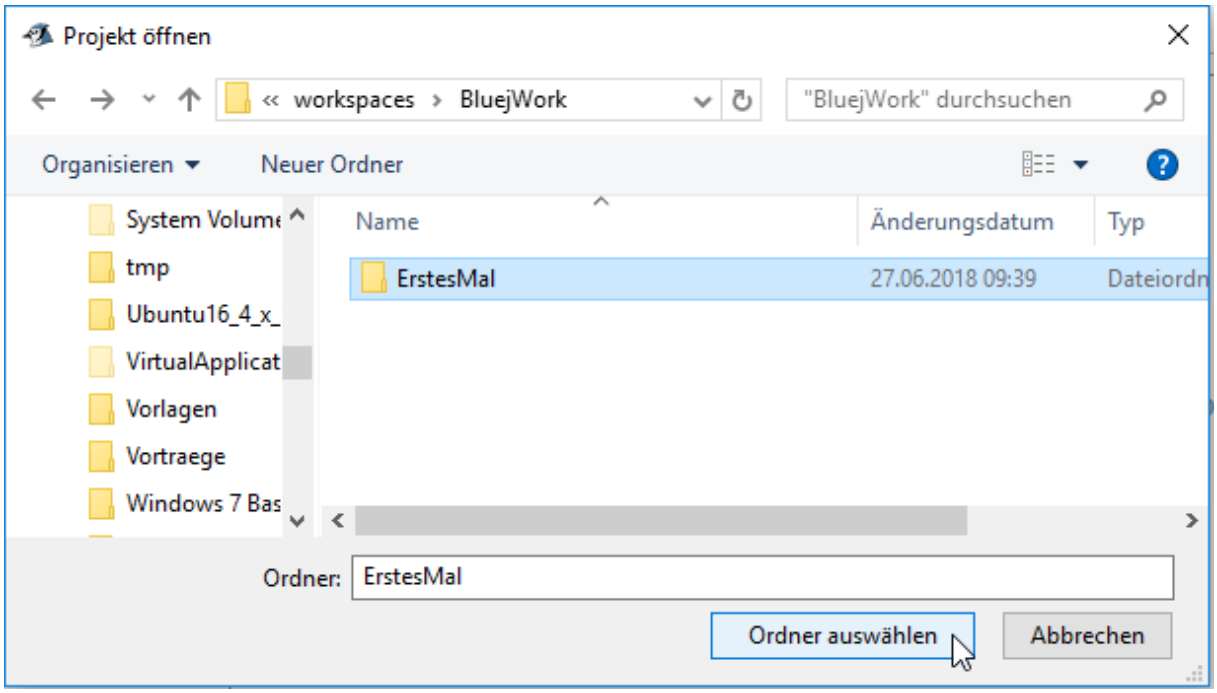



## 4.2 Laden eines existierenden Projekts

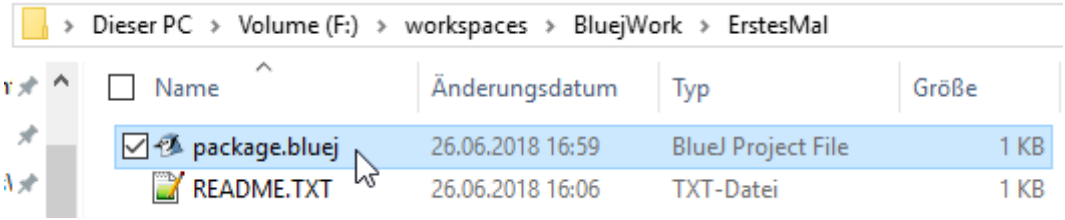
Generell wird beim Start das zuletzt bearbeitete Projekt geladen. Um ein anderes existierendes Projekt zu laden, wird nach dem Start von BlueJ „Projekt -> Projekt öffnen…“ gewählt.



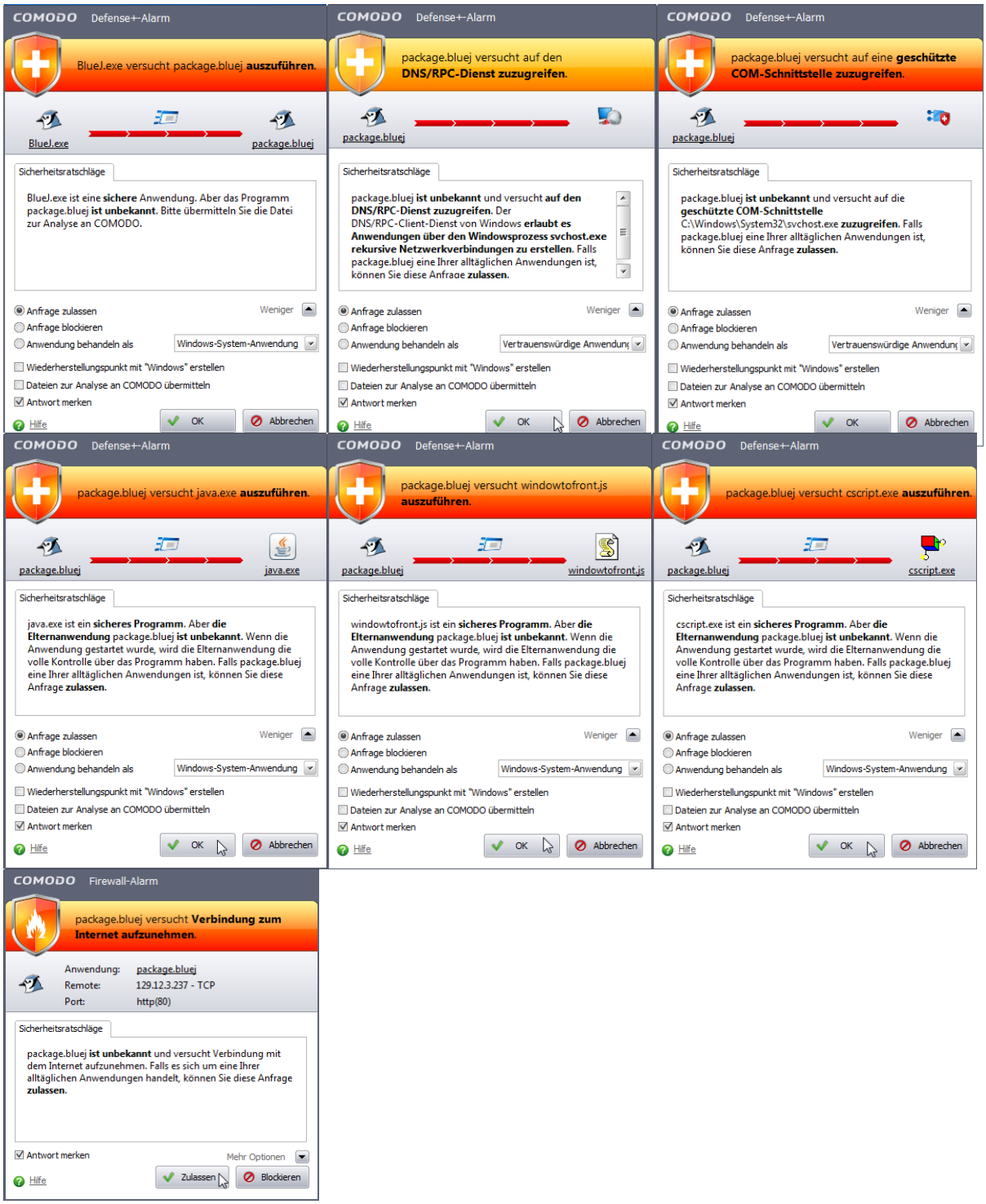

Danach wird zum existierenden Projekt gesteuert und dies mit „Ordner auswählen“ geöffnet.




Ist eine Verknüpfung zwischen der Endung „.bluej“ und dem Programm BlueJ wie bei der Installation beschrieben, erstellt worden, sind Projekte auch mit einem Doppelklick zu öffnen.




Hier können abhängig von der benutzen Firewall zunächst einige Genehmigungen von Zugriffen notwendig sein. Die folgenden Abbildungen zeigen Beispiele, wenn auf einem privaten Rechner die Firewall Comodo genutzt wird.

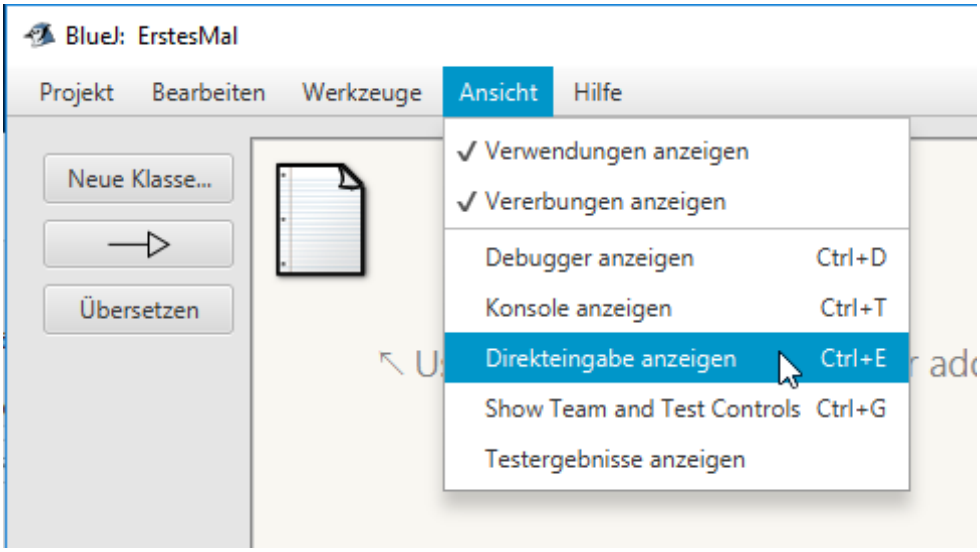



## 4.3 Erste Experimente mit dem Code Pad

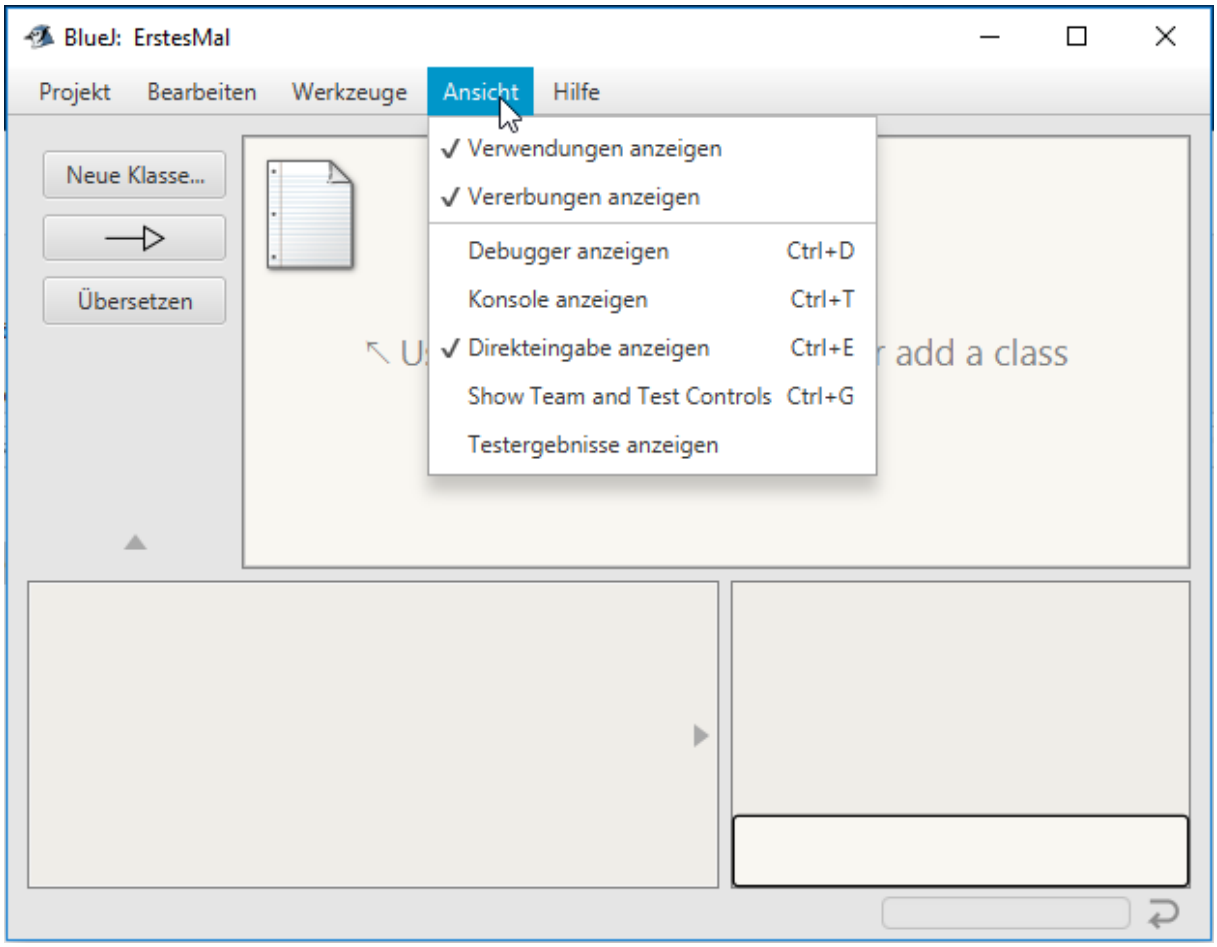
Java ist eine Programmiersprache, die kompiliert wird. Die dann entstehenden Dateien können dann auf den Rechner ausgeführt werden. Genauer entsteht beim Kompilieren sogenannter Byte-Code, der dann auf jedem Rechner ausgeführt werden kann, auf dem eine Java Virtual Machine, genauer eine Java Runtime Edition (JRE) installiert ist.

Damit ist Java eine Programmiersprache, die nicht unmittelbar nach Eingabe der Befehle ausgeführt werden kann, wie es bei Skriptsprachen, wie JavaScript (hat nichts mit Java zu tun) und PHP der Fall ist. Da das sofortige Ausführen von Befehlen mit der Möglichkeit unmittelbar die Ergebnisse zu sehen, gerade für Anfänger eine Lernunterstützung sein kann, hat BlueJ die Möglichkeit geschaffen, einzelne Java-Befehle auszuführen. Dies schafft gerade für Anfänger die Möglichkeit einzelne Programmzeilen auszuführen, bevor die eigentliche Programmierung beginnt. Generell kann diese Möglichkeit auch zum Ausprobieren existierender Klassen und Programme genutzt werden.

Das zugehörige Eingabefenster heißt Code Pad, deutsch Direkteingabe, und muss explizit eingeblendet werden. Hierzu wird unter „Ansicht“ auf „Direkteingabe anzeigen“ geklickt.

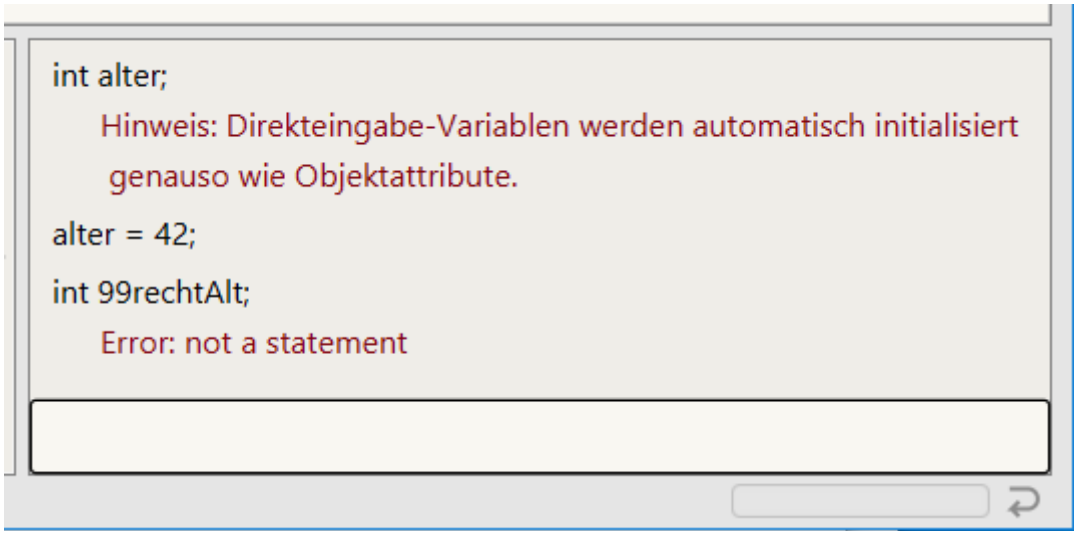



Das Code Pad wird jetzt rechts unten angezeigt. Wird erneut unter „Ansicht“ nachgeschaut, ist einen Haken vor „Direkteingabe anzeigen“ sichtbar.

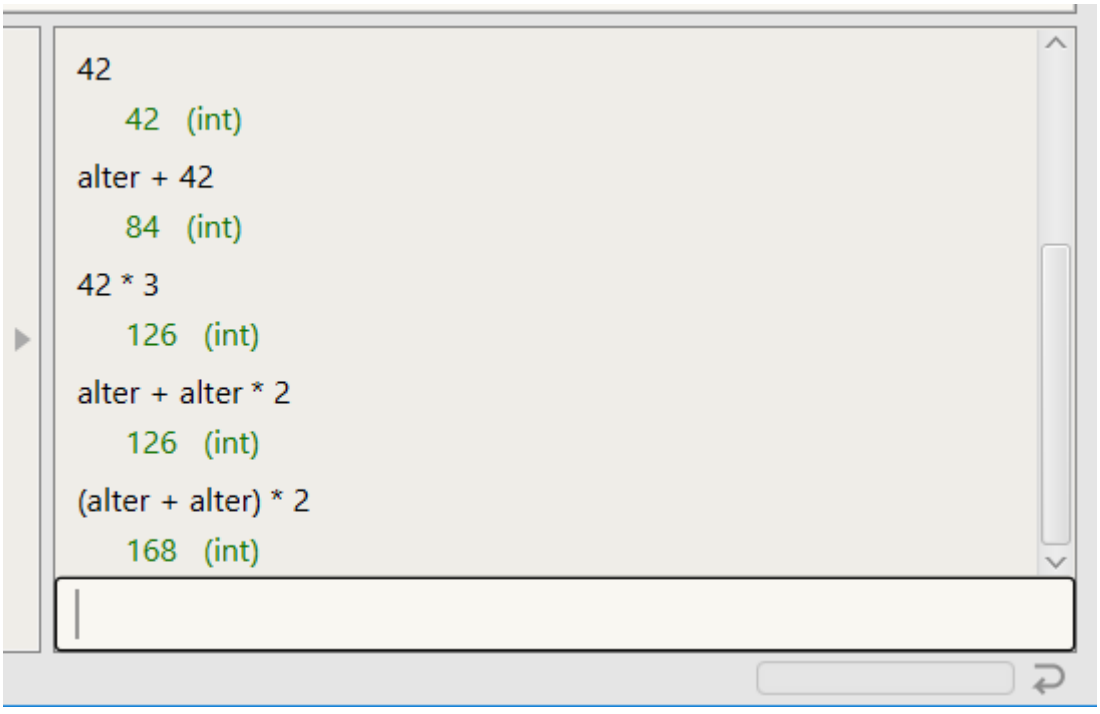



Im Code Pad werden im Wesentlichen Befehle und Ausdrücke eingetippt. Dabei besteht jedes Programm aus einer Folge von Befehlen. Befehle sind daran erkennbar, dass sie mit einem Semikolon beendet werden. Die folgende Abbildung zeigt, wie eine Variable deklariert und eine Zuweisung ausgeführt wird. Zur Ausführung einer Zeile wird einfach „Return“ gedrückt. Der Hinweistext, beginnend mit „Note“ kann einfach ignoriert werden und weist nur daraufhin, dass im Code Pad jede Variable automatisch einen Wert, hier die Zahl Null, erhält.

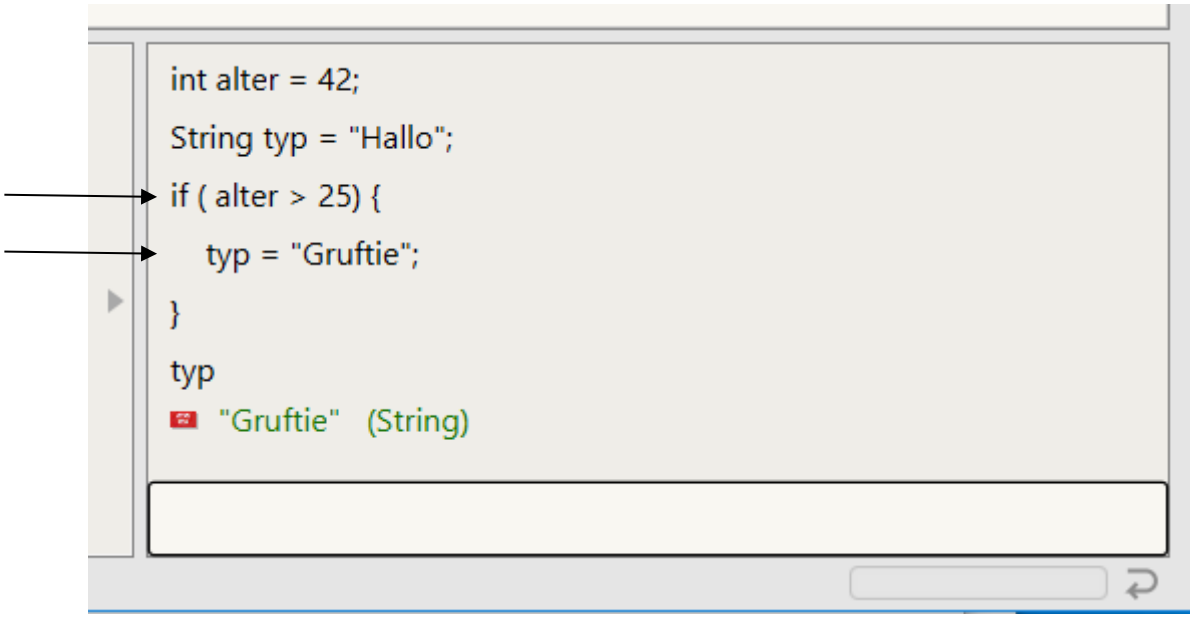
Danach wird versucht, eine zweite Variable zu deklarieren, wobei es hier einen Syntaxfehler gibt. Es ist zu erahnen, dass die Fehlermeldungen oft nicht sehr aussagekräftig sind. Bei einem Fehler stürzt das Code Pad nicht ab, die fehlerhafter Zeile wird ignoriert und es können neue Befehle eingegeben werden. Durch die Pfeil-Tasten „nach oben“ und „nach unten“ kann zu vorher eingegebenen Zeilen zurück bzw. vorwärts manövriert werden. Diese Zeilen sind dann auch editierbar und werden wieder mit „Return“ ausgeführt.



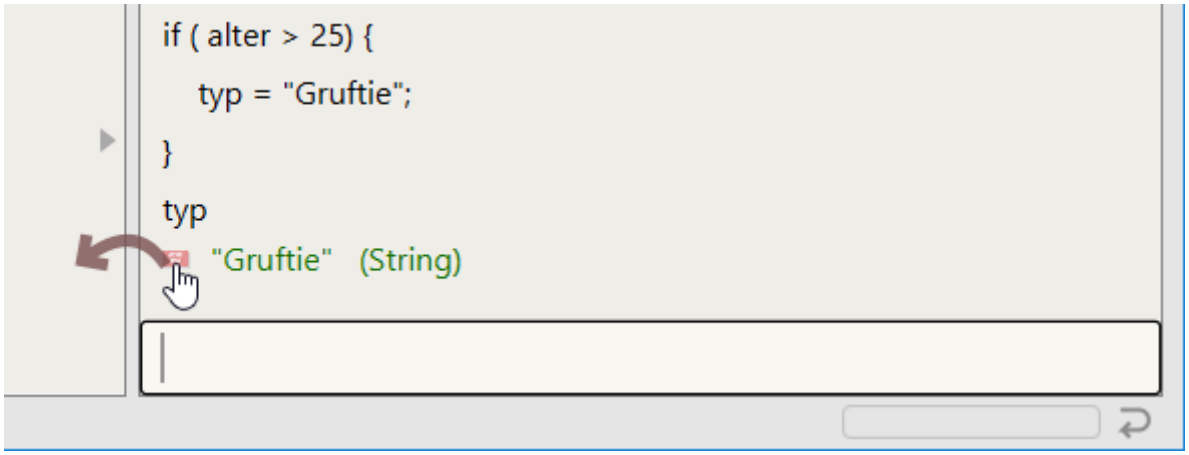

Weiterhin besteht die Möglichkeit in Code Pad Ausdrücke einzugeben, die anschaulich zu einem konkreten Wert ausgewertet werden können. Das Ergebnis oder spätere eine Referenz auf das Ergebnis wird nach dem Drücken der Return-Taste ausgegeben. Ausdrücke enden nicht mit einem Semikolon. Die folgende Abbildung zeigt einige Beispiele.



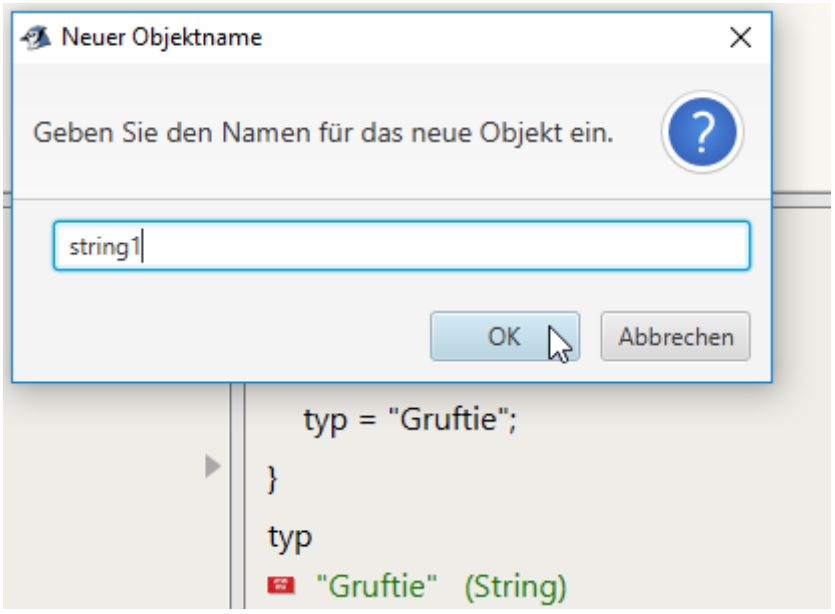

Sollen Befehle eingeben werden, die über mehr als zwei Zeilen gehen, muss am Zeilenende Shift+“Return“-Taste eingeben werden. Das folgende Beispiel zeigt auch, dass Werte für Variablen vom Typ String in Anführungsstrichen eingegeben werden. Bei den mit einem Pfeil markierten Zeilen wurde der Zeilenumbruch mit Shift+Return durchgeführt. Wird dies vergessen, erscheint eine Fehlermeldung in der Ausgabe und der gesamte Befehl muss ernerut eingegeben werden. Die letzte Zeile „typ“ wiederholt die Möglichkeit den Wert einer Variablen sich ausgeben zu lassen.



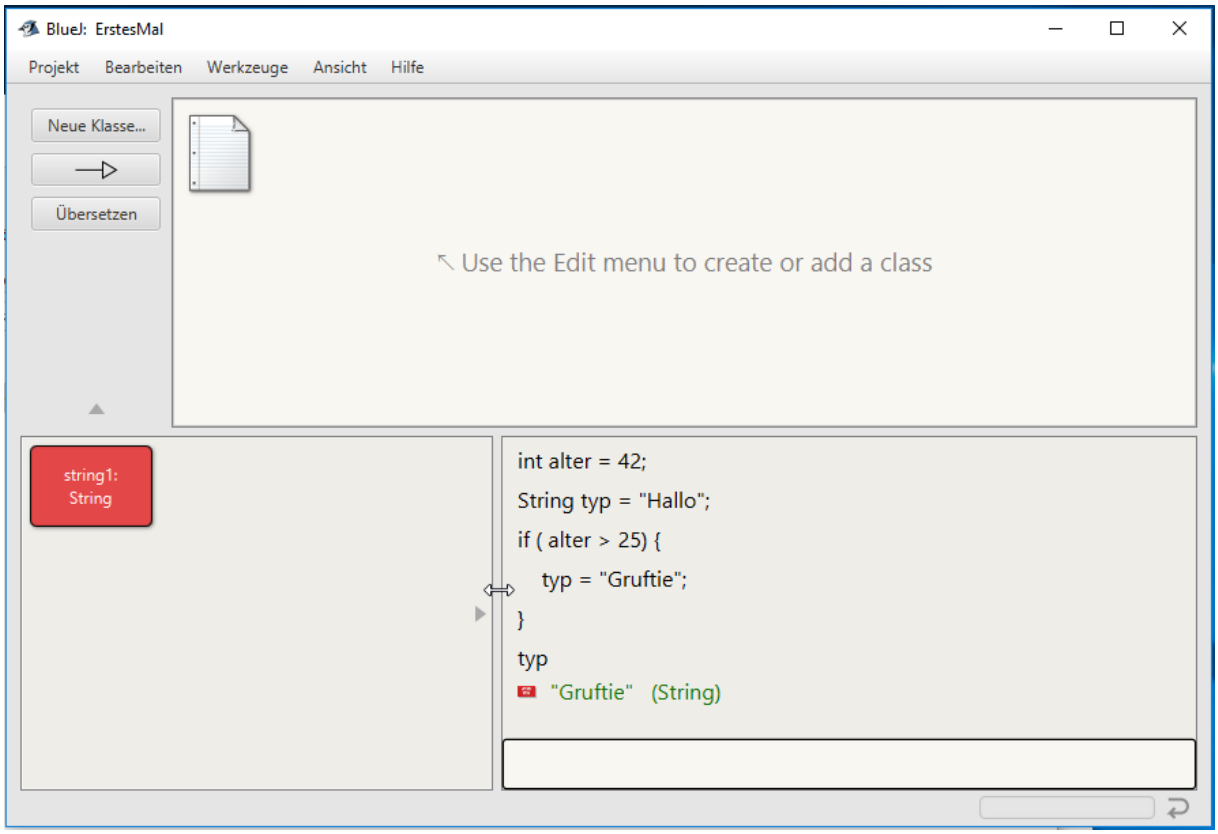

Neben der Ausgabe des Textes ist ein kleiner roter Kasten zu sehen, der andeutet, dass es sich bei dem Ergebnis des letzten Ausdrucks (typ) um ein Objekt, genauer eine Objektreferenz handelt. Wird die Maus auf den roten Kasten bewegt, erscheint ein Pfeil, der nach links auf die Objektleiste zeigt.



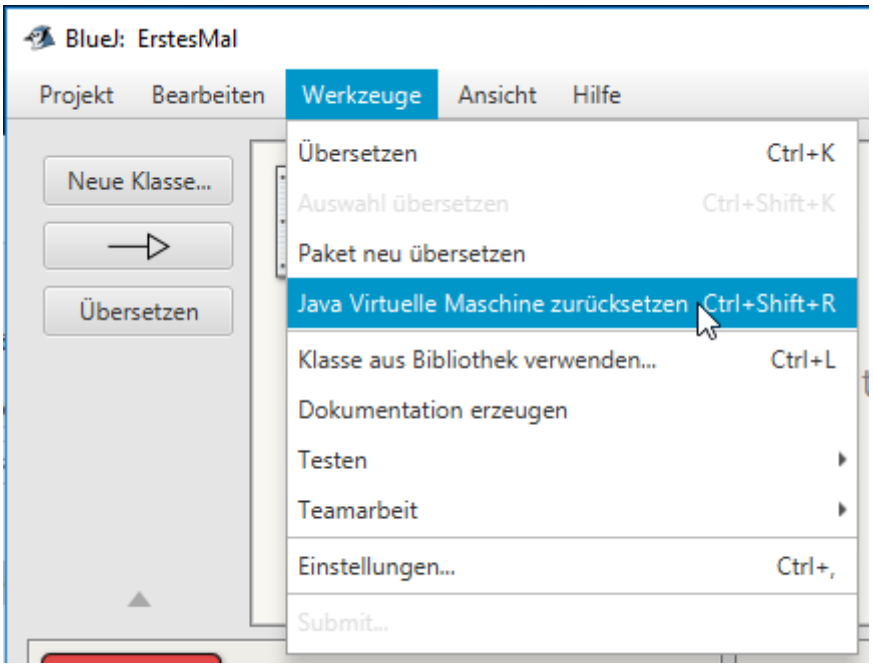

Wird dann auf den Kasten geklickt, öffnet sich das folgende Dialogfenster und dem Objekt muss ein neuer Name gegeben werden. Der Name muss den Regeln für Variablennamen entsprechen, mit einem Buchstaben beginnen und keine Leerzeichen enthalten. Mit einem Klick auf „OK“ wird das Objekt in die Objektleiste übernommen.



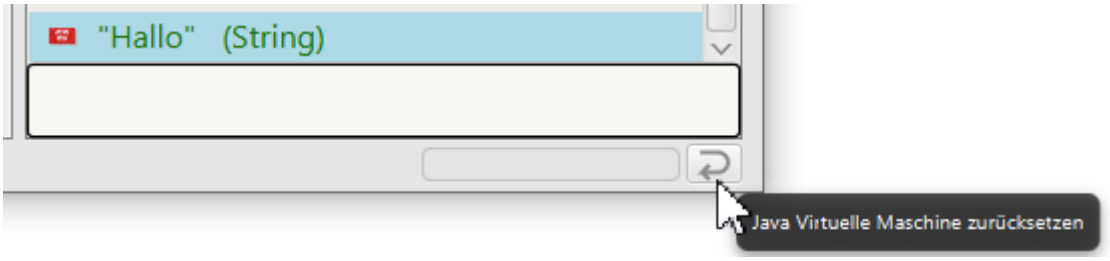

Das Objekt befindet sich dann links in der Objektleiste. Die genauere Nutzung dieser Leiste wird in „4.7 Ausführung von Exemplarmethoden“ und nachfolgenden Kapiteln beschrieben.



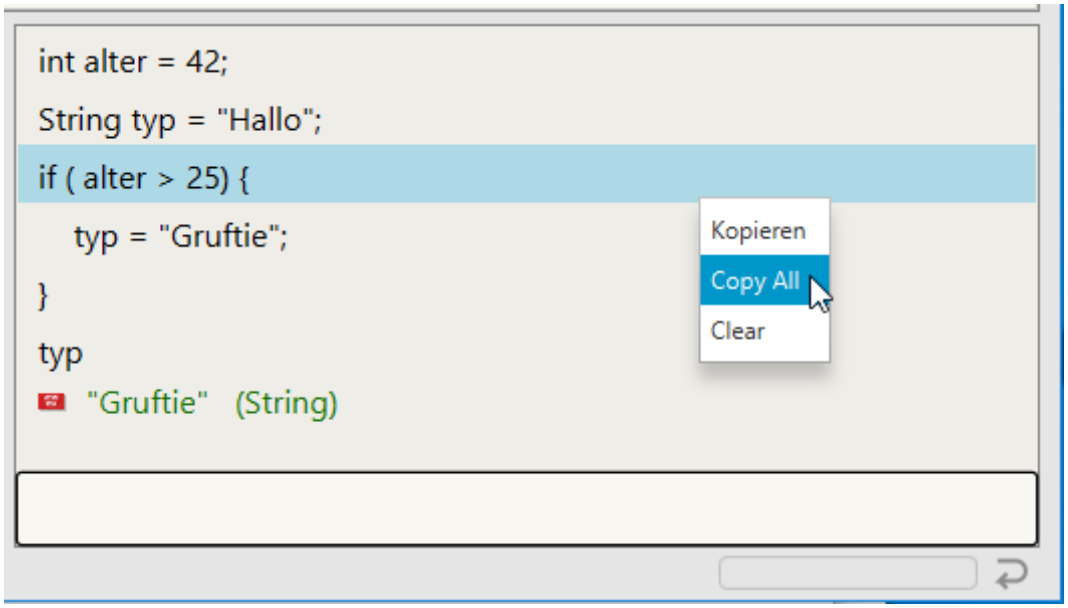

Sollen alle eingegebenen Variablen gelöscht werden, besteht eine Möglichkeit darin, die zur Ausführung benutzte Java Virtual Maschine zu resetten. Dies ist durch die Auswahl „Werkzeuge“ und „Java Virtuelle Maschine zurücksetzen“ möglich.



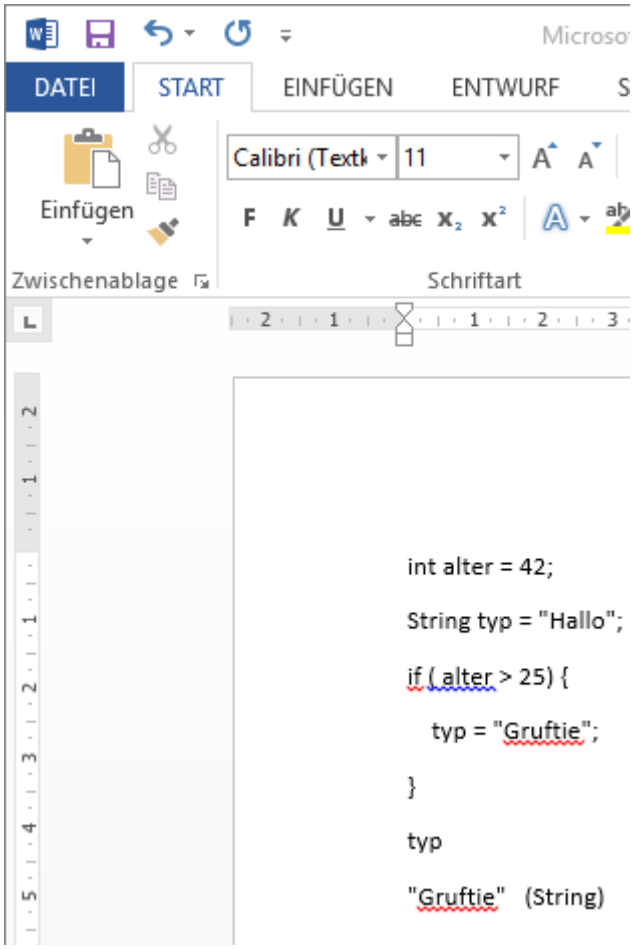

Alternativ ist auch der gebogene Pfeil rechts-unten nutzbar.



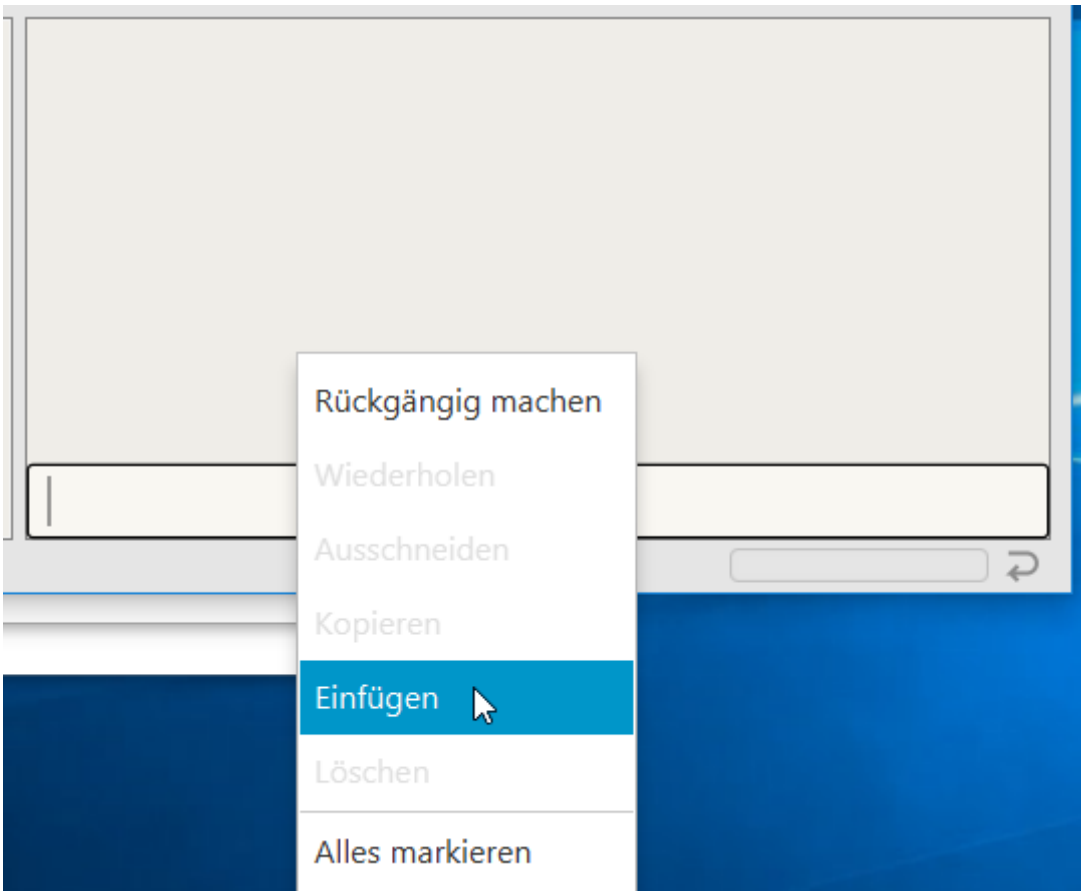

Der genaue Zustand des Code Pads kann nicht abgespeichert werden, es besteht aber die Möglichkeit die Eingaben mit einem Rechtsklick und der Auswahl „Kopieren“, die die aktuelle Zeile kopiert, oder mit „Copy All“ den gesamten Inhalt zu kopieren und mit der Tastenkombination „Strg“+“C“ z. B. in eine Word-Datei einzufügen.




Die folgende Abbildung zeigt, dass Ausgaben mit kopiert werden. Alternativ kann von den Ergebnissen ein Bildschirmfoto gemacht werden, wie es in Kapitel „9 Installation des Screenshot-Werkzeugs Faststone Capture“ beschrieben wird.



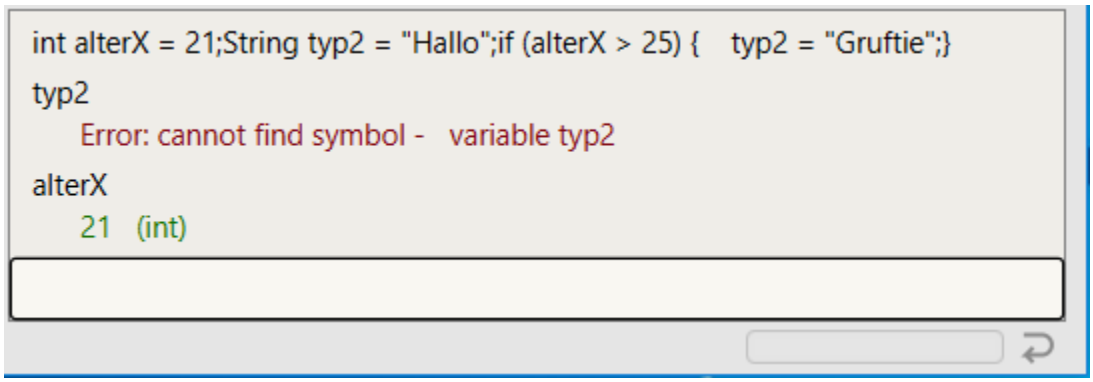
Eingaben sind auch ins Code Pad zurück kopierbar, wobei dann natürlich die Ausgaben nicht mitkopiert werden sollten. Die mit „Strg“+“V“ ins Code Pad kopierten Texte werden insgesamt als ein Befehl angesehen, sollte dieser einen Fehler enthalten, wird er nicht ausgeführt. Statt der der Tastenkombination kann auch über einen Rechtsklick in der Eingabezeile „Einfügen“ genutzt werden.



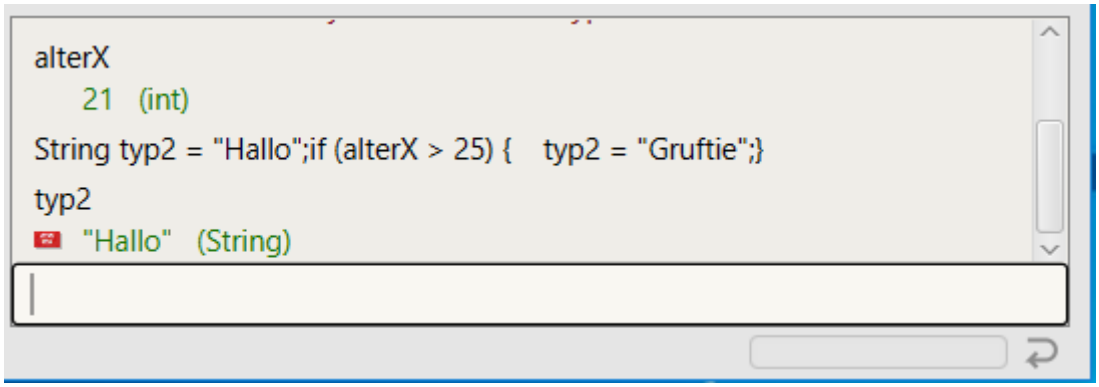

In der folgenden Abbildung wurde der folgende Text

```
int alterX = 21;

String typ2 = "Hallo";

if (alterX > 25) {

    typ2 = "Gruftie";

}
```

nach Code Pad kopiert und danach der angegebene Ausdruck ausgeführt. Die folgende Abbildung zeigt, dass diese Möglichkeite in BlueJ 4.2.1 leider nicht funktioniert, da u. a. Zeilenumbrüche nicht ordentlich kopiert werden.

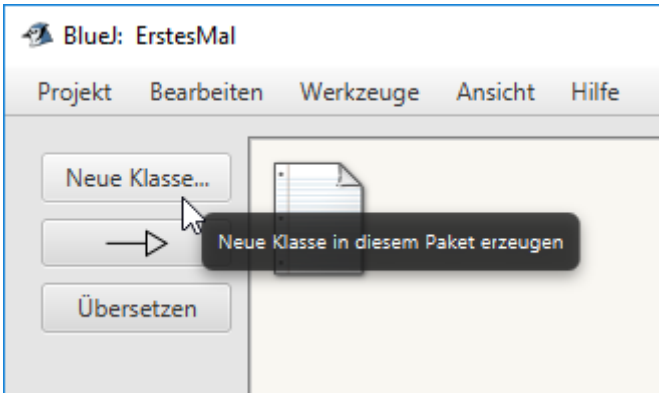



Wird die Zeile mit der kopierten Eingabe zerlegt, zeigt die folgende Abbildung, dass der Ansatz mit dem Zurückopieren generell funktioniert, es also anscheinend Probleme gibt, wenn mehr als eine Variable deklariert wird. Generell dürfen in Java mehrere mit einem Semikolon getrennte Befehle in einer Zeile hintereinander stehen, was aber schlechter Programmierstil ist.



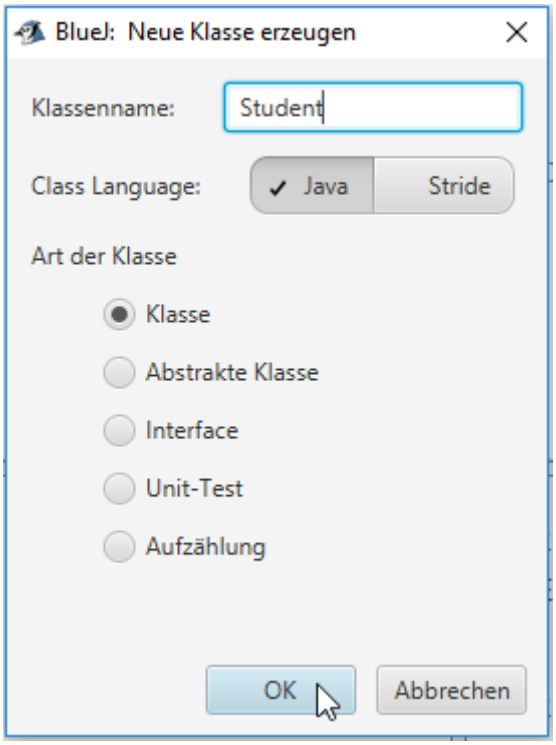

## 4.4 Erstellung einer ersten Klasse

Nachdem ein Projekt angelegt ist, kann die eigentliche Entwicklung starten. Hierzu wird zunächst „Neue Klasse…“ geklickt, um eine erste Klasse zu erstellen.

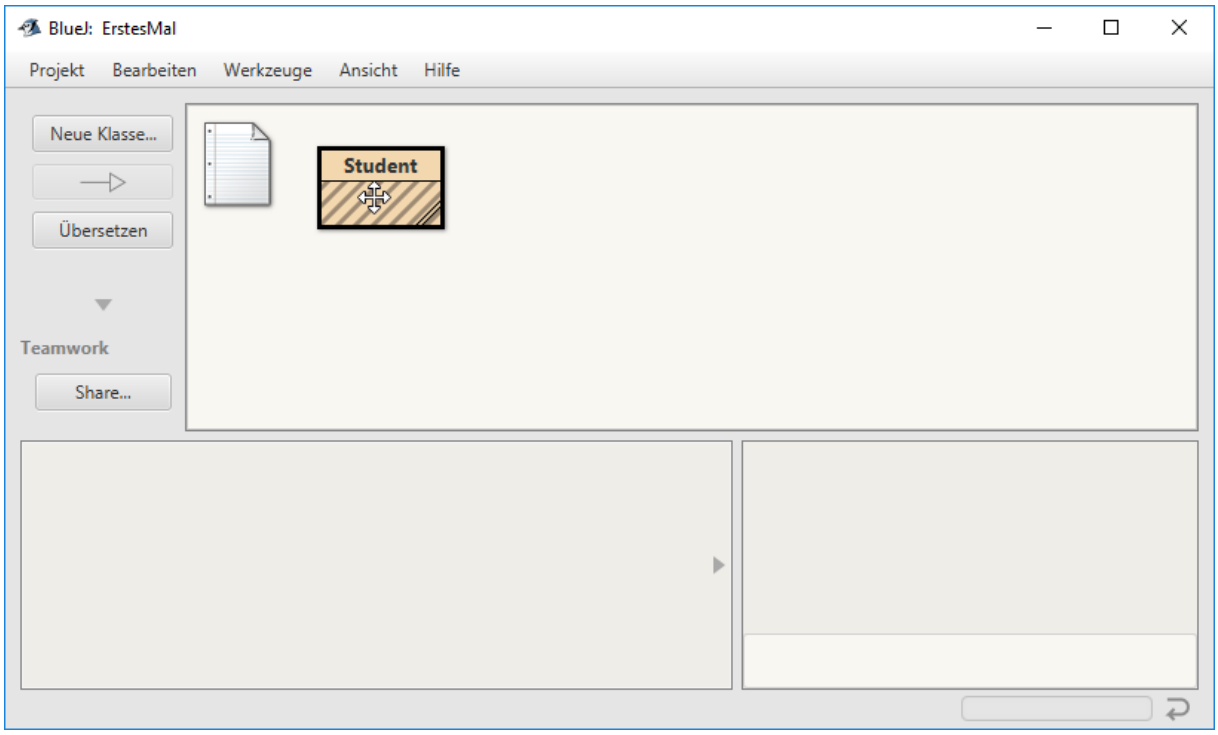



Die neue Klasse muss einen Klassennamen haben, der im Feld „Klassenname:“ eingetragen wird. Der Name soll mit einem Großbuchstaben beginnen und nur kleine und große Buchstaben, als Ausnahme auch Ziffern, niemals aber Umlaute oder Sonderzeichen, beinhalten.

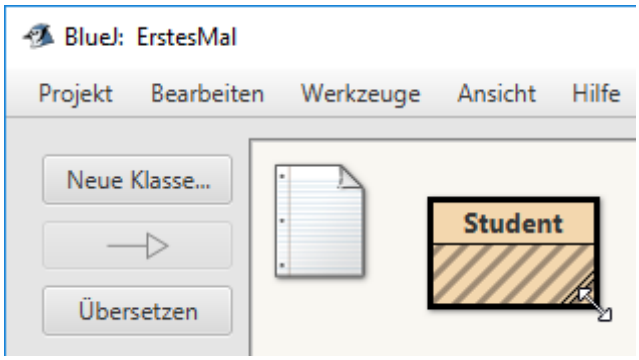
Hier wird eine Klasse „Student“ angelegt. Weiterhin ist die Art der Klasse wählbar, was allerdings später einfach im Quellcode geändert werden kann. Hier wird die Standardeinstellung genutzt. Die Eingabe wird mit einem Klick auf „Ok“ abgeschlossen.



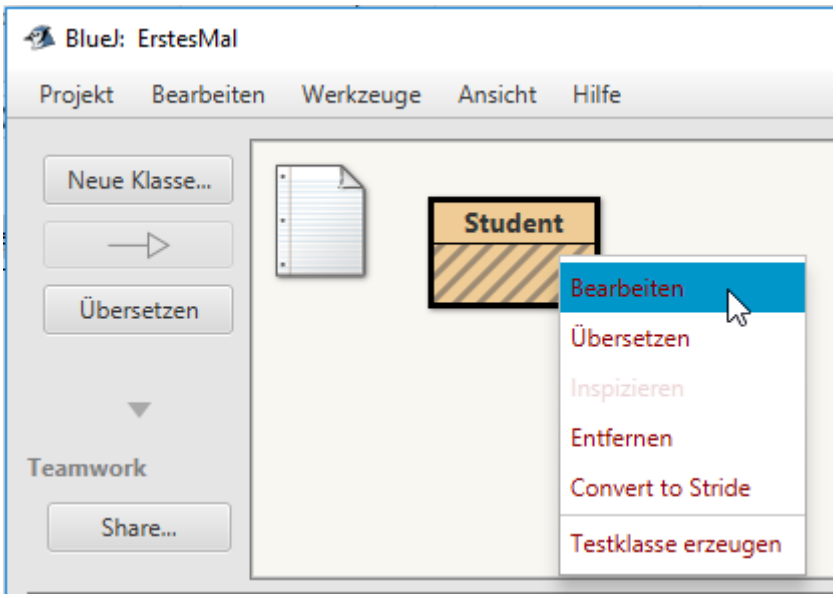

Im Arbeitsbereich wird jetzt die Klasse angezeigt, die mit gedrückter linker Maustaste beliebig im Arbeitsbereich platziert werden kann.



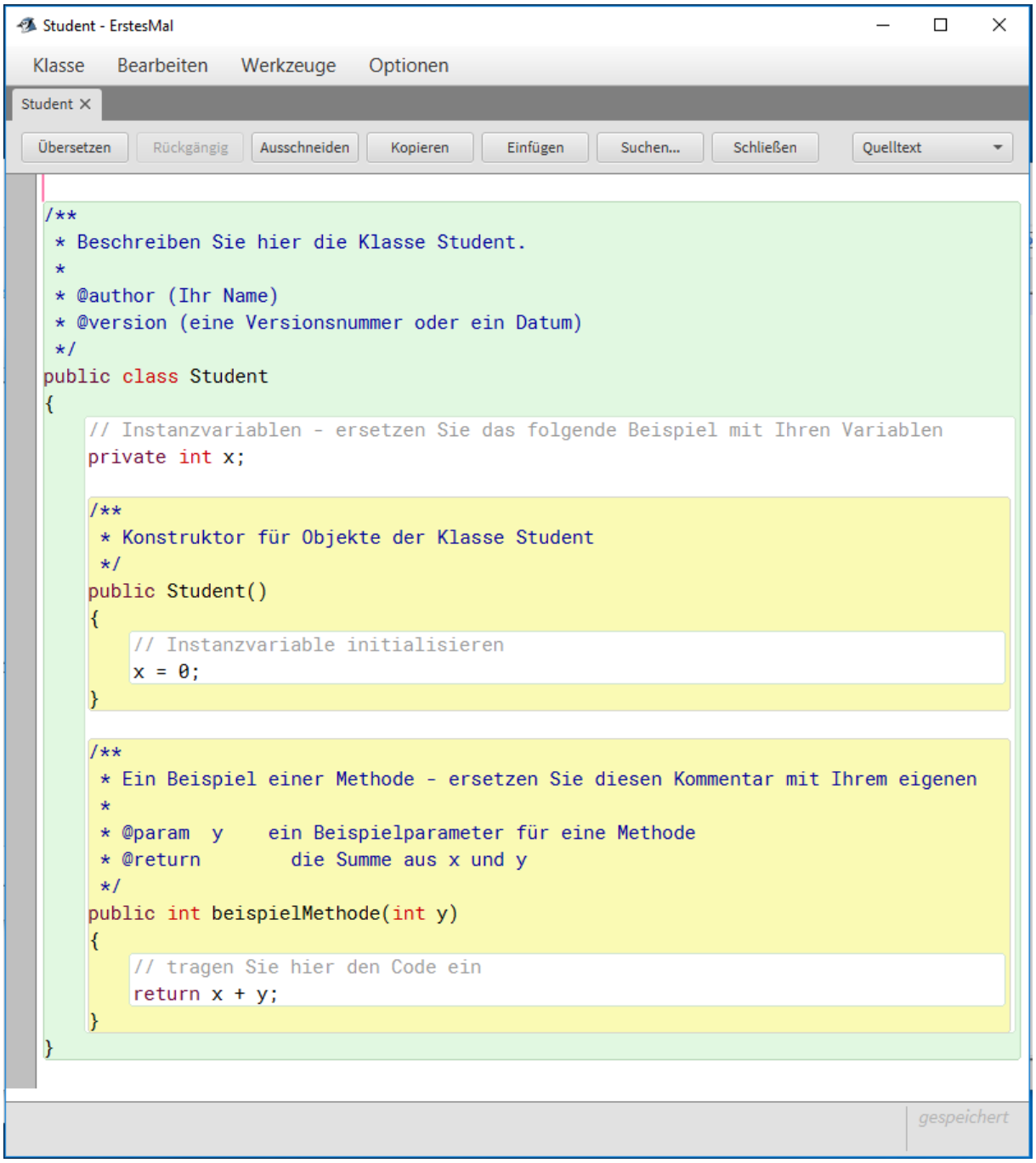
Wird die Maus über die rechte untere Ecke der Klasse bewegt, ist die Möglichkeit erkennbar, mit gedrückter linker Maustaste die Größe der Klasse zu ändern.



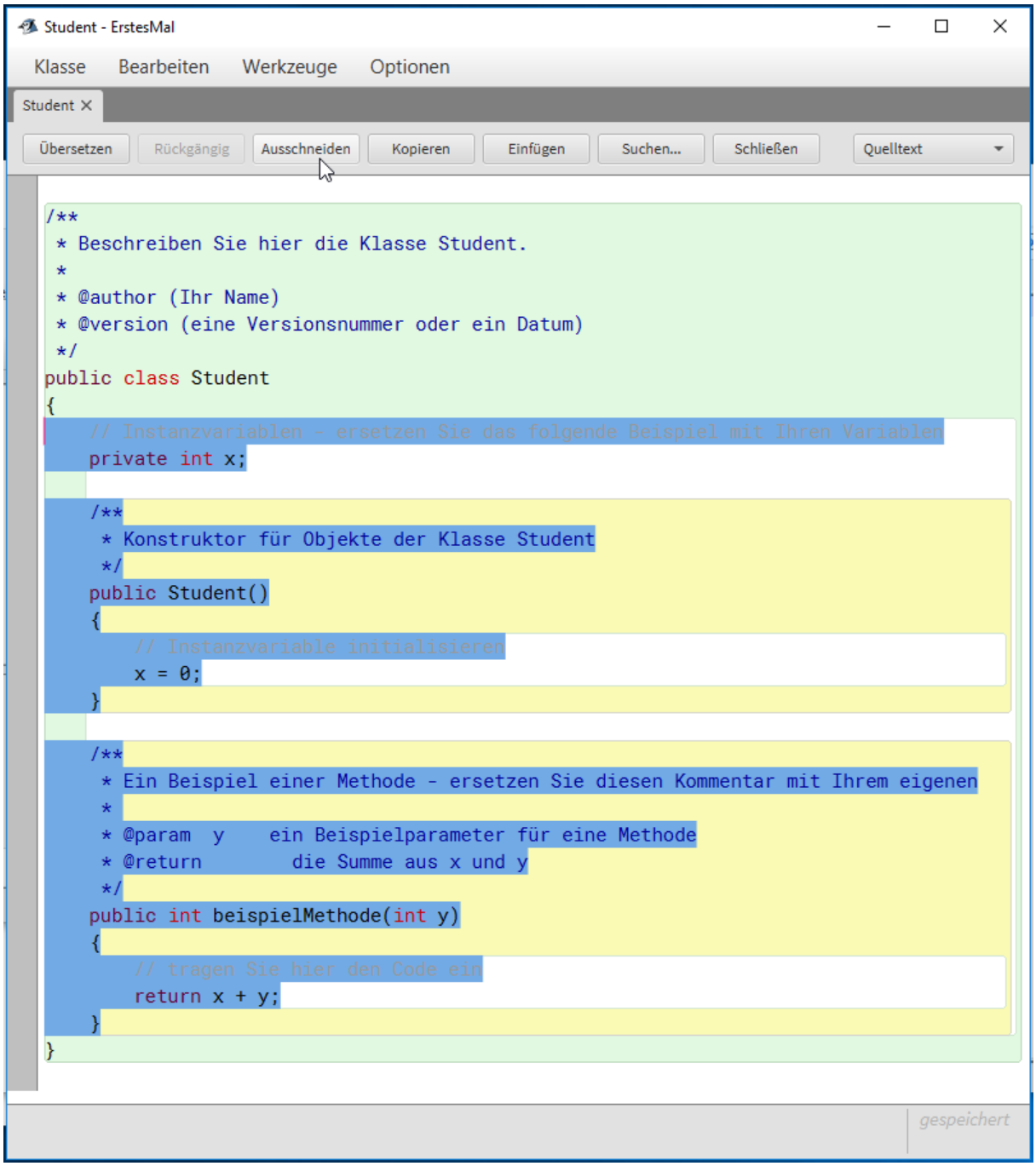

Um die Klasse mit Leben zu füllen, wird ein Rechtsklick auf der Klasse gemacht und „Bearbeiten“ ausgewählt, alternativ kann ein Doppelklick auf der Klasse ausgeführt werden.




Es öffnet sich ein Editor, der schon bemerkenswert viel vermeintlichen Programmcode enthält. Generell ist anzumerken, dass der Editor einige wesentliche Funktionen einer Code-Entwicklungsumgebung hat, kommerziellere Werkzeuge aber deutlich mehr Möglichkeiten bieten. Da diese Entwicklungsumgebungen, wie NetBeans und Eclipse, aber für Anfänger eine recht unübersichtliche Vielfalt an Nutzungsmöglichkeiten bieten und BlueJ als einzige Umgebung detaillierte Einblicke in die Objekterstellung und Objektnutzung bietet, ist BlueJ die bessere Alternative für die erste Auseinandersetzung mit der objektorientierten Software-Entwicklung.




Der enthaltene Text in der Abbildung zeigt die typische Struktur einer Klasse. Der Text ist schrittweise durch die eigene Implementierung ersetzbar, vereinfachend werden hier oder alle Informationen innerhalb der geschweiften Klammern der Klasse gelöscht. Das Markieren erfolgt wie üblich mit gedrückter linker Maustaste oder mit der Tastenkombination „Strg“+“A“. Das Entfernen ist z. B. über den „Ausschneiden“-Knopf möglich. Der genaue Inhalt, der beim Erstellen einer neuen Klasse angezeigt wird, ist konfigurierbar. In der KleukersSEU werden weniger der am Anfang potenziell verwirrenden Informationen für eine Klasse generiert.

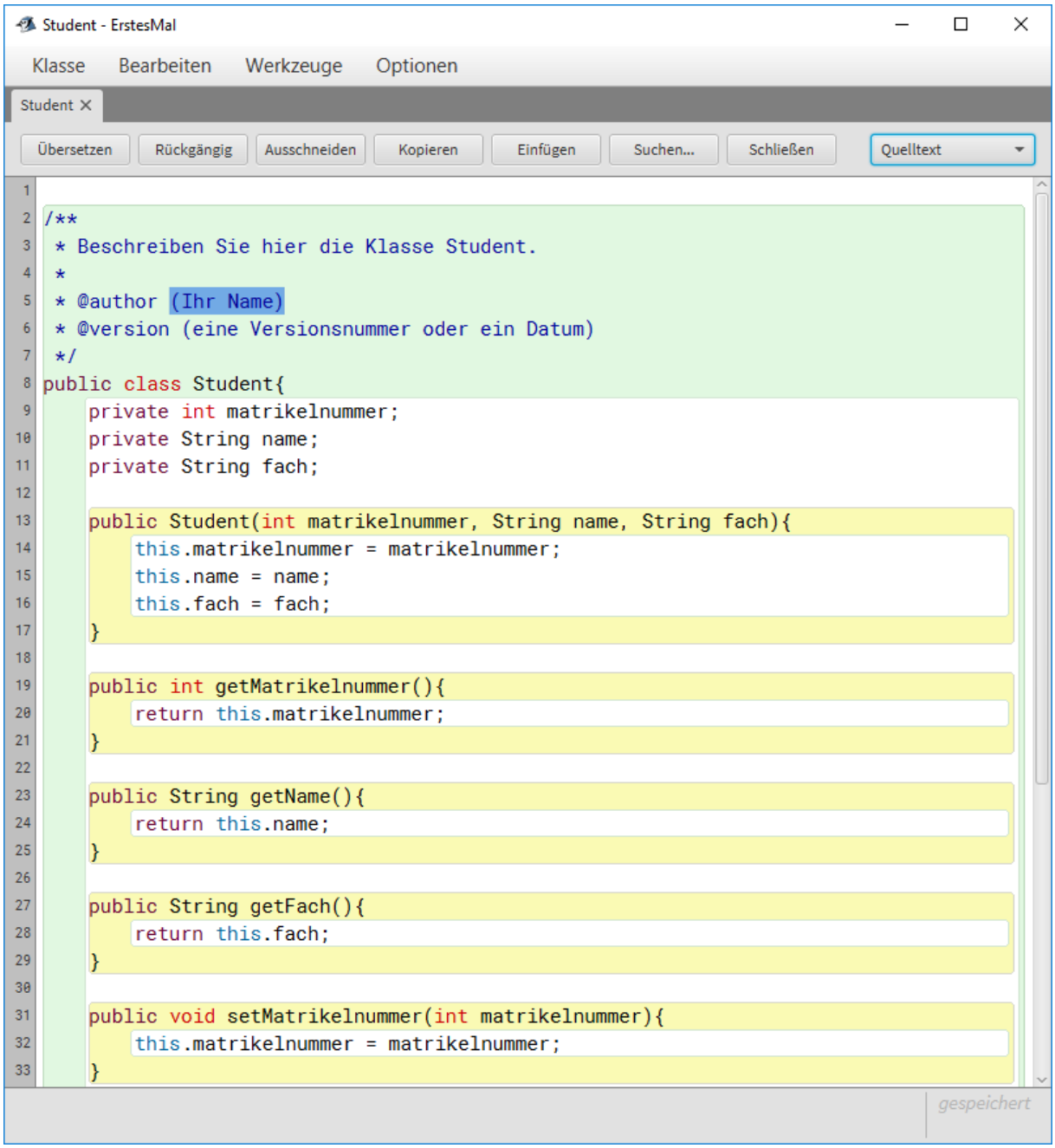



Nun kann die eigentliche Programmierung beginnen, der obere Kommentar kann stehengelassen und später ergänzt werden. Nach etwas Tipperei kann die Klasse wie folgt aussehen, dabei sollte die Tabulator-Taste zum Einrücken genutzt werden. Soll eine Tabulatorposition rückwärts gesprungen werden, wird „Shift“+“Tabulator“-Taste gedrückt. Weiterhin sollte für jede öffnende Klammer zunächst die schließende Klammer eingeben werden, um dann den Bereich zwischen den Klammern füllen.

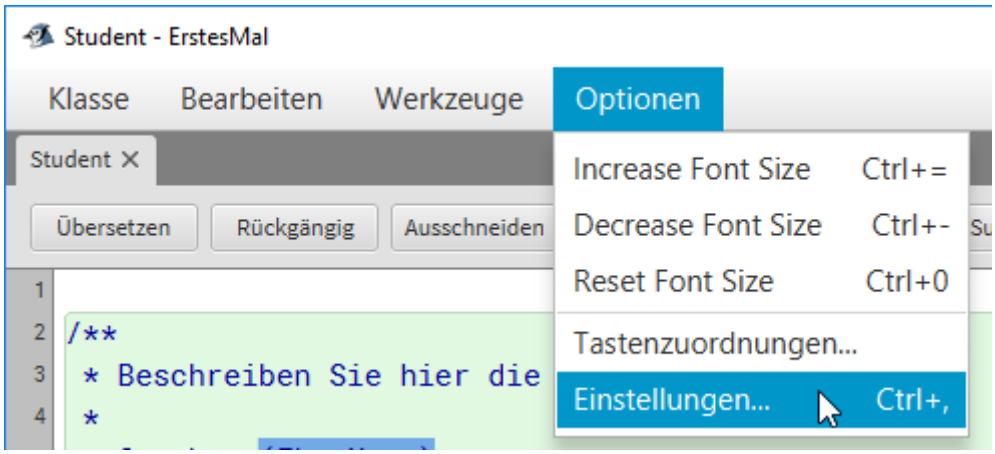
Es ist zu bemerken, dass hier eine andere Variante der Zuordnung der Position der öffnenden zur schließenden Klammer stattfindet, als im vorgegebenen Beispiel aus der letzten Abbildung. Hier ist es nur wichtig, projektweit die einheitliche Formatierung zu nutzen. Um die korrekte

Syntax zu überprüfen, wird der „Übersetzen“-Knopf gedrückt. Das sollte immer nach der Eingabe einer Exemplarvariable oder Methode erfolgen, um frühzeitig Fehler zu erkennen. Dies ist der zentrale Gedanke, bei der Entwicklung inkrementell vorzugehen und sich schrittweise von einer funktionierenden Teillösung zur nächsten funktionierenden Teillösung weiterzuhangeln.

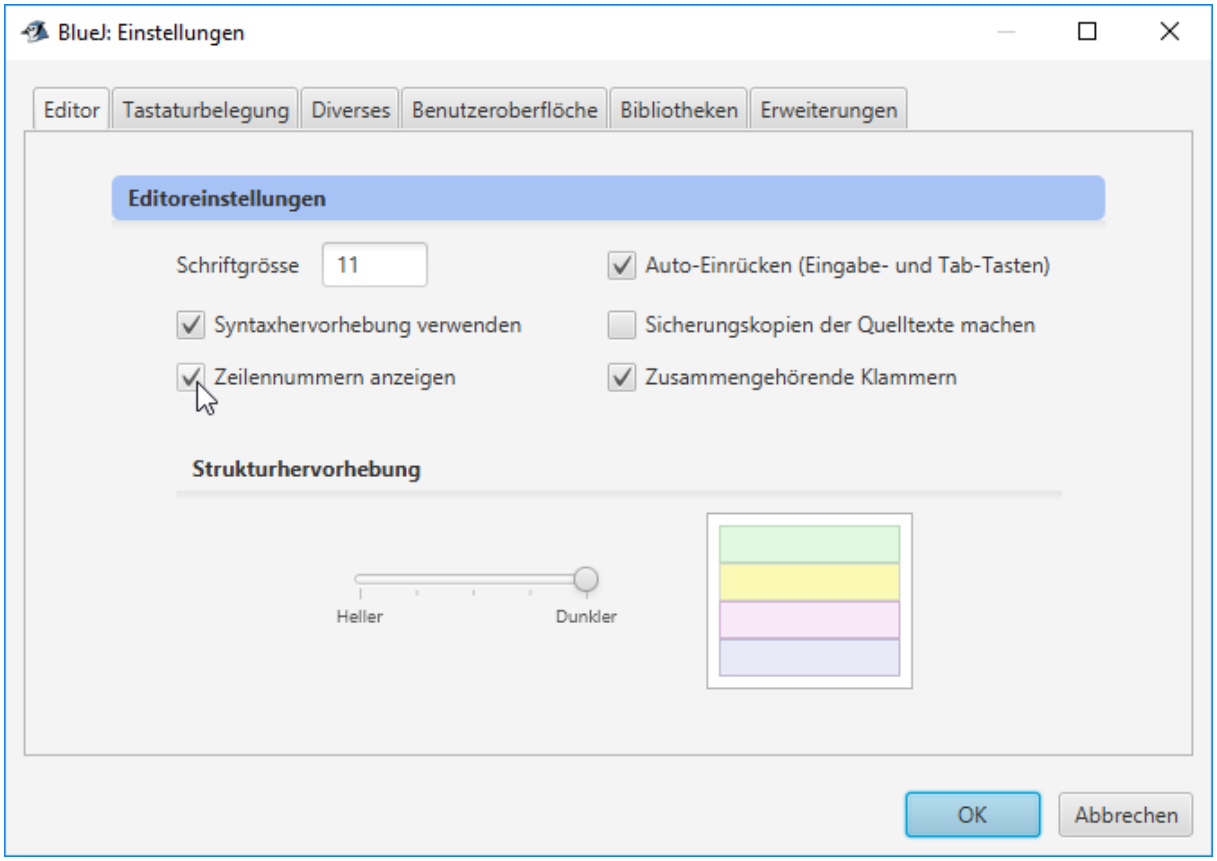
```
public class Student{
    private int matrikelnummer;
    private String name;
    private String fach;

    public Student(int matrikelnummer, String name, String fach){
        this.matrikelnummer = matrikelnummer;
        this.name = name;
        this.fach = fach;
    }

    public int getMatrikelnummer(){
        return this.matrikelnummer;
    }

    public String getName(){
        return this.name;
    }

    public String getFach(){
        return this.fach;
    }

    public void setMatrikelnummer(int matrikelnummer){
        this.matrikelnummer = matrikelnummer;
    }

    public void setName(String name){
        this.name = name;
    }

    public void setFach(String fach){
        this.fach = fach;
    }

    @Override
    public String toString(){
        return this.name + " ("+this.matrikelnummer+" ): "
               + this.fach;
    }

}
```




Die vorherige Abbildung zeigt am linken Rand Zeilennummern, die zunächst nicht eingeblendet sind. Da diese u. a. die Beschreibung von Fehlerorten erleichtern, wird hier kurz eingeschoben, wie sie eingeschaltet werden können. Hierzu wird auf „Optionen“ und dann „Einstellungen…“ geklickt.




Unter dem anzuklickenden Reiter „Editor“, wird der Haken bei „Zeilennummern anzeigen“ auf der linken Seite geklickt und „OK“ geklickt.

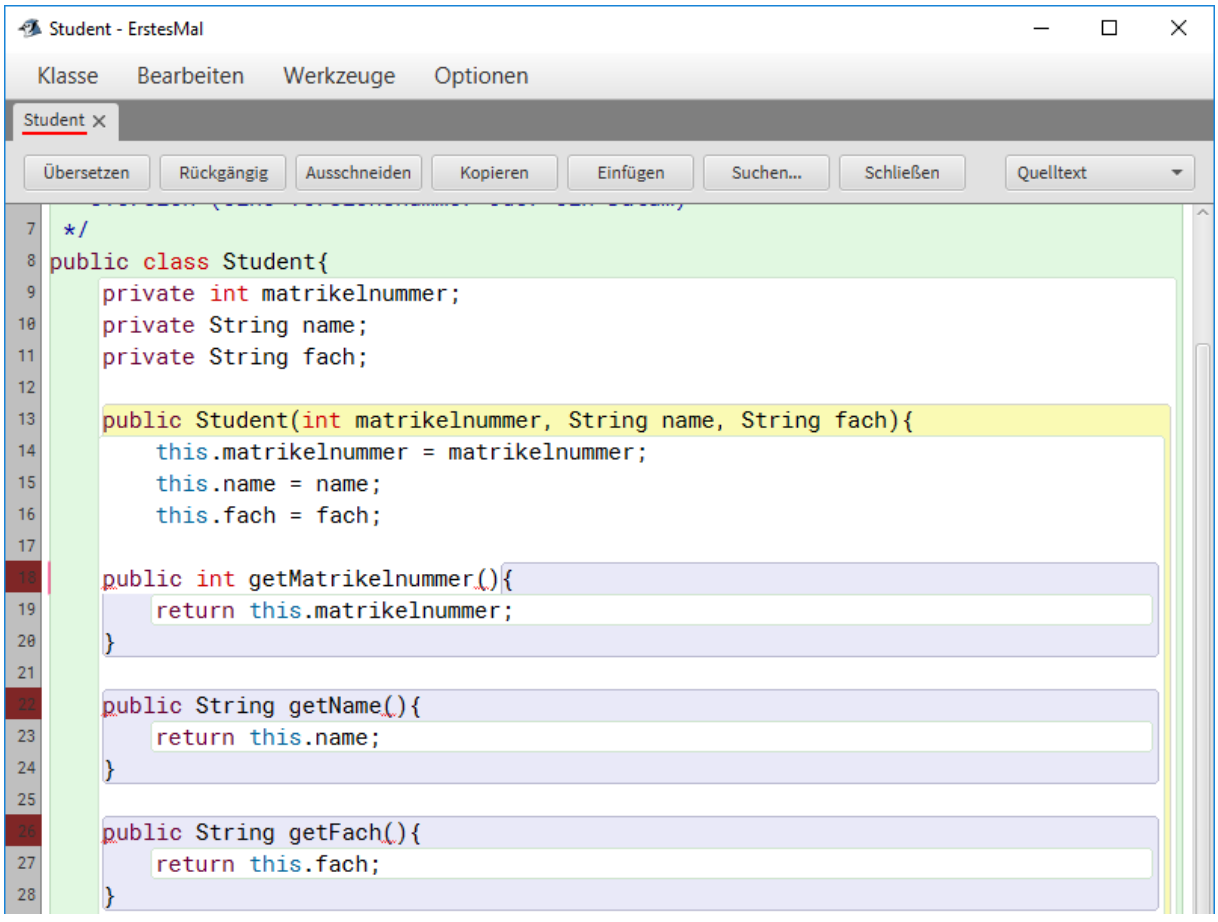



Im Fehlerfall bei der Eingabe von syntaktisch falschem Programmcode (hier Methode setFach), wird die problematische Zeile links markiert und beim Linksklick auf die Methode ein Hinweis auf den Fehler ausgegeben. Genauer auf den ersten gefundenen Fehler, weitere Fehler werden nicht angezeigt.

```
35      public void setName(String name){
36          this.name = name;
37      }
38
39      public setFach(String fach){
40          thi
41      }
42
43      @Override
44      public String toString(){
45          return this.name + " ("+this.matrikelnummer+" ): " + this.fach;
46      }
47
48  }
```

invalid method declaration; return type required

Es ist zu beachten, dass bei der Fehlererkennung zwar das Finden nicht schwierig ist, die Gestaltung einer sinnvollen Fehlermeldung aber oft schwer bis unmöglich ist. Erfahrene Entwickler wissen deshalb, dass sie nicht unbedingt der angezeigten Fehlermeldung trauen können und den Fehler auch im umgebenden Programmcode der markierten Stelle suchen müssen. Besonders kritisch sind hier vergessene Klammern, da dann irgendein nachfolgender Programmteil als Fehler markiert wird. Im folgenden Beispiel wurde die schließende Klammer des Konstruktors vergessen und die Folgezeile als kritisch markiert.

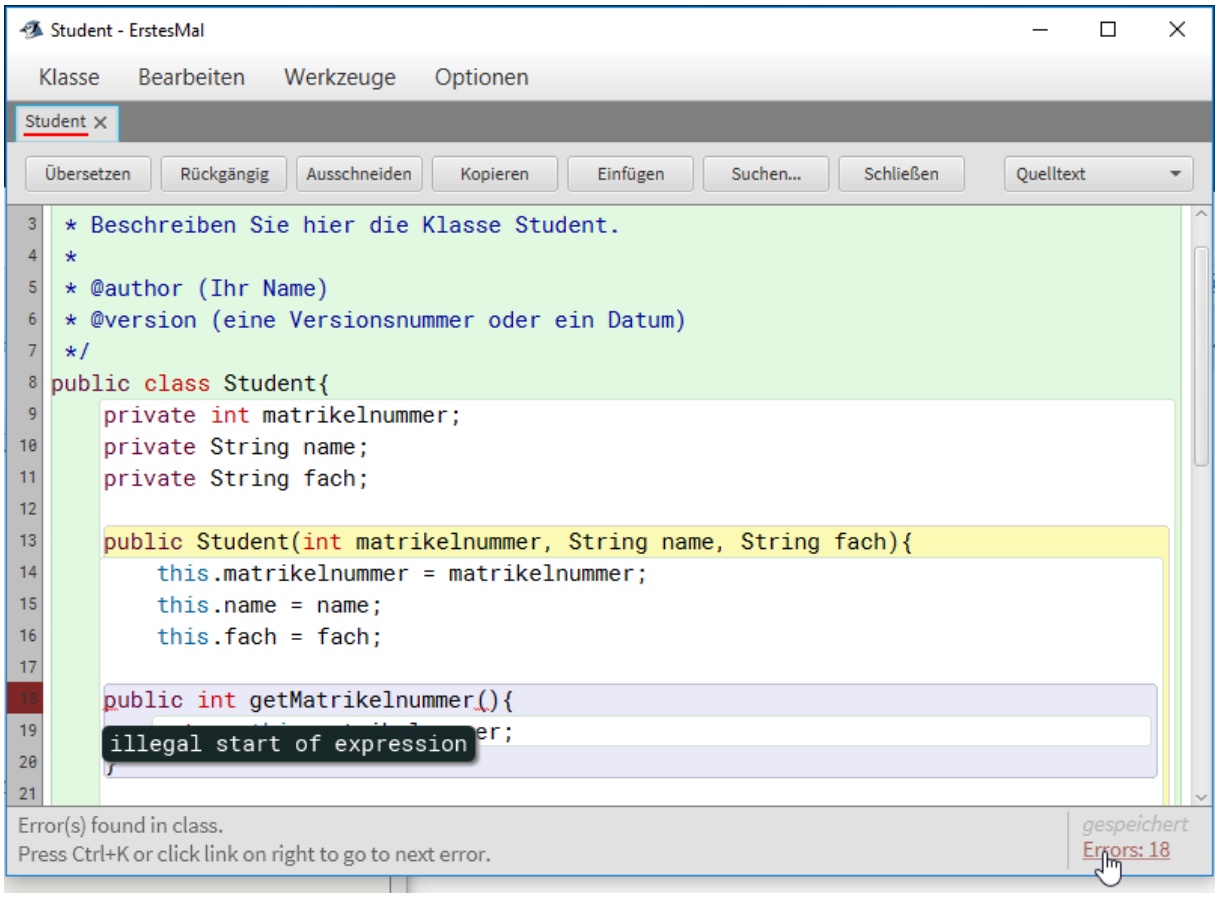



```
7   */
8   public class Student{
9       private int matrikelnummer;
10      private String name;
11      private String fach;
12
13      public Student(int matrikelnummer, String name, String fach){
14          this.matrikelnummer = matrikelnummer;
15          this.name = name;
16          this.fach = fach;
17
18      public int getMatrikelnummer(){
19          return this.matrikelnummer;
20      }
21
22      public String getName(){
23          return this.name;
24      }
25
26      public String getFach(){
27          return this.fach;
28      }
```

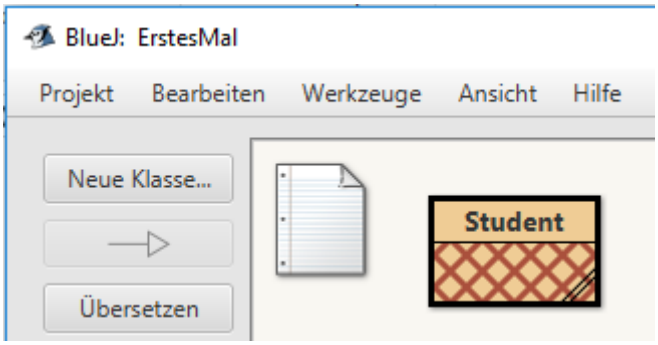
Für Klammern ist anzumerken, dass durch einen Klick auf eine Klammer die zugehörige Klammer ebenfalls markiert wird. Entspricht dies nicht den Erwartungen stimmt etwas mit den Klammern nicht. Die folgende Abbildung zeigt, dass zur öffnenden runden Klammer der Parameterliste die zugehörige schließende runde Klammer hervorgehoben wird.

```
12
13      public Student(int matrikelnummer, String name, String fach){
14          this.matrikelnummer = matrikelnummer;
15          this.name = name;
16          this.fach = fach;
```

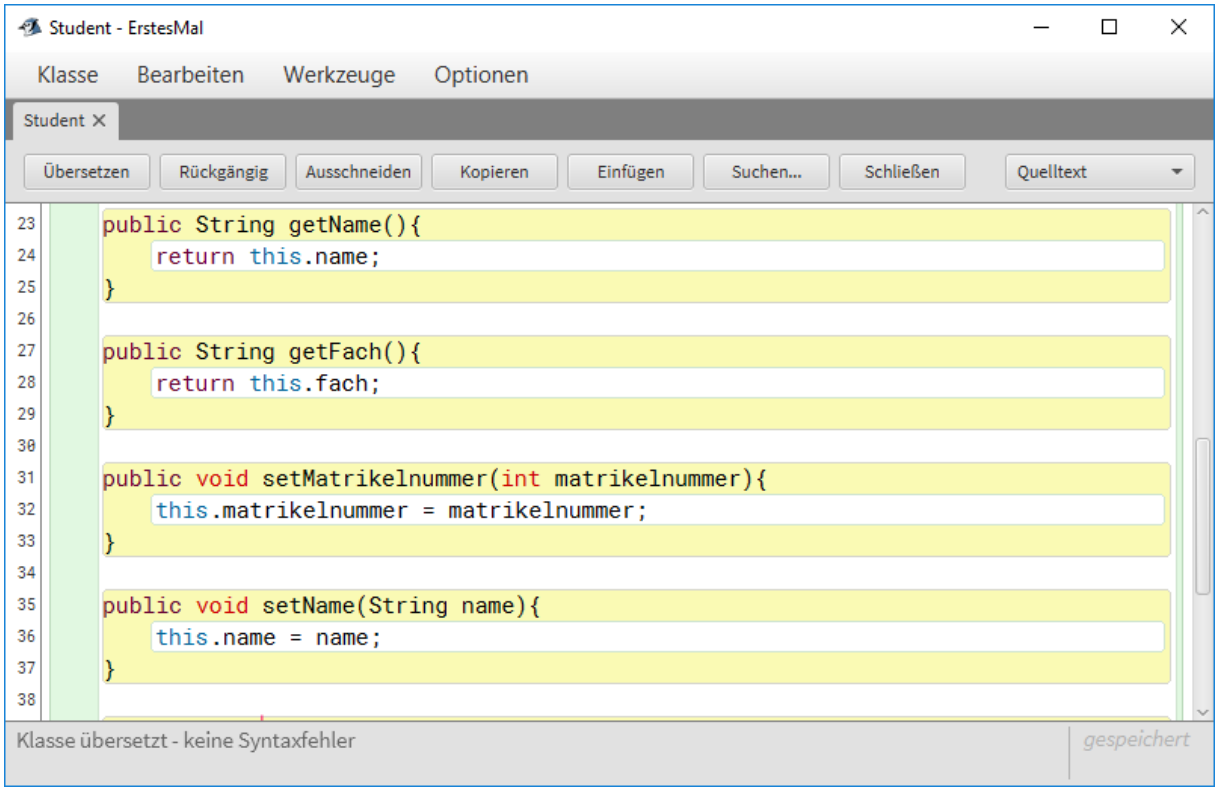
Die bisher beschriebene Fehlerausgabe erfolgt bereits beim Tippen des Programms, dabei klappt die Anzeige der Fehler leider nicht immer. Generell sollte immer versucht werden, dass Programm mit „Übersetzen“ zu kompilieren, da dabei im ersten Schritt die korrekte Syntax geprüft wird. Etwaige Fehler werden in der Fußzeile angezeigt, durch wiederholtes Klicken auf den Errors-Link rechts-unten werden die Fehler schrittweise durchlaufen und die Fehlermeldungen ausgegeben. Dies erfolgt auch, wenn mehrfach auf den „Übersetzen“-Knopf geklickt wird.



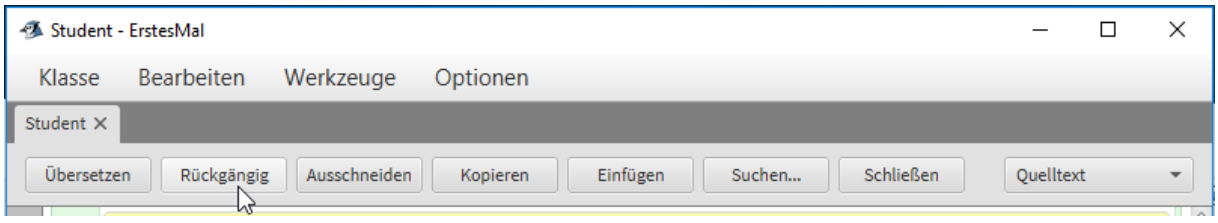

Nicht übersetzbare Klassen sind auch im Klassenfenster von BlueJ an der Schraffierung erkennbar.



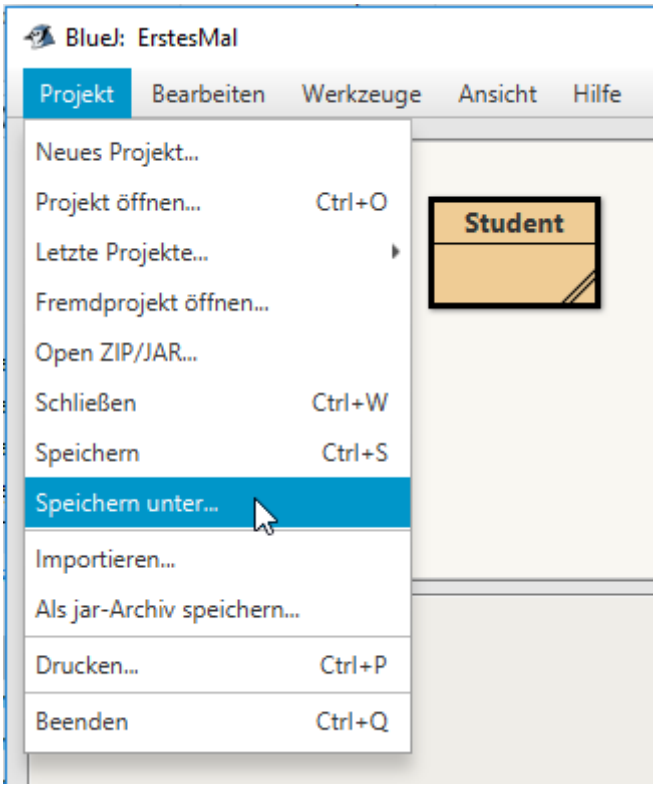

Nach der Korrektur aller Syntax-Fehler wird die Meldung „Klasse übersetzt – keine Syntaxfehler“ ausgegeben. Der Editor kann über „Close“ jederzeit verlassen werden, da Änderungen sofort gespeichert sind, was sichtbar an der Meldung „gespeichert“ rechts unten im Fenster ist.



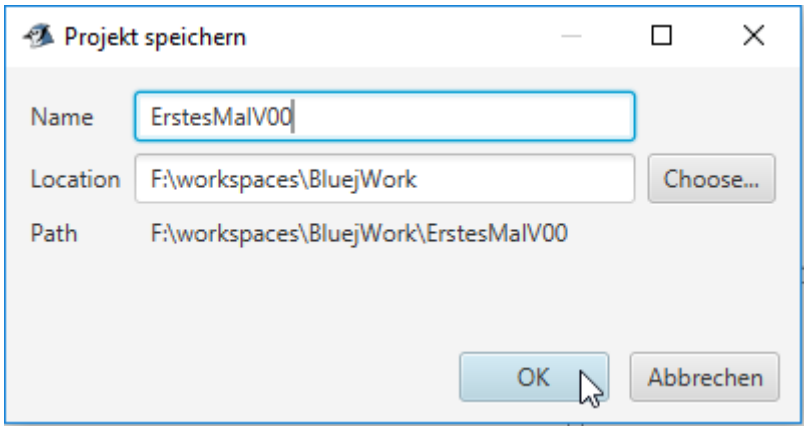

Die Besonderheit des Editors ist es damit, dass alle Änderungen sofort gespeichert werden, es also nicht durch unvorsichtiges Schließen des Editors möglich ist, Änderungen zu verlieren. Mit dem Knopf „Undo“ besteht trotzdem die Möglichkeit, die letzten Änderungen schrittweise wieder rückgängig zu machen.



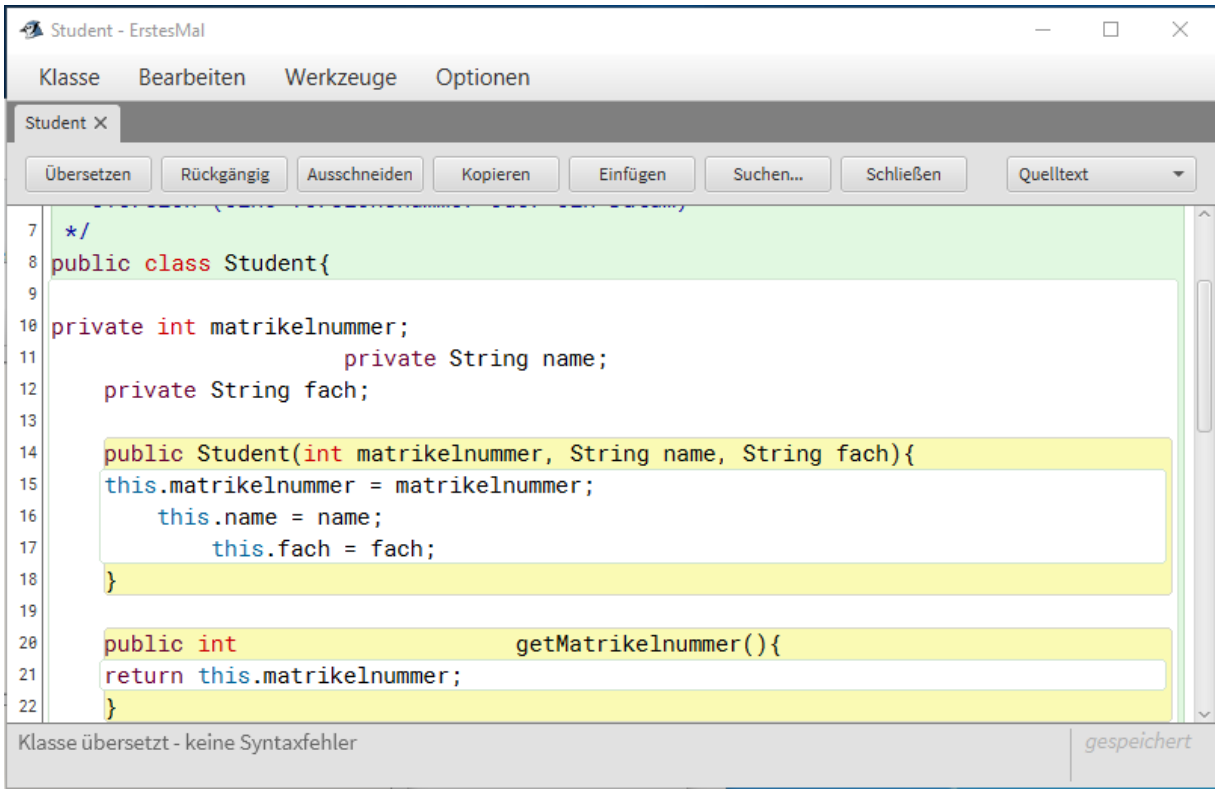
Bei größeren Programmierexperimenten kann es sinnvoll sein, Zwischenversionen der erstellten Software abzuspeichern, was hier in der Form von eigenen Projekten passiert. Versierte Entwickler können selbst herausfinden, wie BlueJ mit einer Versionsverwaltung wie Subversion oder Git verknüpft werden kann. Zur Speicherung der Zwischenlösung wird einfach „Projekt -> Speichern unter…“ gewählt.



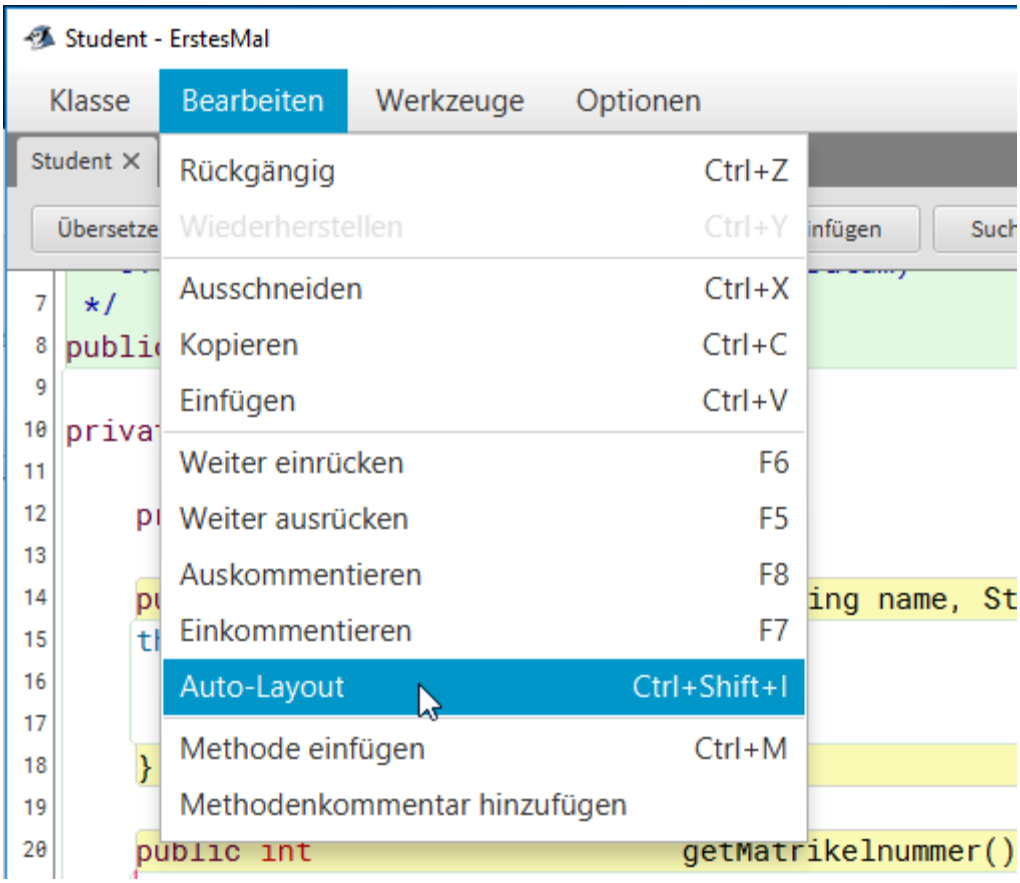

Im Namen des Projekts kann dann z. B. eine Versionsnummer eintragen werden, die beim erneuten Speichern dann manuell hochgezählt wird. Das Speichern erfolgt mit einem Klick auf den Knopf „OK“.



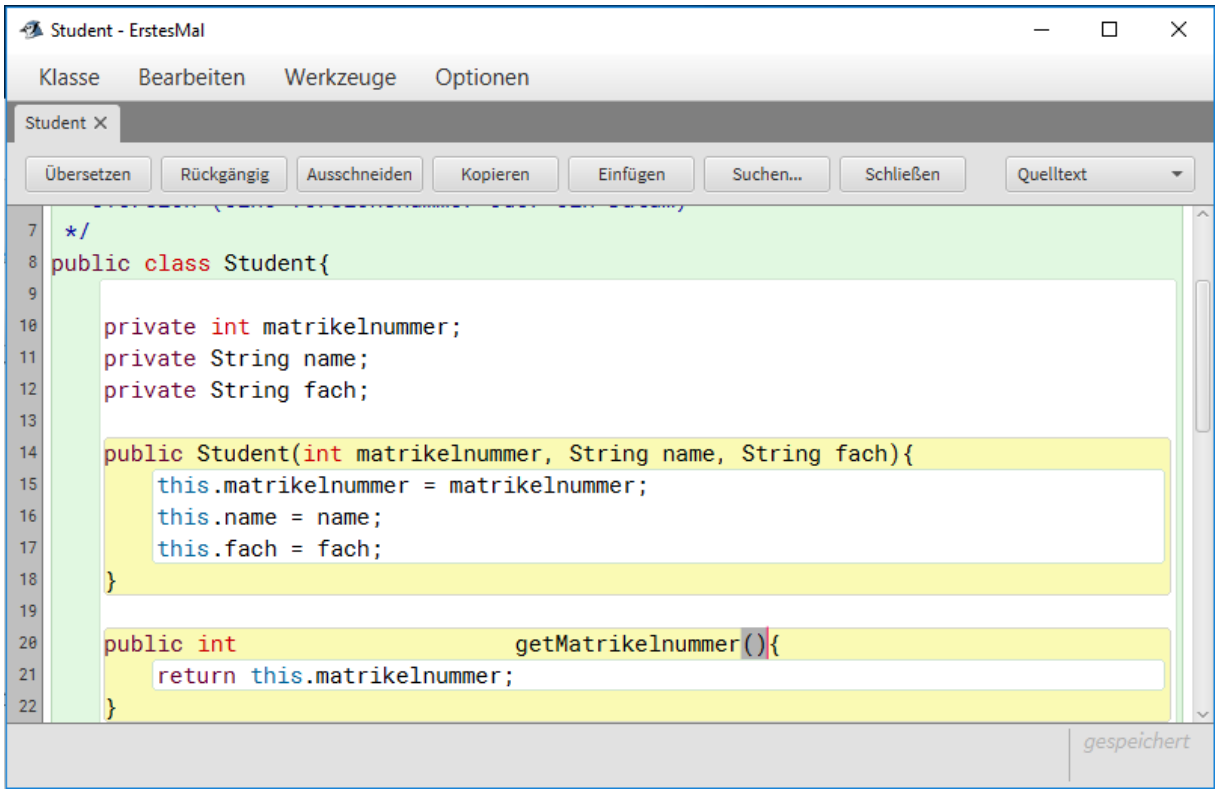

Gerade Anfänger haben durch viele notwendige Änderungen oft das Problem, dass die Formatierung des Programms verloren geht, wie das folgende Beispiel zeigt.




Hier hilft ein Klick auf „Bearbeiten -> Auto-layout“, um zu einer ordentlichen Formatierung zu kommen.

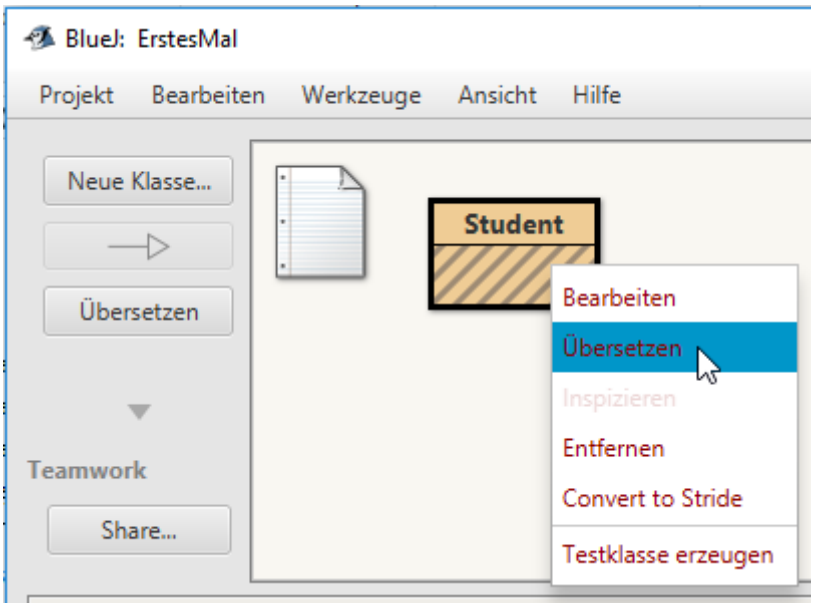



Das Ergebnis sieht dann wie folgt aus, es ist erkennbar, dass eventuell überflüssige Leerzeichen manuell entfernt werden müssen.

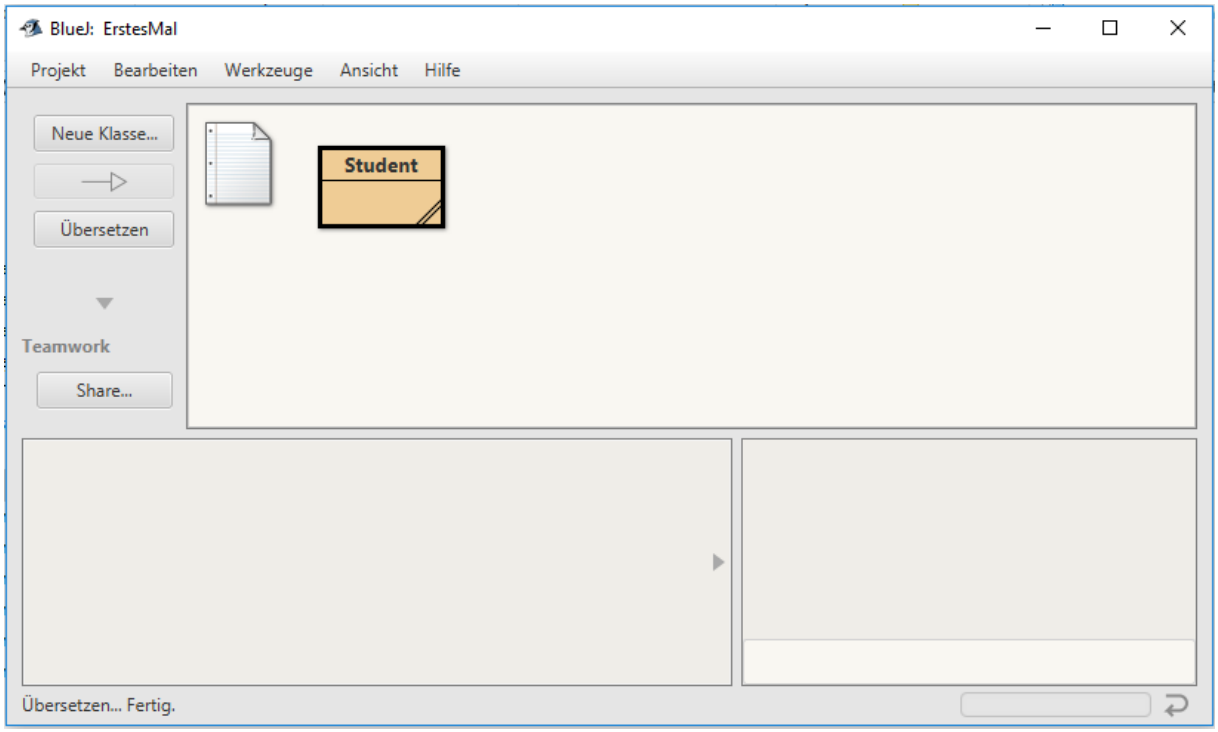



## 4.5 Kompilierung einer Klasse

Bevor Objekte von Klassen erstellt werden können, müssen diese in Byte-Code übersetzt (kompiliert) werden, der dann von der Java-Virtual Machine ausgeführt wird. Eine noch nicht übersetzte Klasse ist in der Klassen-Ansicht daran erkennbar, dass sie schraffiert ist. Dies passiert z. B., wenn der Editor geschlossen wird ohne als letzte Aktion ein „Compile“ auszuführen. Bei größeren Projekten führt die Änderung einer Klasse dazu, dass alle anderen Klassen, die Objekte dieser Klasse nutzen, auch wieder kompiliert werden müssen. Statt die Klassen wie bereits im vorherigen Unterkapitel beschrieben einzeln im Editor zu übersetzen, wird z. B. ein Rechtsklick auf der Klasse gemacht und „Übersetzen“ ausgewählt. Alternativ kann der Knopf „Übersetzen“ auf der linken Seite genutzt werden.



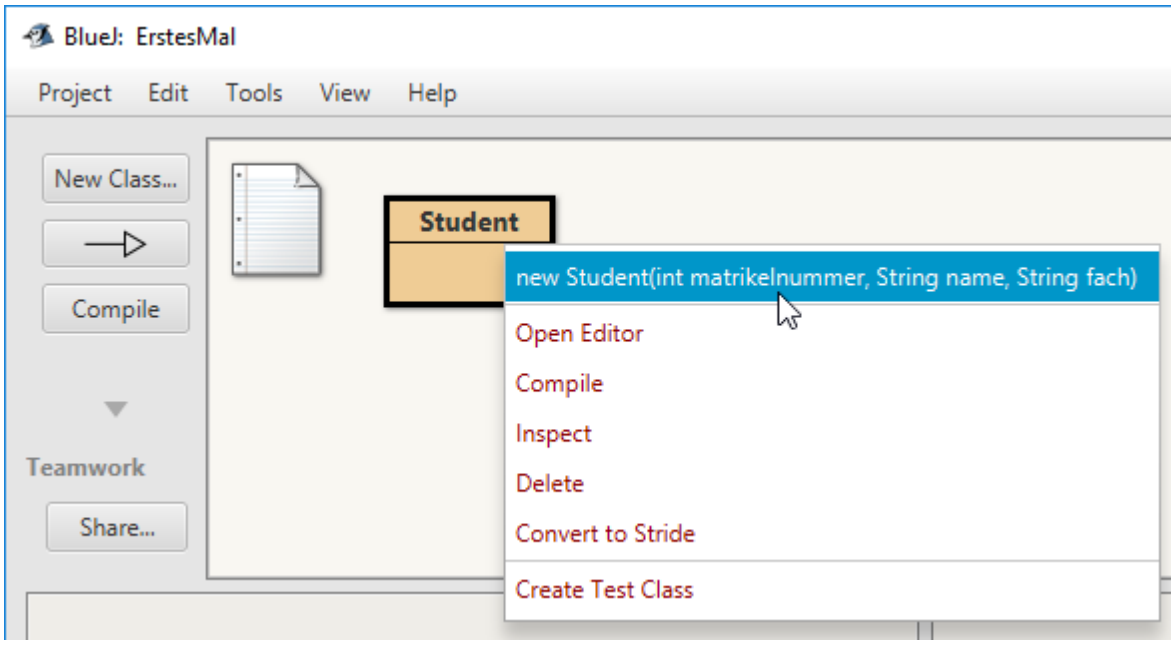

Danach wird die übersetzte Klasse als einfacher Rahmen dargestellt. Sollte das Programm gerade im Editor erstellt und dabei übersetzt worden sein, liegt sie bereits beim Schließen des Editors übersetzt zur Nutzung vor. Sollte beim Kompilieren ein Fehler gefunden werden, wird der Editor für diese Klasse geöffnet und der erste Fehler markiert.



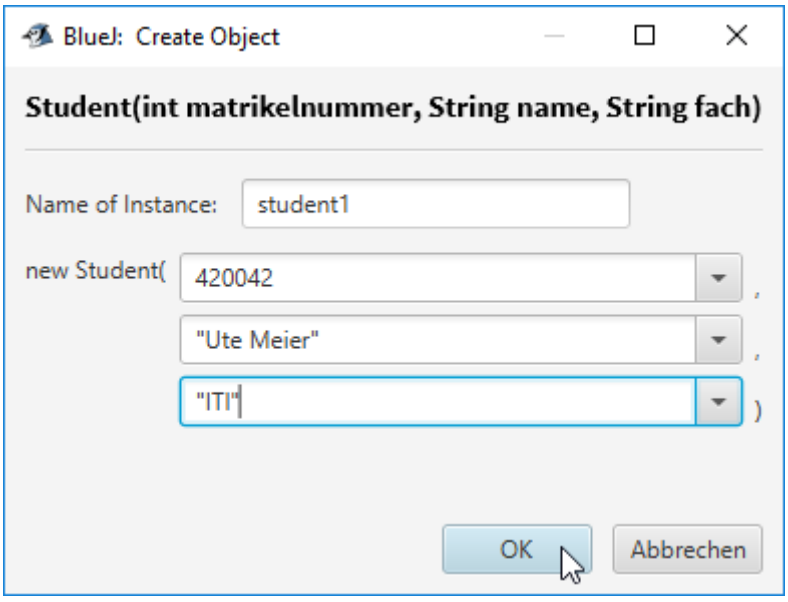

## 4.6 Erstellung eines Objektes einer Klasse

Befinden sich ein oder mehrere Klassen im Arbeitsbereich, können direkt von diesen Klassen Objekte (manchmal auch Instanzen genannt) erzeugt werden. Dies ist notwendig, damit der

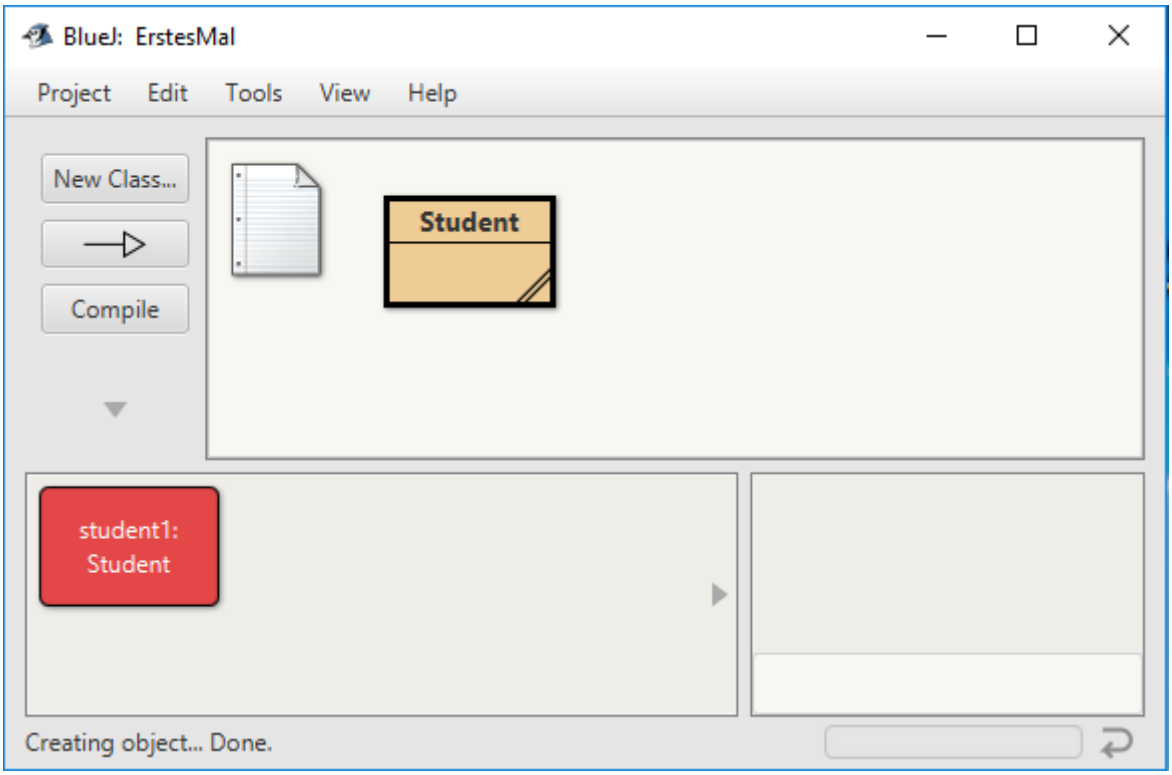
geschriebene Programmcode mit den Objektmethoden überhaupt ausgeführt werden kann. Es wird ein Rechtsklick auf der Klasse gemacht, dann werden im oberen Bereich alle nutzbaren Konstruktoren der Klasse beginnend mit „new“ angezeigt, von denen einer mit einem Klick ausgewählt werden muss.




Nun müssen für die Parameter des Konstruktors die konkreten Werte angegeben werden. Weiterhin muss jedes Objekt einen individuellen Namen erhalten, die Eingabe wird mit „Ok“ abgeschlossen. Es ist zu beachten, dass Texte (Objekte vom Typ String) in Hochkommata angegeben werden. Ohne diese Hochkommata würde die Eingabe als Variablenname interpretiert und geschaut, ob ein Wert für diese Variable bisher bekannt ist.

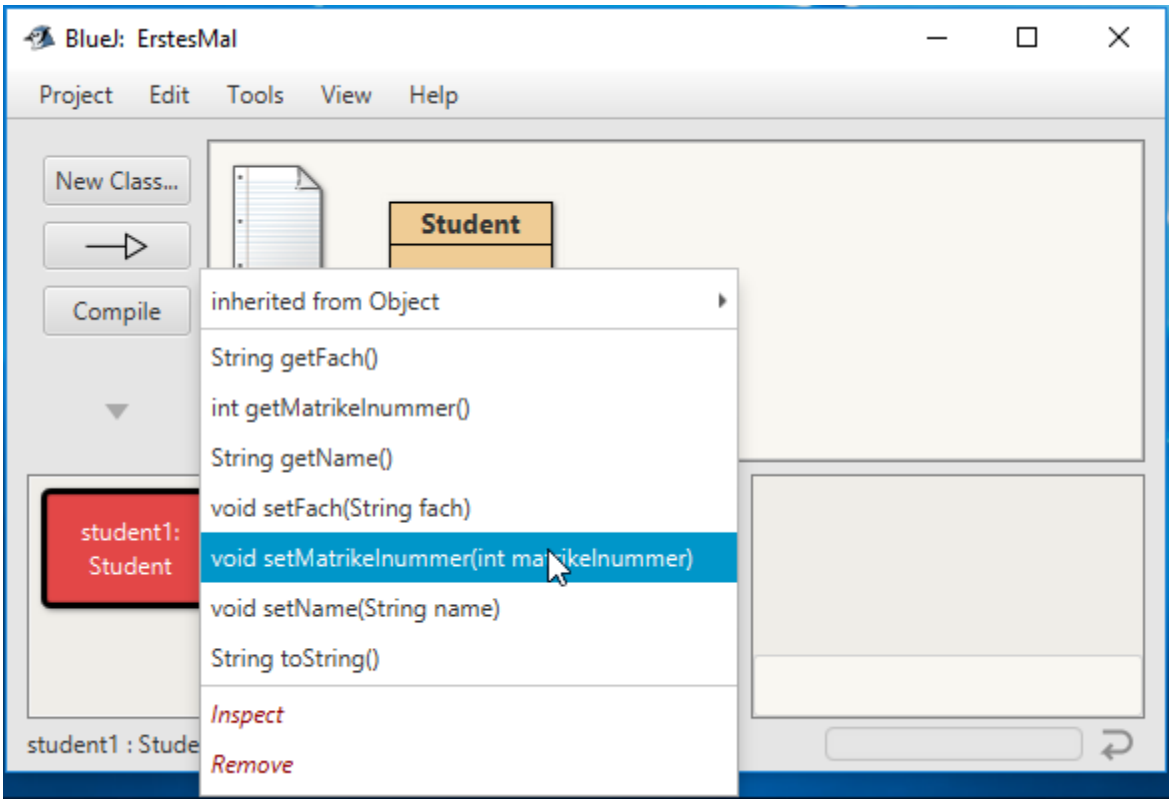



In der unteren Leiste von BlueJ, der Objektleiste, sind die momentan nutzbaren Objekte sichtbar, für die der Objektname und die Klasse angegeben werden.

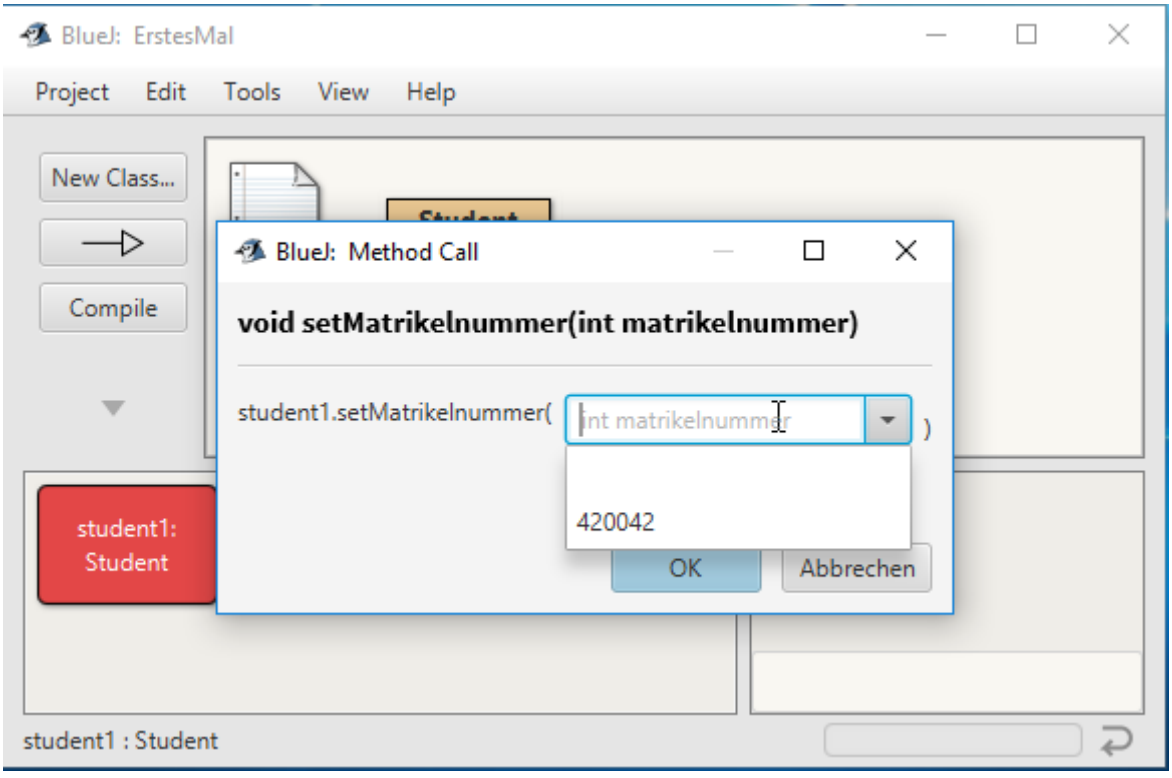



## 4.7 Ausführung von Exemplarmethoden

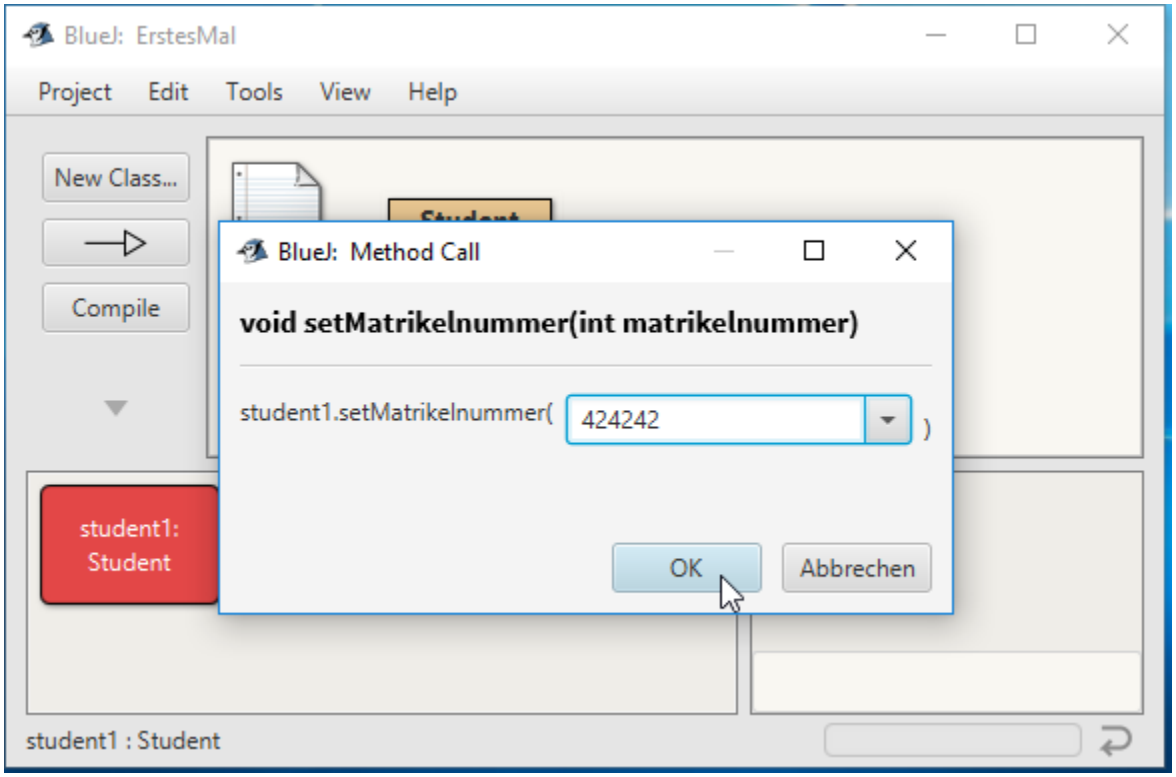
Durch einen Rechtsklick auf einem Objekt links-unten in der Objektleiste können mehrere Aktionen ausgeführt werden, wie z. B. die Änderung der Matrikelnummer mit dem Aufruf der zugehörigen set-Methode.



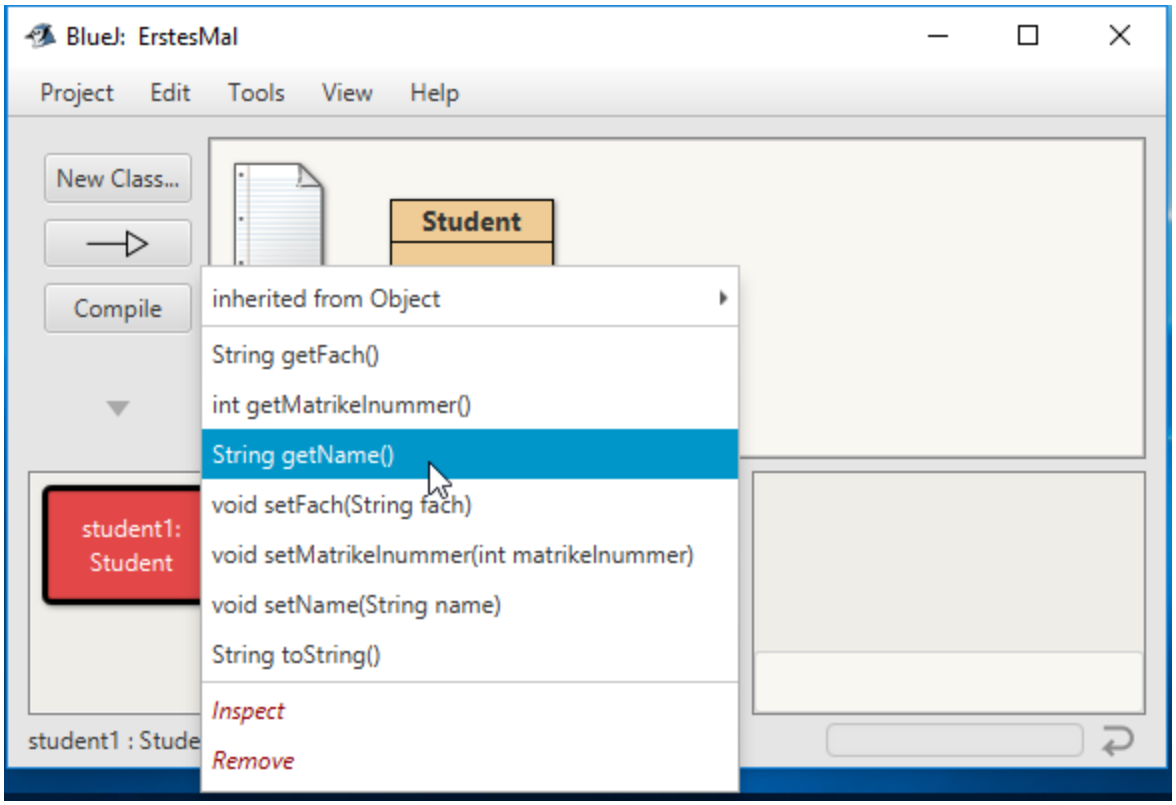

Für jede Methode müssen wieder Parameterwerte angegeben werden. Der kleine Pfeil rechts beim Eingabefeld ermöglicht eine Übersicht über bisher bekannte Eingabewerte.



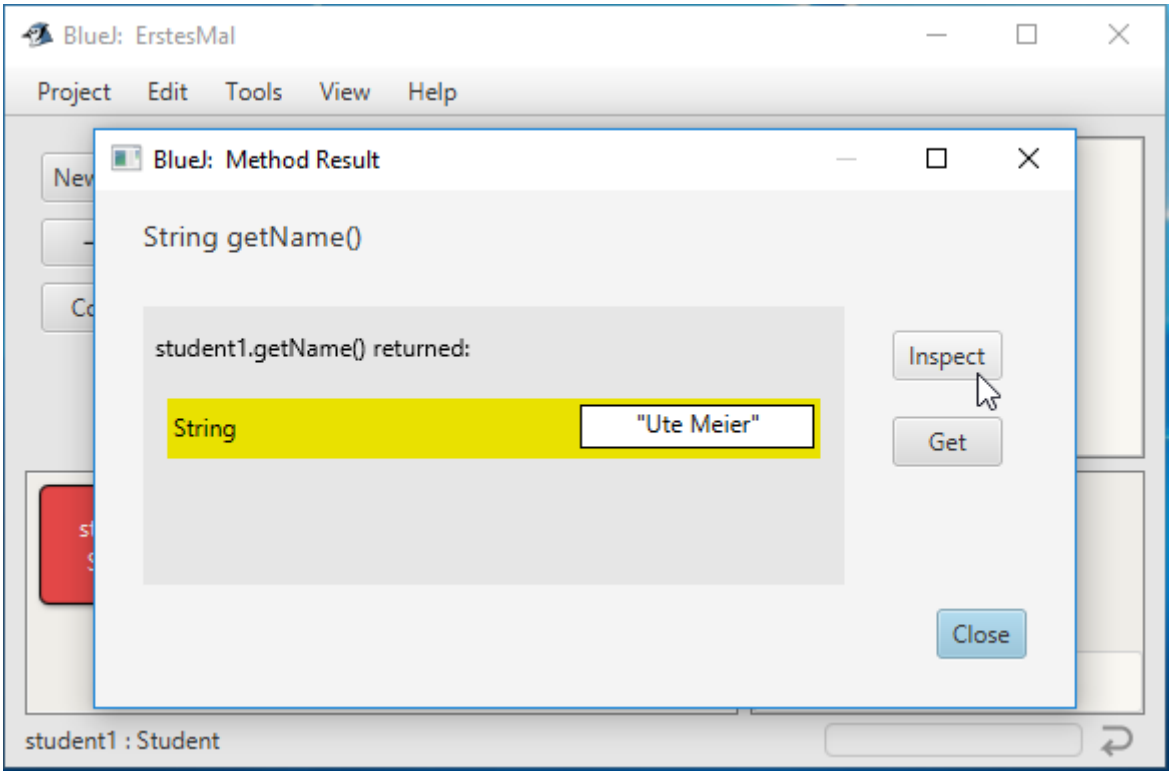
Neue Werte können einfach eingetippt und über „OK“ bestätigt werden.



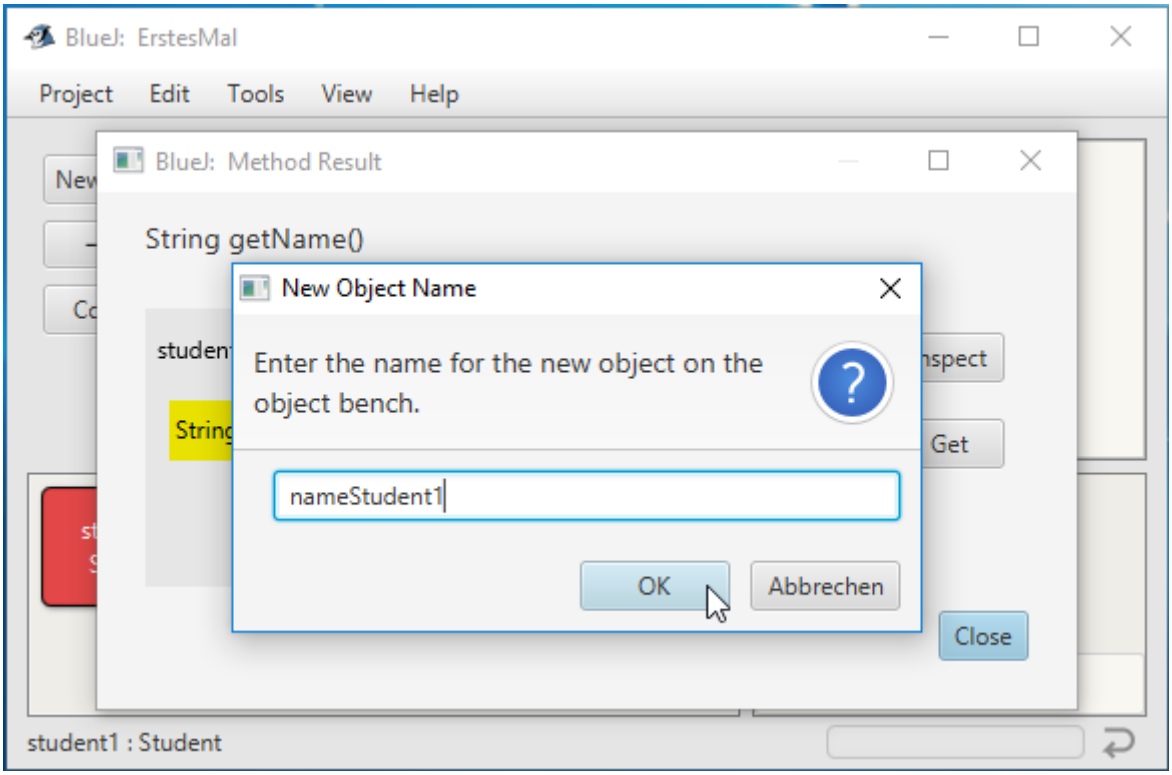

In dem vorherigen Fall wurde eine Methode ohne Rückgabewert genutzt, der Typ der Rückgabe ist void. Bei solchen Methoden ist kein weiterer Effekt bei der Ausführung beobachtbar. Hat die aufgerufene Methode ein Ergebnis, wird dieses angezeigt und kann, wenn es sich um ein Objekt handelt, weiter untersucht werden. Es wird jetzt die getName()-Methode aufgerufen.



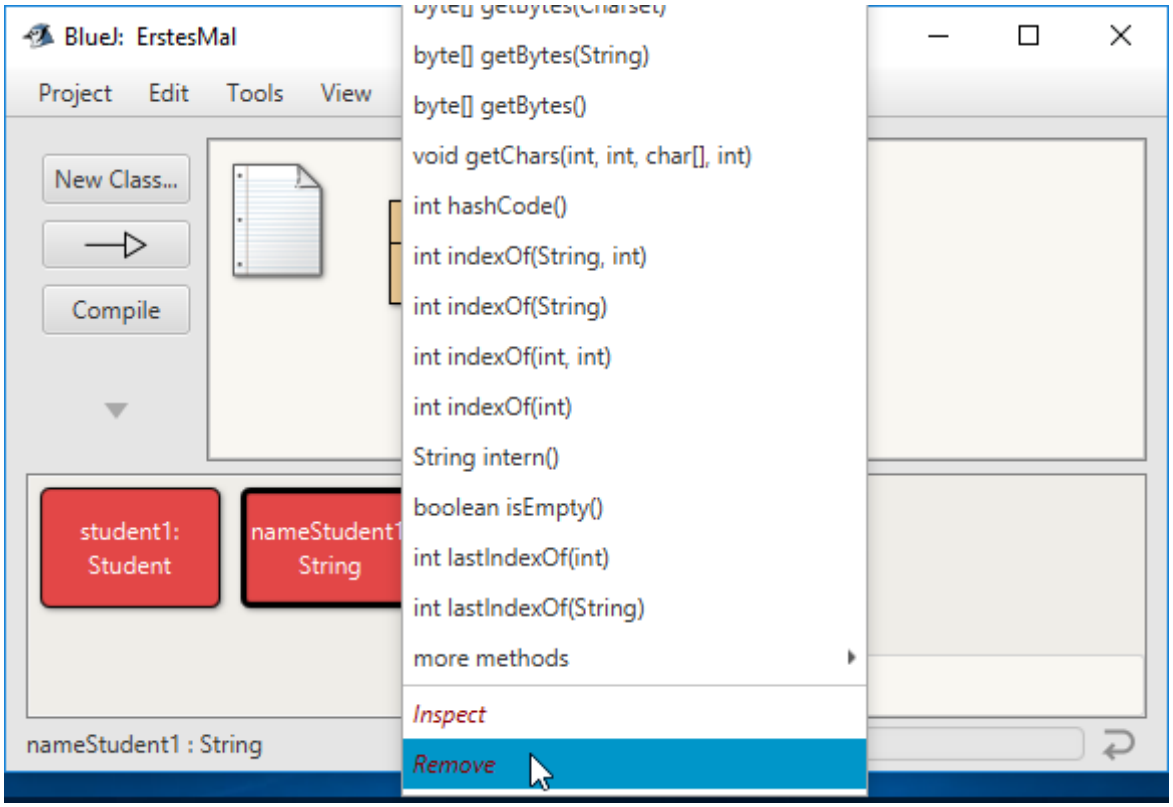

Das Ergebnis ist ein Objekt, das mit den im folgenden Abschnitt beschriebenen Verfahren auch über „Inspect“ genauer betrachtet werden kann.



Wird im vorherigen Fenster den „Get“-Knopf geklickt, wird das Ergebnisobjekt zur Objektleiste hinzugefügt, dazu muss dem Objekt zunächst ein eindeutiger Name gegeben und mit „OK“ bestätigt werden.

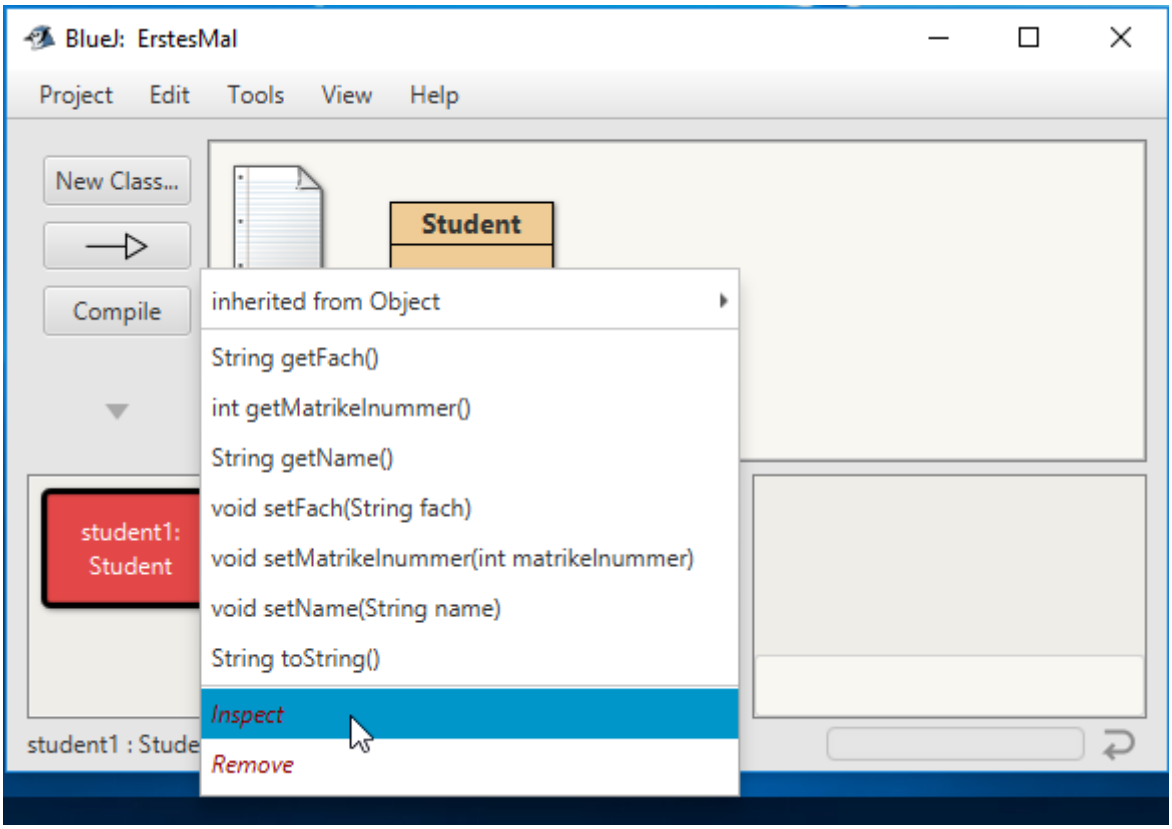



Danach befindet sich das Objekt in der Objektleiste und kann bei Bedarf über einen Rechtsklick mit der Auswahl von „Remove“ auch wieder gelöscht werden.

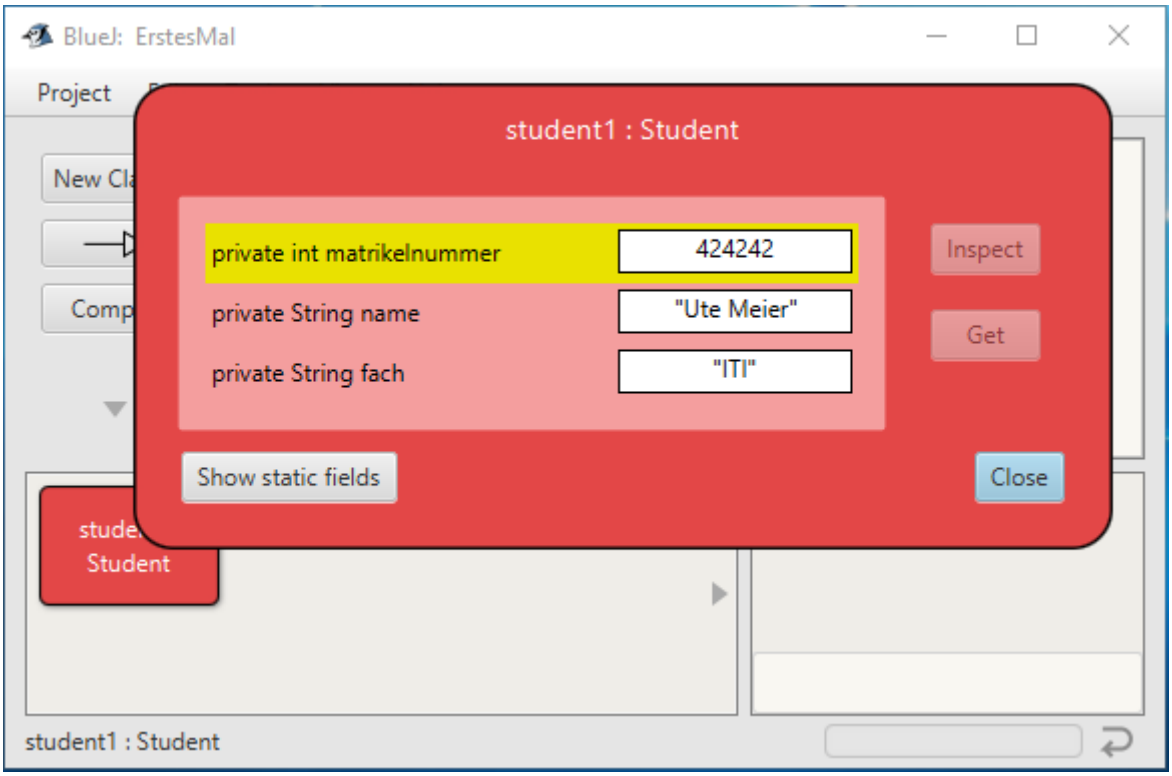



## 4.8 Genaue Analyse eines Objekts

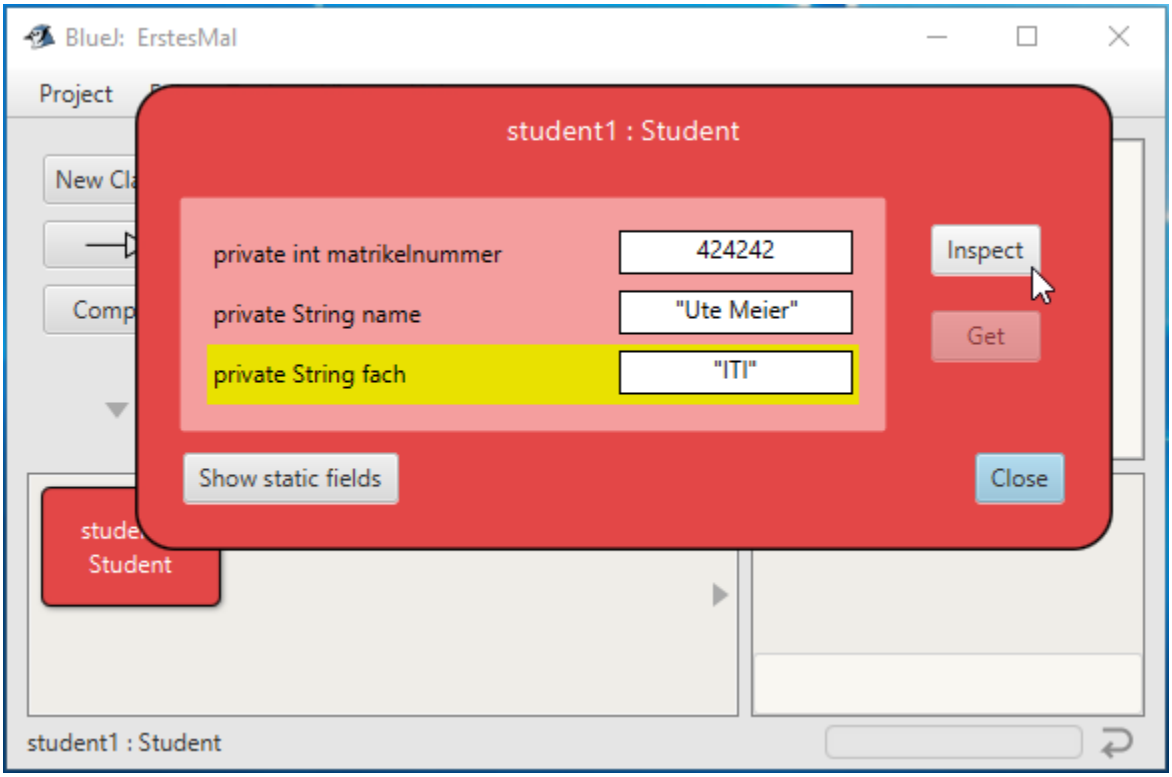
Sollen die Eigenschaften eines Objekts genauer angesehen werden, wird nach einem Rechtsklick auf dem Objekt den Punkt „Inspect“ gewählt. Alternativ führt ein Doppelklick auf dem Objekt zum selben Ergebnis.



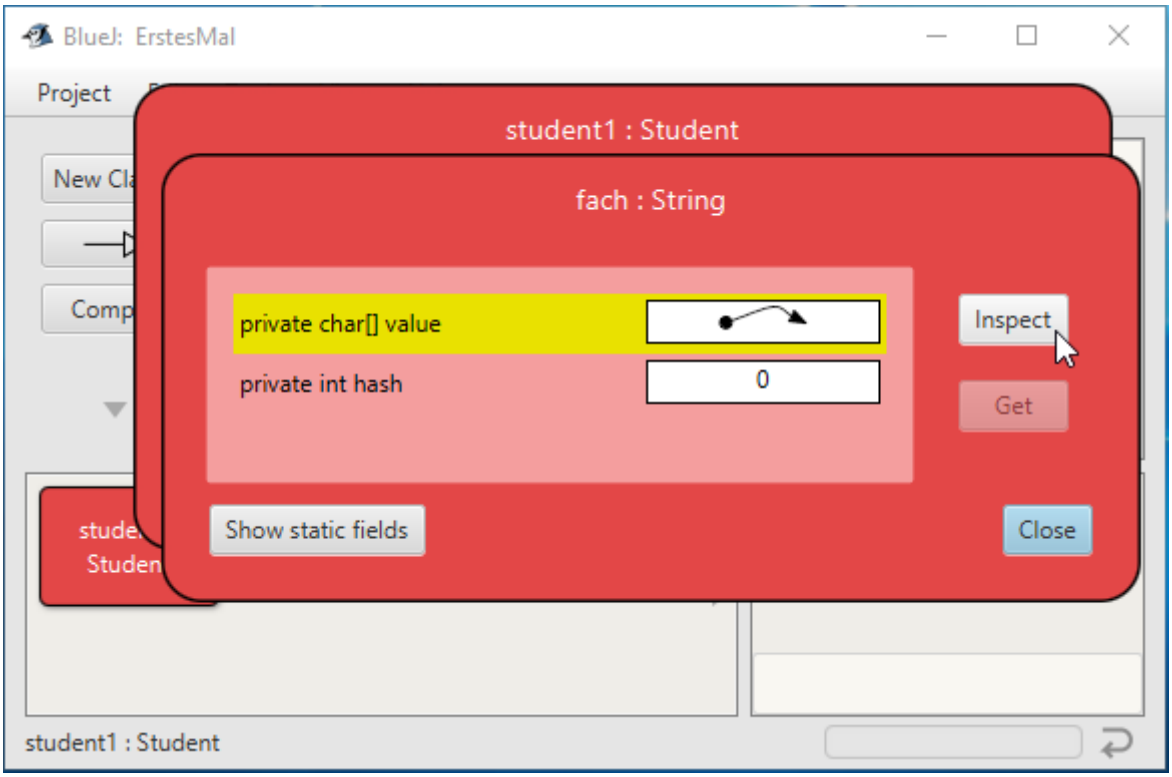

Es wird eine Übersicht über die aktuellen Werte der Exemplarvariablen ausgegeben. Über den Knopf „Show static fields“ könnte zu den Klassenvariablen gelangt werden.



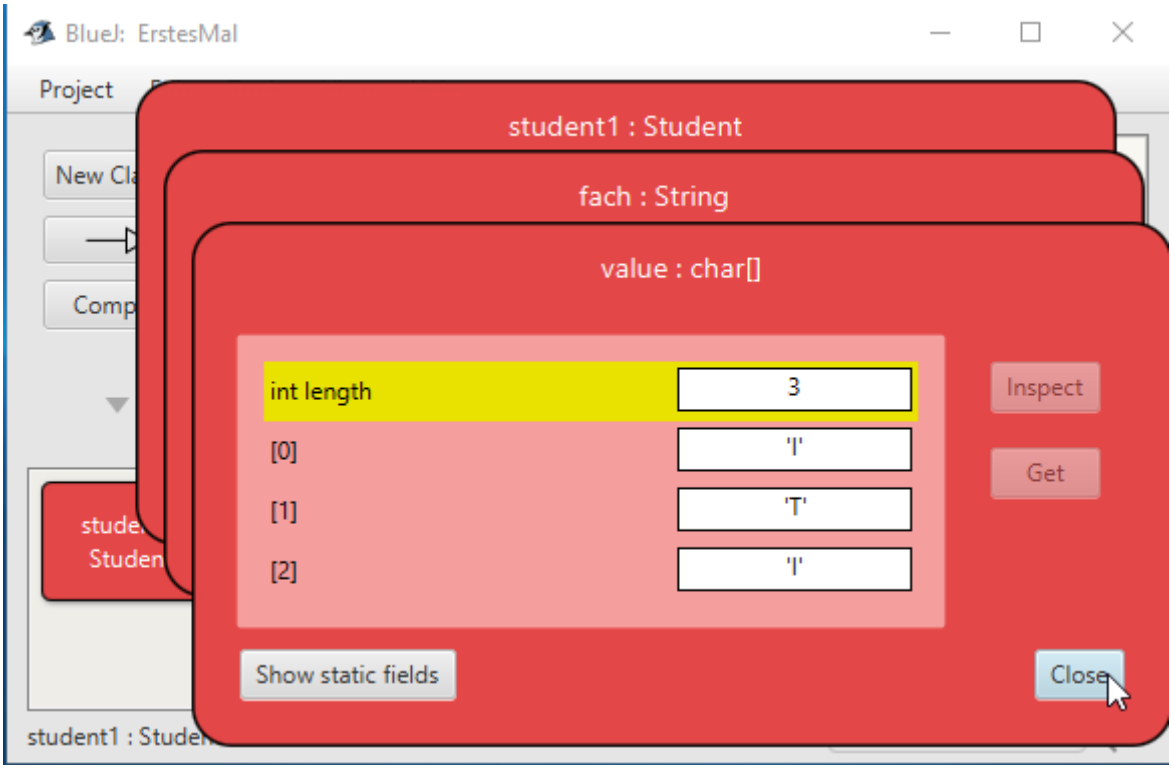
Sollte es sich bei den Werten wiederum um Objekte handeln, kann die Variable angeklickt werden, damit dann der Knopf „Inspect“ nutzbar wird und geklickt werden kann. Da Strings auch Objekte sind, ist hier ein „Inspect“ möglich.




Ohne auf Details einzugehen ist erkennbar, dass der String u. a. aus einem char-Array besteht. Beim char-Array, einem Objekt, wird wieder „Inspect“ geklickt.



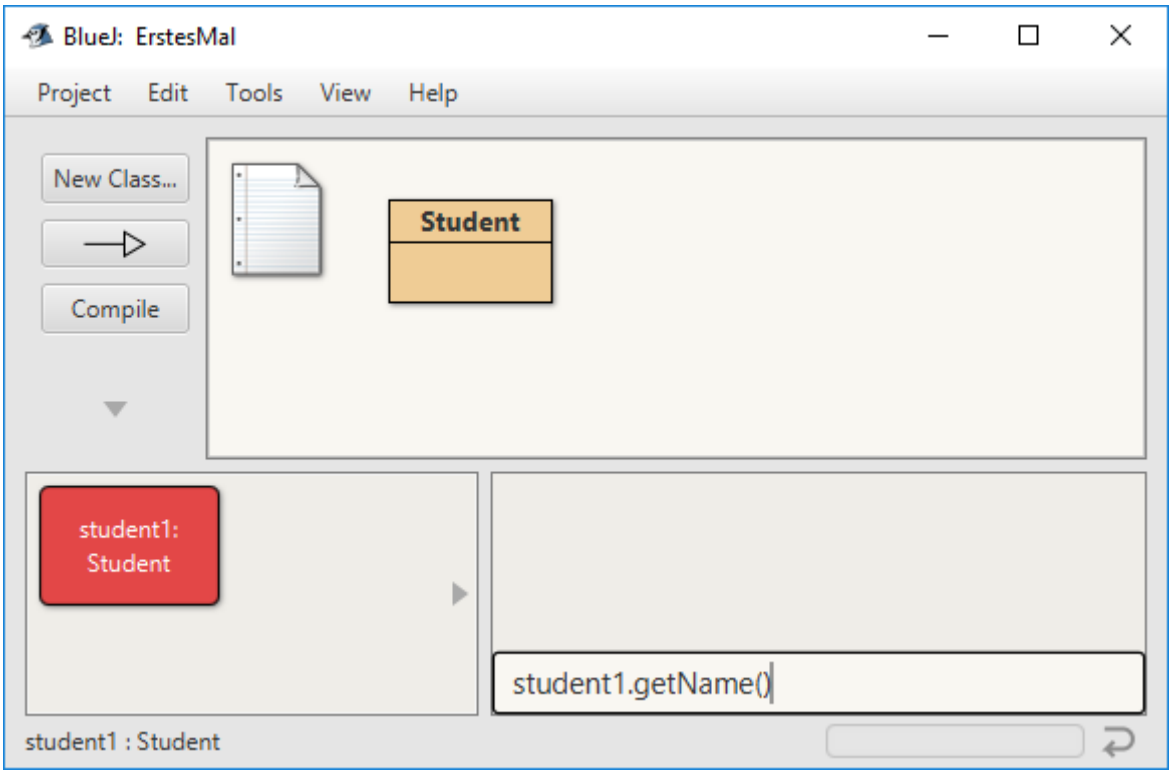

Von dem Array sind seine Länge und der Inhalt der einzelnen Felder bekannt, die mit ihrer Position im Array ausgegeben werden. Über die mehrmalige Nutzung von „Close“ können die Objekt-Ansichten wieder geschlossen werden.



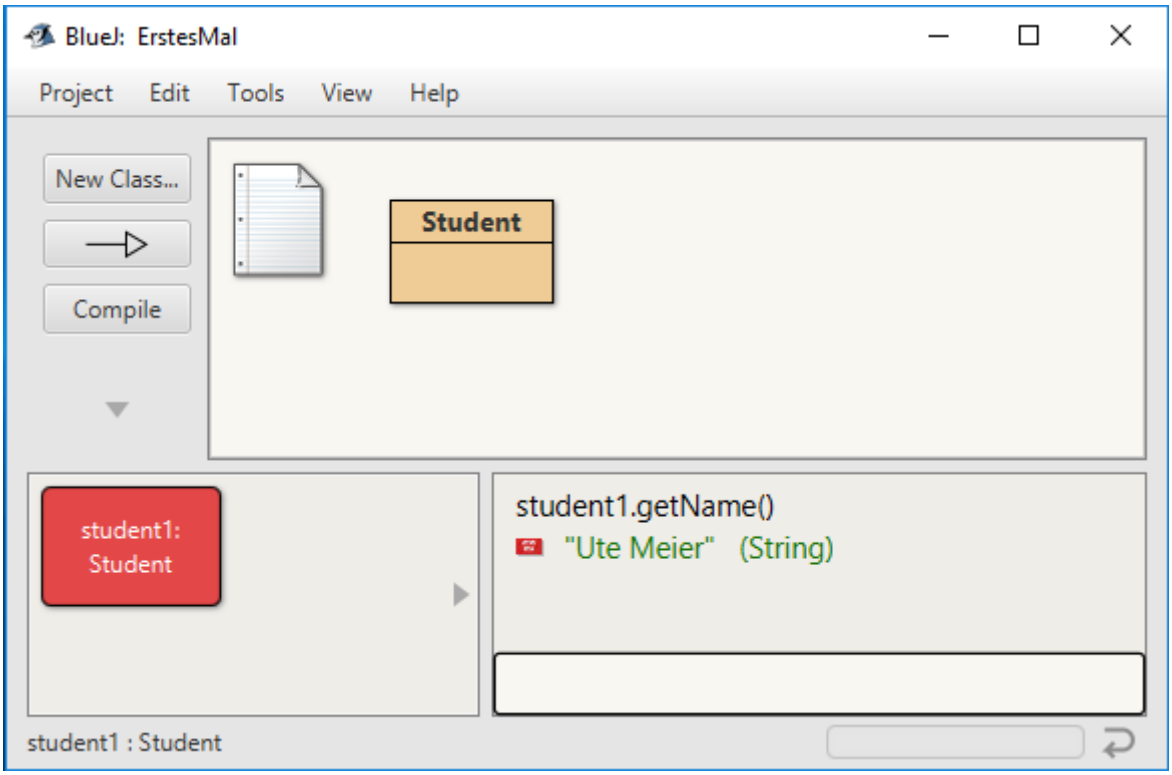
## 4.9 Weitere Nutzung des Code Pad

Erste Informationen zum Code Pad stehen im Abschnitt „4.3 Erste Experimente mit dem Code Pad“. Es sei daran erinnert, dass erstellte Objekte und in Code Pad erstellte Skripte nach der Beendung von BlueJ nicht mehr zur Verfügung stehen, da sie als nicht permanent nutzbares Material angesehen werden.

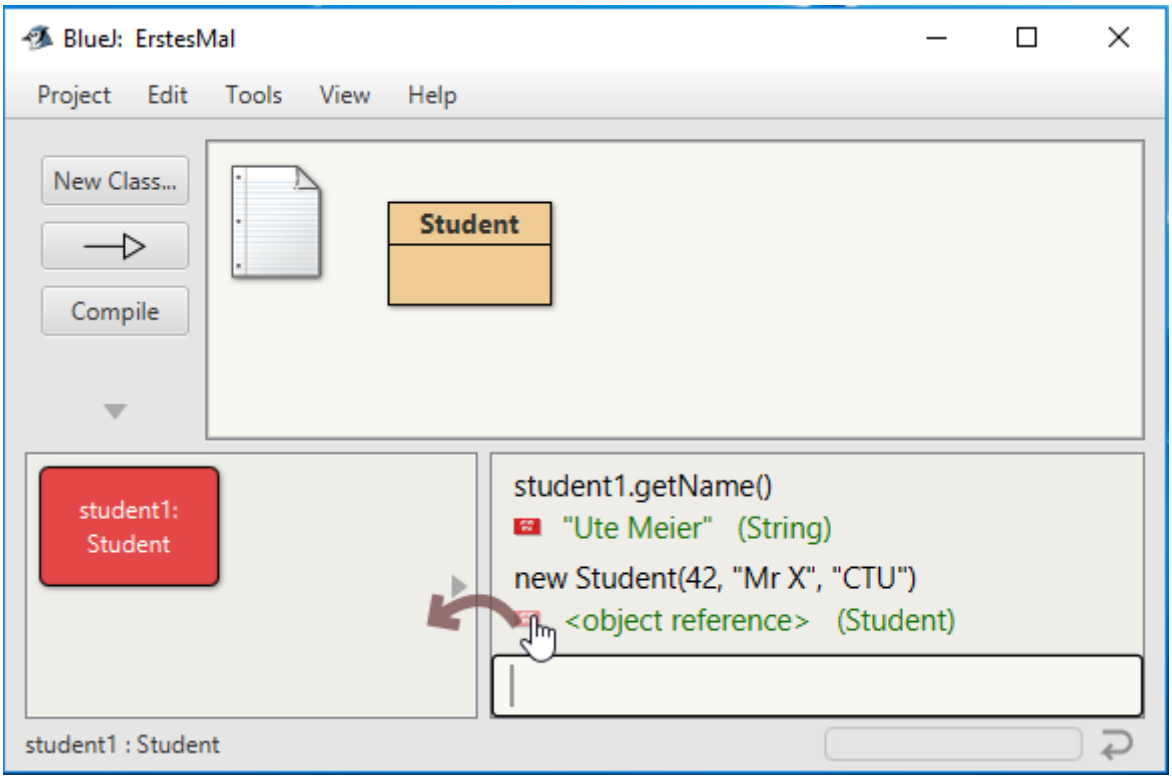
Zunächst können im Code Pad auch einfache Java-Anweisungen mit Methodenaufrufen auf Objekten aus der Objekt-Leiste eingetippt werden.



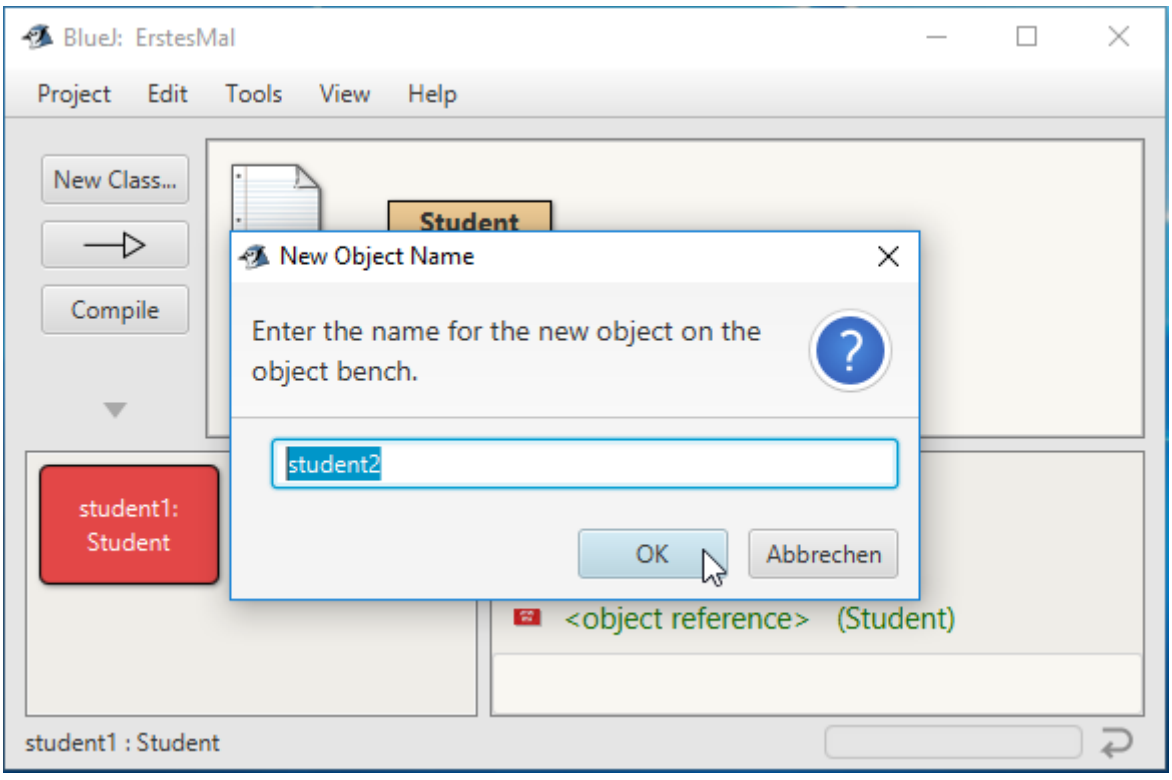

Nach Drücken der „Return“-Taste wird das Programmfragment ausgeführt und das Ergebnis des Ausdrucks mit seinem Typen angezeigt.



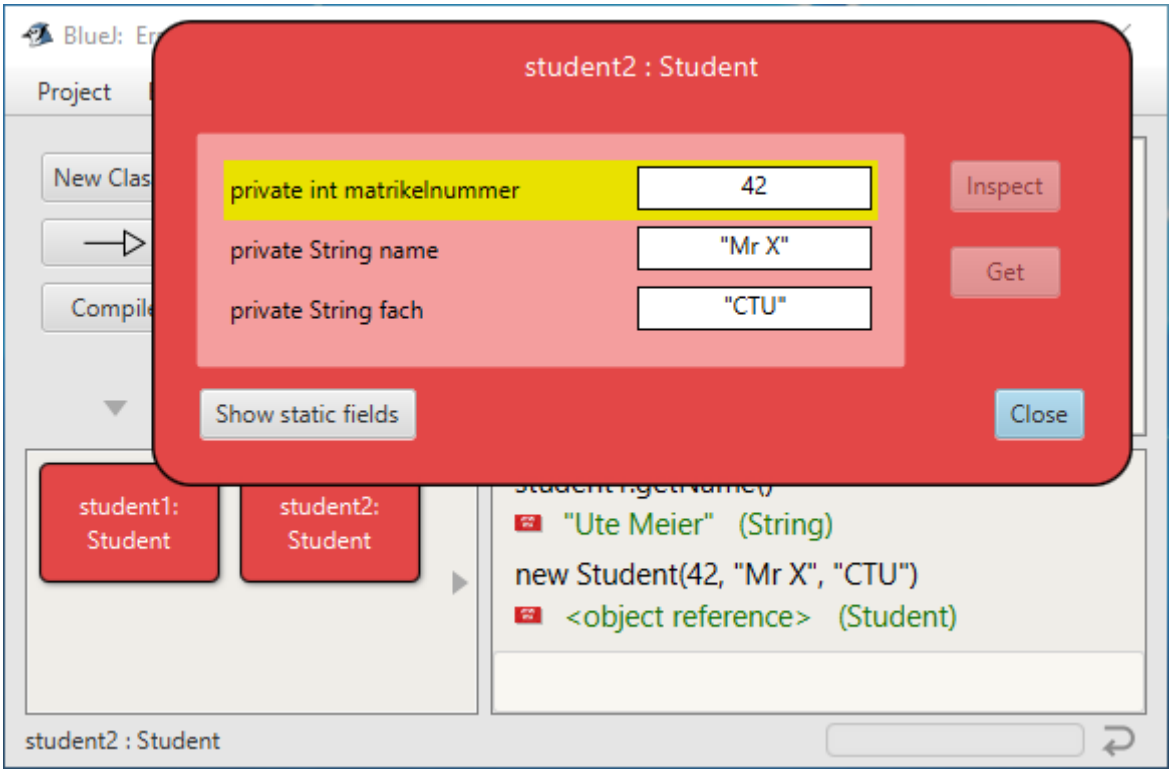

Ergebnisse solcher Berechnungen können auch Objekte sein (war bei dem ersten Ausdruck bereits der Fall), dann wird am linken Rand von Code Pad ein kleiner roter Kasten als Repräsentant für das Ergebnisobjekt angezeigt. Es ist zu beachten, dass hinter der Zeile mit new kein Semikolon steht, es handelt sich dadurch um einen Ausdruck, der ausgewertet wird. Wird der Mauszeiger auf den roten Kasten gezogen, erscheint ein gebogener Pfeil, der auf die Objektleiste zeigt.



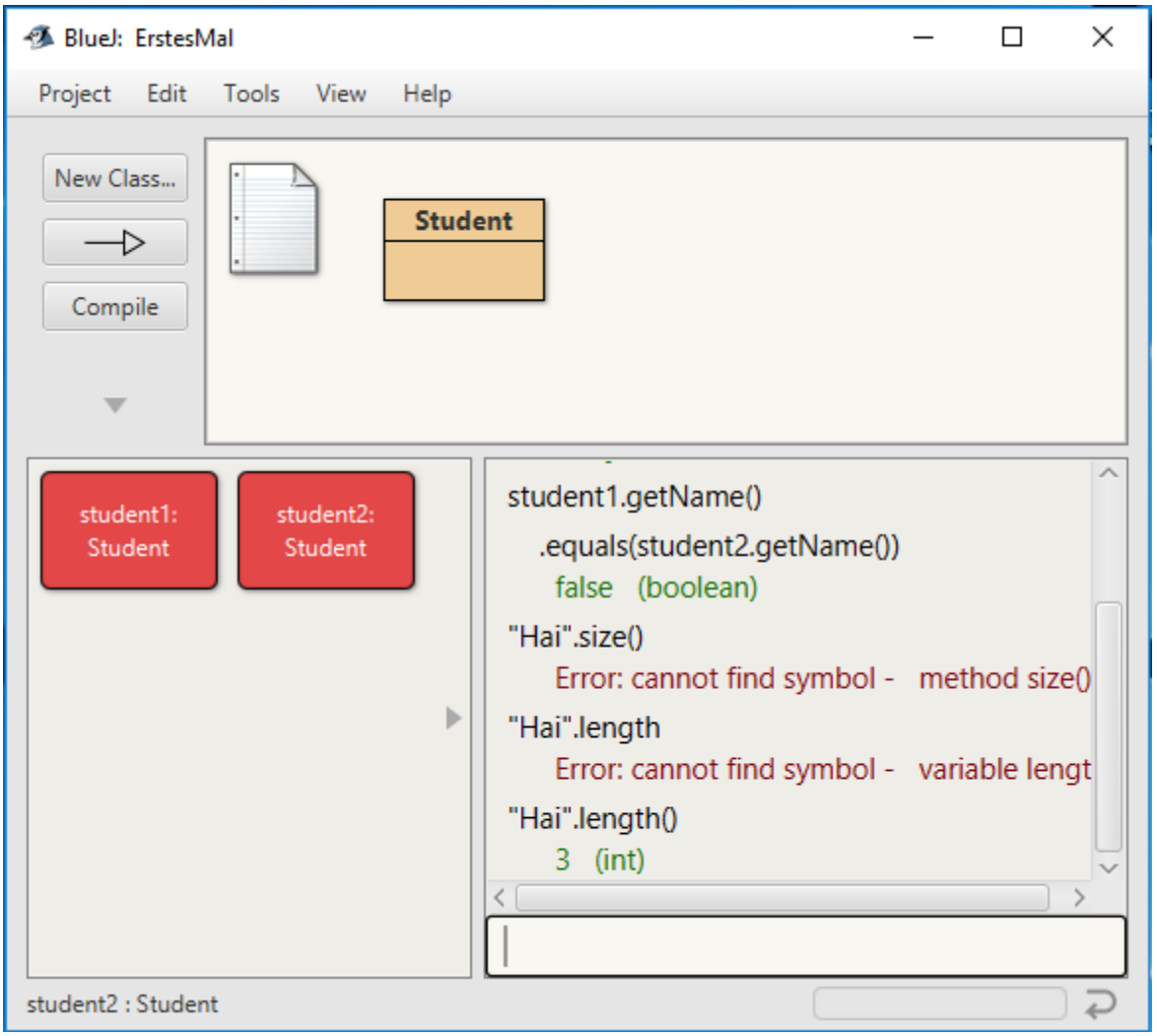

Dieses Objekt kann durch Drücken der linken Maustaste in die Objekt-Leiste übernommen werden. Es wird nach einem Namen für das Objekt gefragt, der dann mit „OK“ bestätigt wird. Variablennamen aus dem Code Pad werden leider nicht übernommen, da es nicht sichergestellt ist, dass das Objekt überhaupt einen Namen hat, was beim Ergebnis eines Ausdrucks, wie im vorherigen Fall nicht der Fall sein muss.



Danach kann das Objekt genau wie die anderen existierenden Objekte genutzt werden. Die folgende Abbildung zeigt die Ausgabe, nachdem ein „Inspect“ auf dem Objekt ausgeführt wurde.



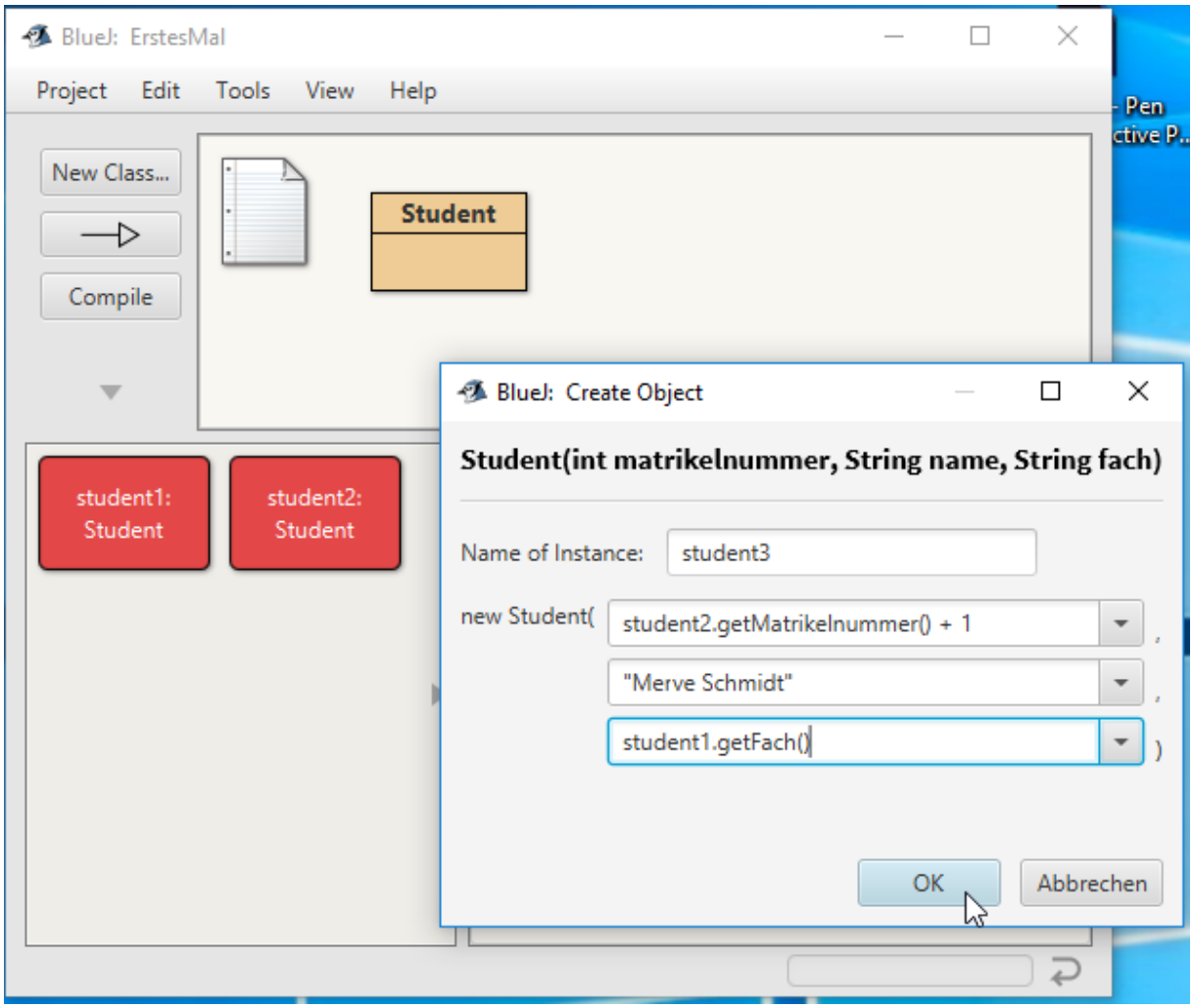

Das folgende Bild zeigt weitere Java-Programmzeilen, die einfach im Code Pad ausgeführt werden können. Um Code über mehrere Zeilen schreiben zu können, muss am Zeilenende jeweils „Shift“+“Return“ zusammen gedrückt werden.



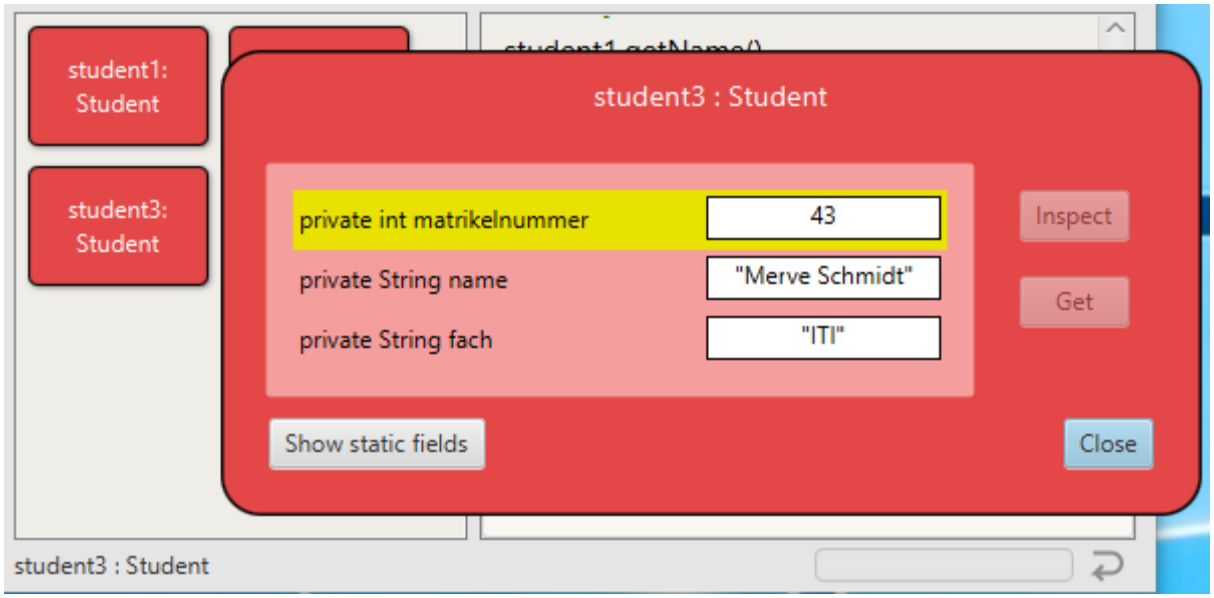

Bei einem Tippfehler oder sollen frühere Zeilen wiederholt werden, ist dies durch die Pfeiltasten (nach oben für vorherigen Befehl) aufrufbar. Die Zeilen werden einzeln eingeblendet, können aber mit „Shift“+“Return“ wieder zu einem längeren Befehl zusammengesetzt werden.

## *4.10 Weitere Nutzung von Objekten*

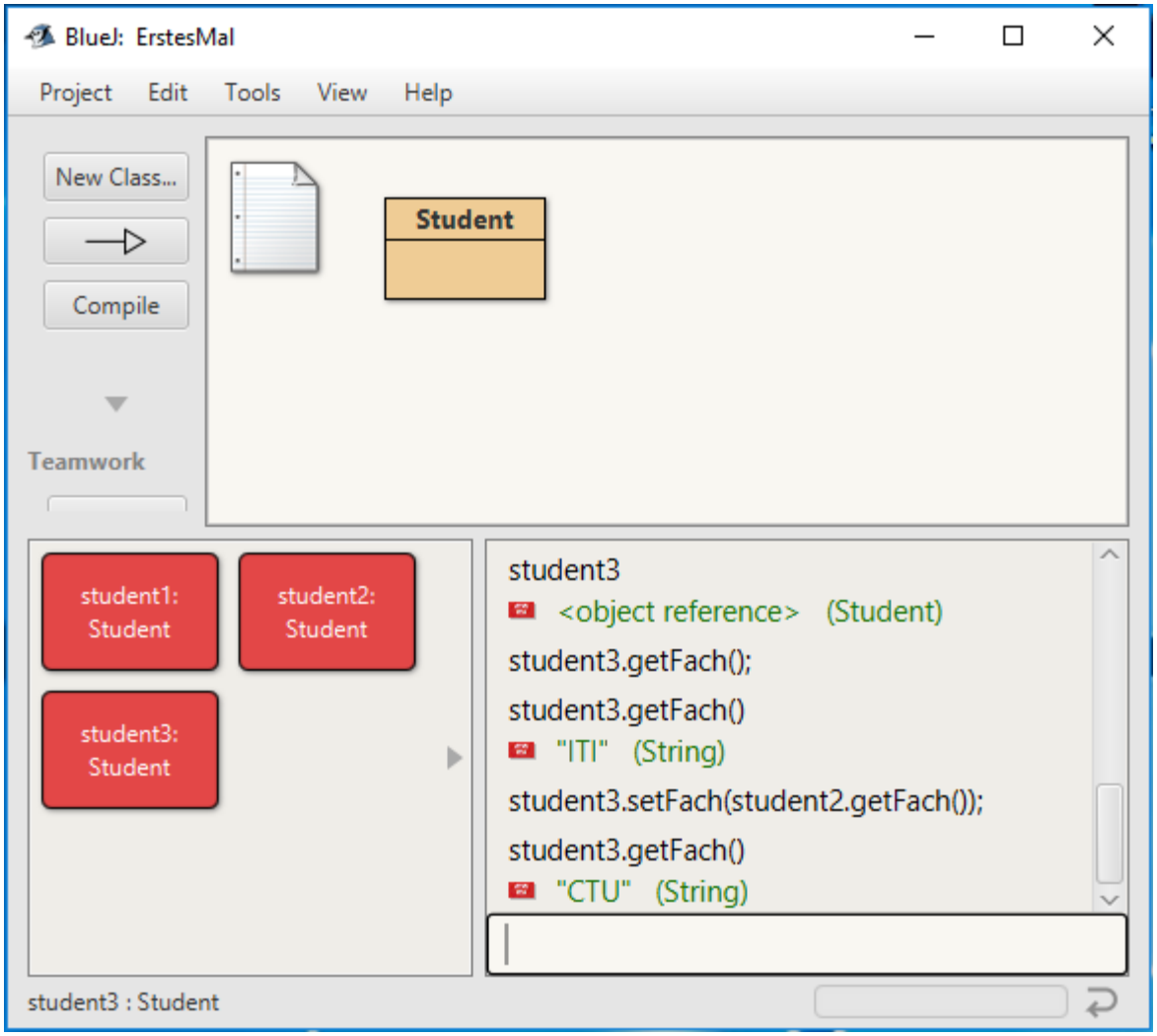
Objekte in der Objektleiste können auch weiter bei der Objekterstellung oder bei Methodenaufrufen als Parameter genutzt werden. Die folgende Abbildung zeigt, wie zunächst im Code Pad ein neues Objekt der Klasse Student angelegt wird und bei den Parametern Ausdrücke genutzt werden, die auf Objekte der Objektleiste zugreifen.




Die Inspektion zeigt, dass der Ansatz erfolgreich war.



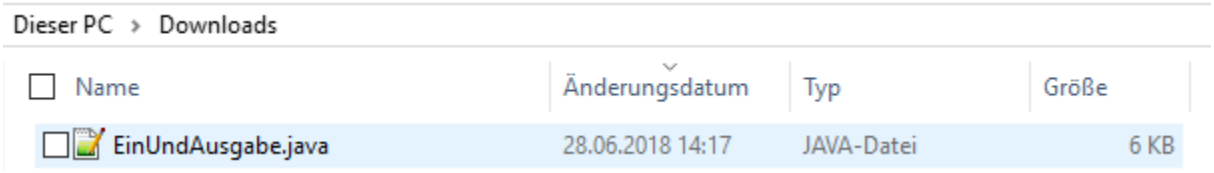
Die folgende Abbildung zeigt, dass auch von Code Pad aus auf Objekte der Objektleiste über deren Namen zugegriffen werden kann. Es wird auch deutlich, dass zur Ausgabe von Rückgabewerten kein Semikolon in die Zeile stehen darf. Es handelt sich wieder um einen Ausdruck. Die Zeile mit dem Semikolon wird als Befehl aufgefasst. Dieser Aufruf der Methode ist syntaktisch korrekt. Da in der Zeile keine Zuweisung an eine Variable stattfindet, wird das Ergebnis einfach ignoriert.



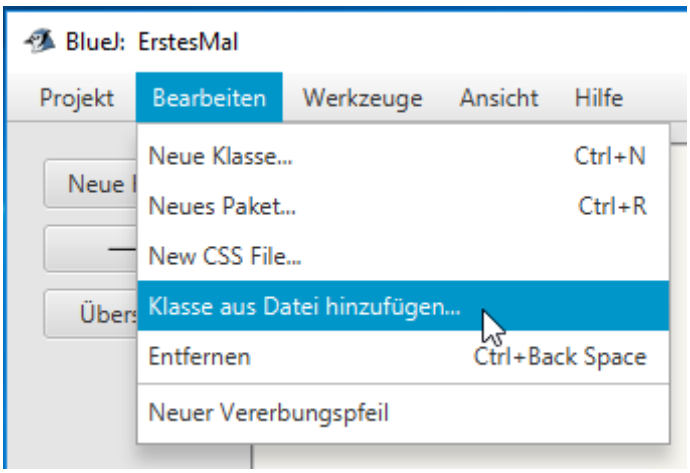
# 5 Arbeiten mit mehreren Klassen in BlueJ

## *5.1 Laden externer Klassen*

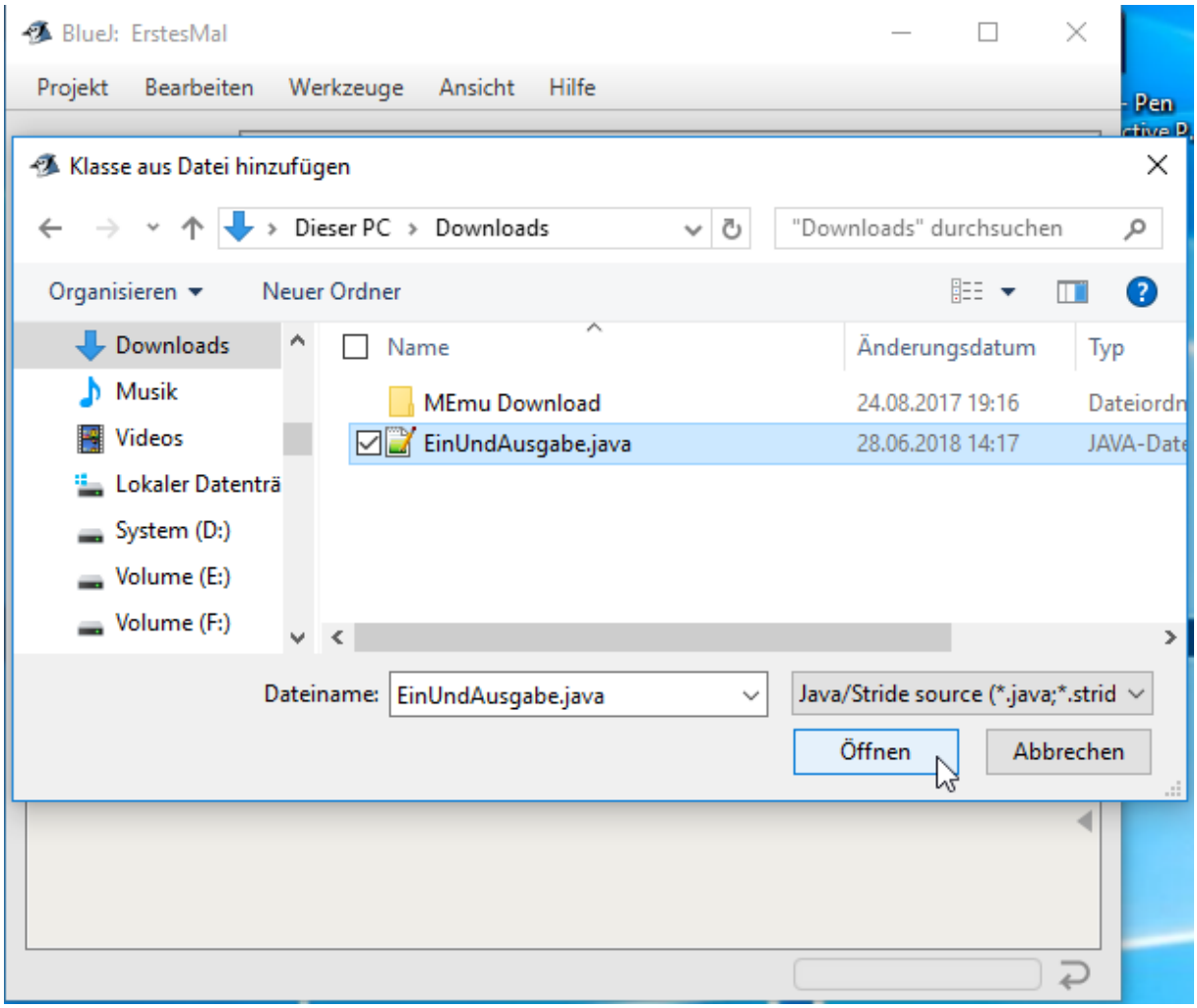
Sollen Klassen genutzt werden, die in anderen Projekten entwickelt wurden, sind diese in das neue Projekt zu integrieren. Als Beispiel soll die Klasse EinUndAusgabe.java geladen werden, die sich hier als Beispiel im Download-Ordner des Rechners befindet.



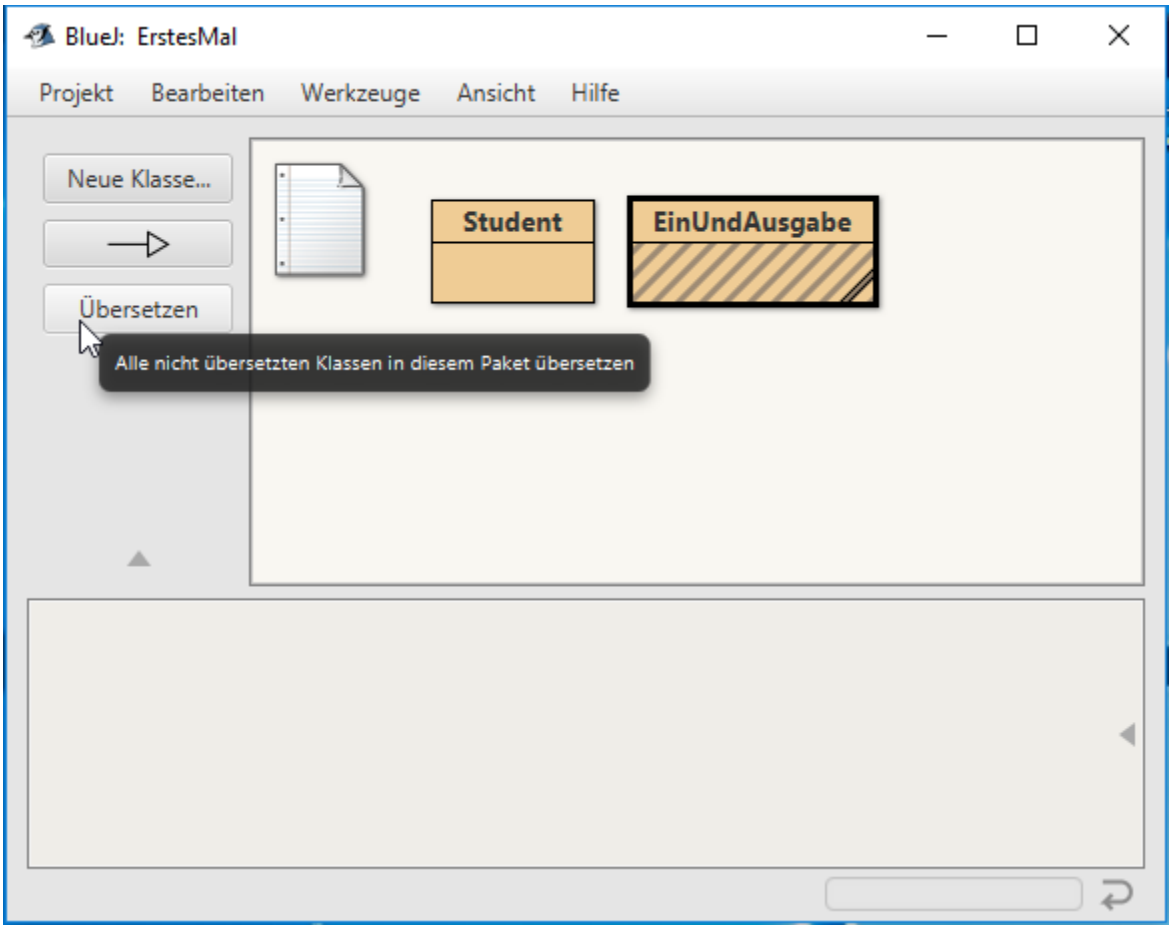

Zum Einlesen wird „Bearbeiten -> Klasse aus Datei hinzufügen…“ geklickt.



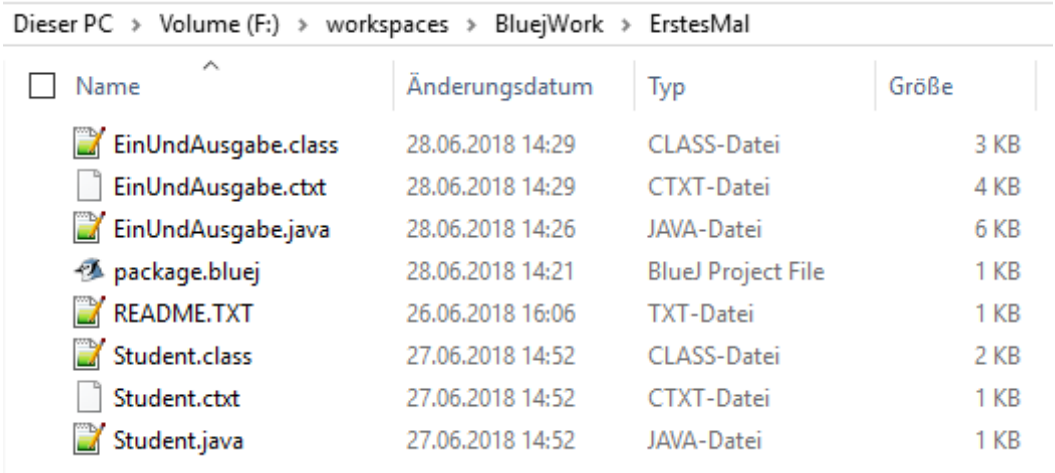

Nun wird im Datei-Browser die Klasse gesucht und dann „Öffnen“ geklickt.




Es ist erkennbar, dass die Klasse geladen, aber noch nicht kompiliert wurde, was z. B. mit einem Klick auf „Übersetzen“ möglich ist.

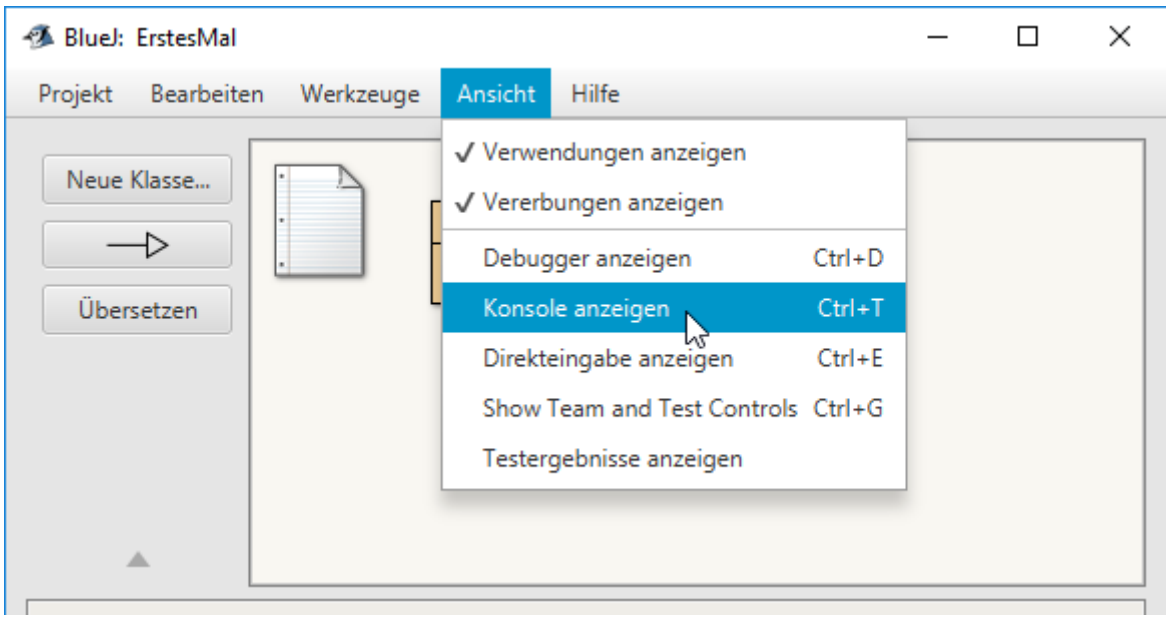



Die Datei EinUndAusgabe.java wurde als echte Kopie in das Projekt geladen. Damit würden Änderungen an dieser Klasse nicht zu Änderungen in der Ursprungsklasse führen. Dies wird deutlich, wenn der Projektordner geöffnet wird. Alternativ können neue Klassen bei noch nicht gestartetem BlueJ in den Projektordner kopiert werden. Dies Klassen werden beim nächsten Öffnen des Projekts gefunden.

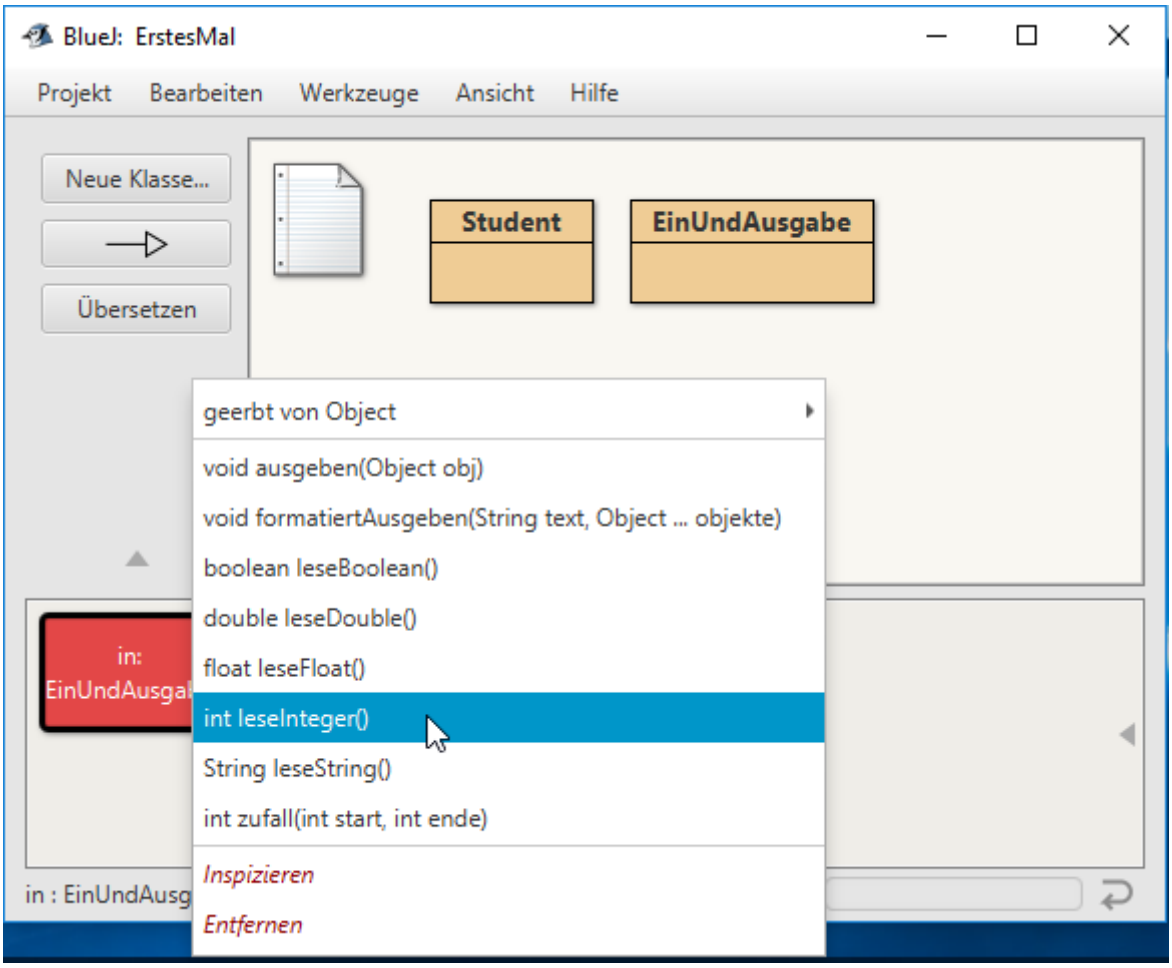


## 5.2 Nutzung der Eingabeklasse

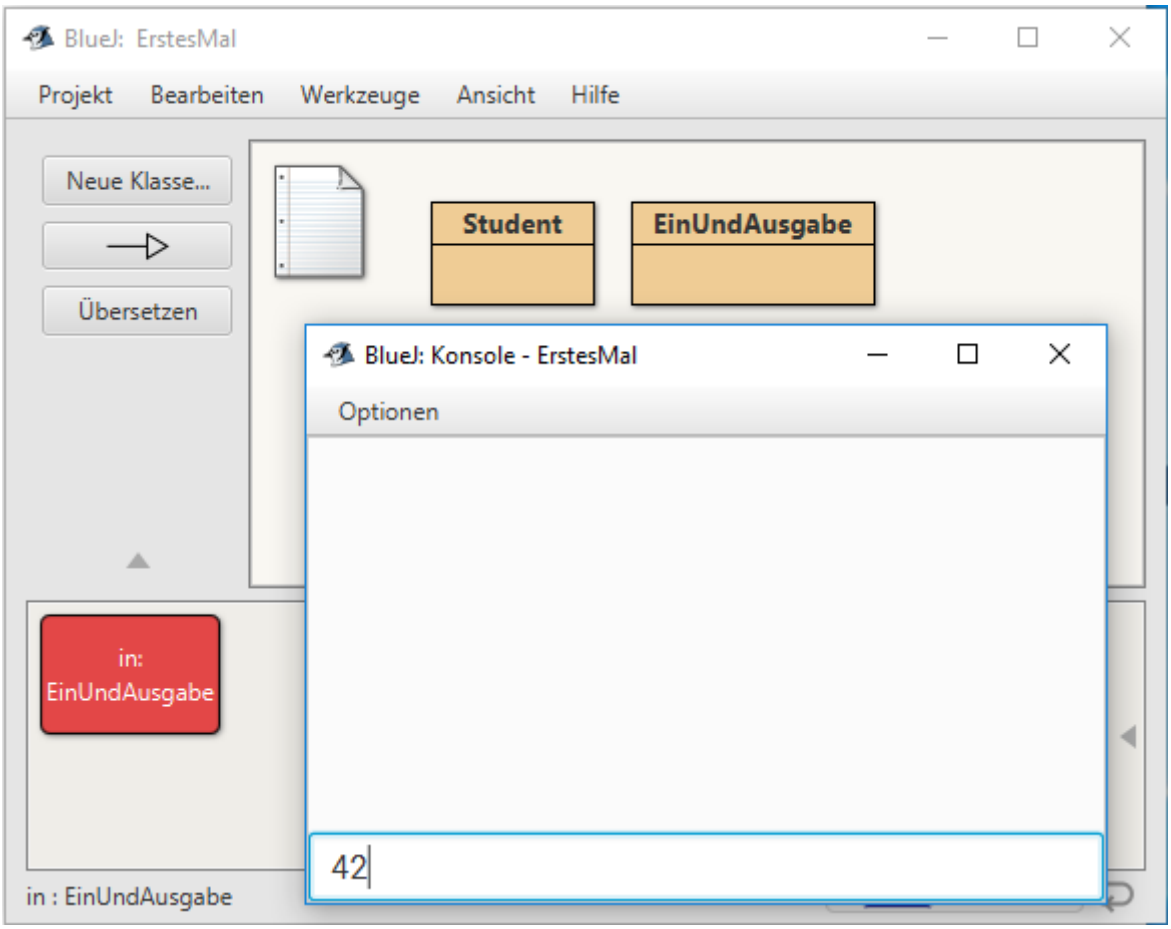
Zunächst ist es sinnvoll, die Ein- und Ausgabe-Konsole (auch Terminal genannt) über „View -> Show Terminal“ sichtbar zu machen. Hier werden Ausgaben von Programmen sichtbar, weiterhin kann der Nutzer hier Eingaben machen.



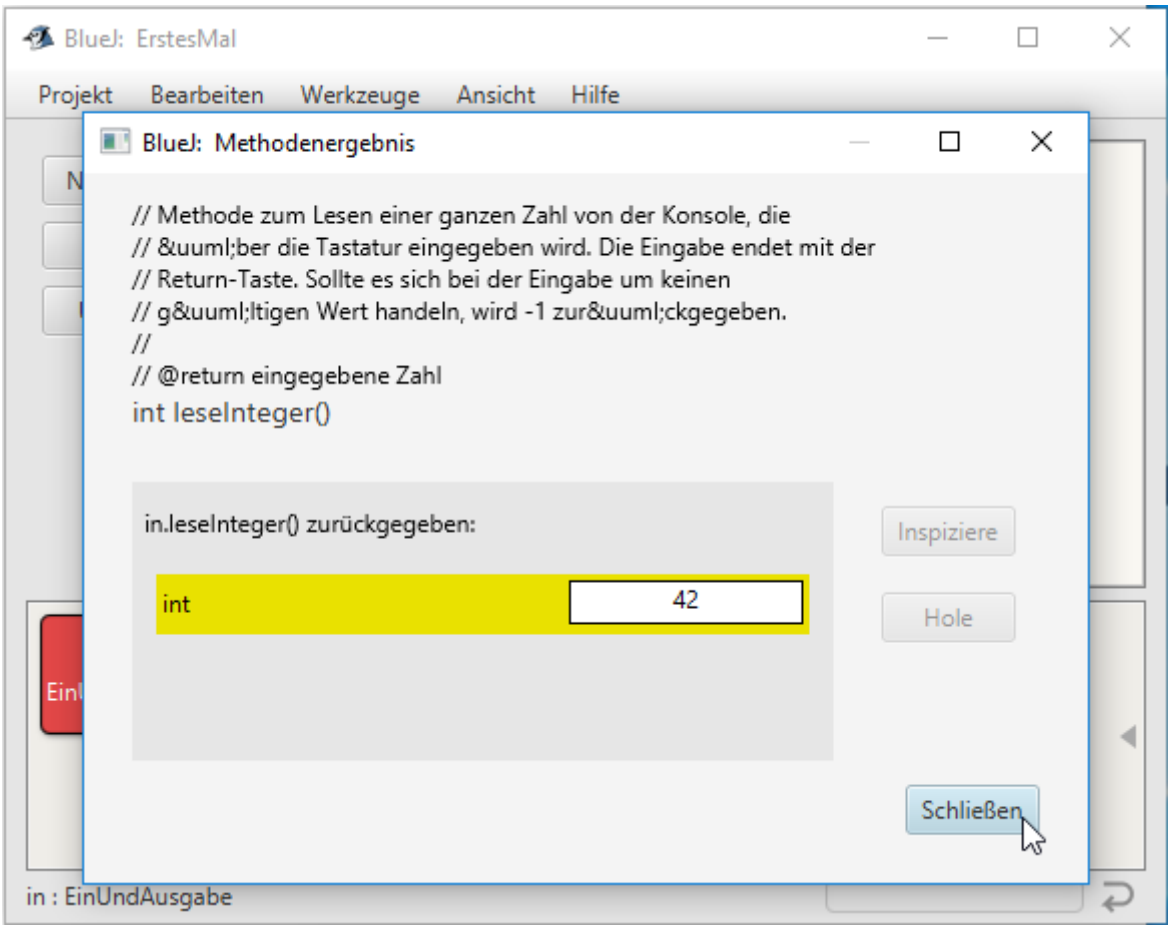

Soll eine Methode aufgerufen werden, wird ein Objekt erzeugt, ein Rechtsklick auf dem Objekt gemacht und die gewünschte Methode ausgewählt.



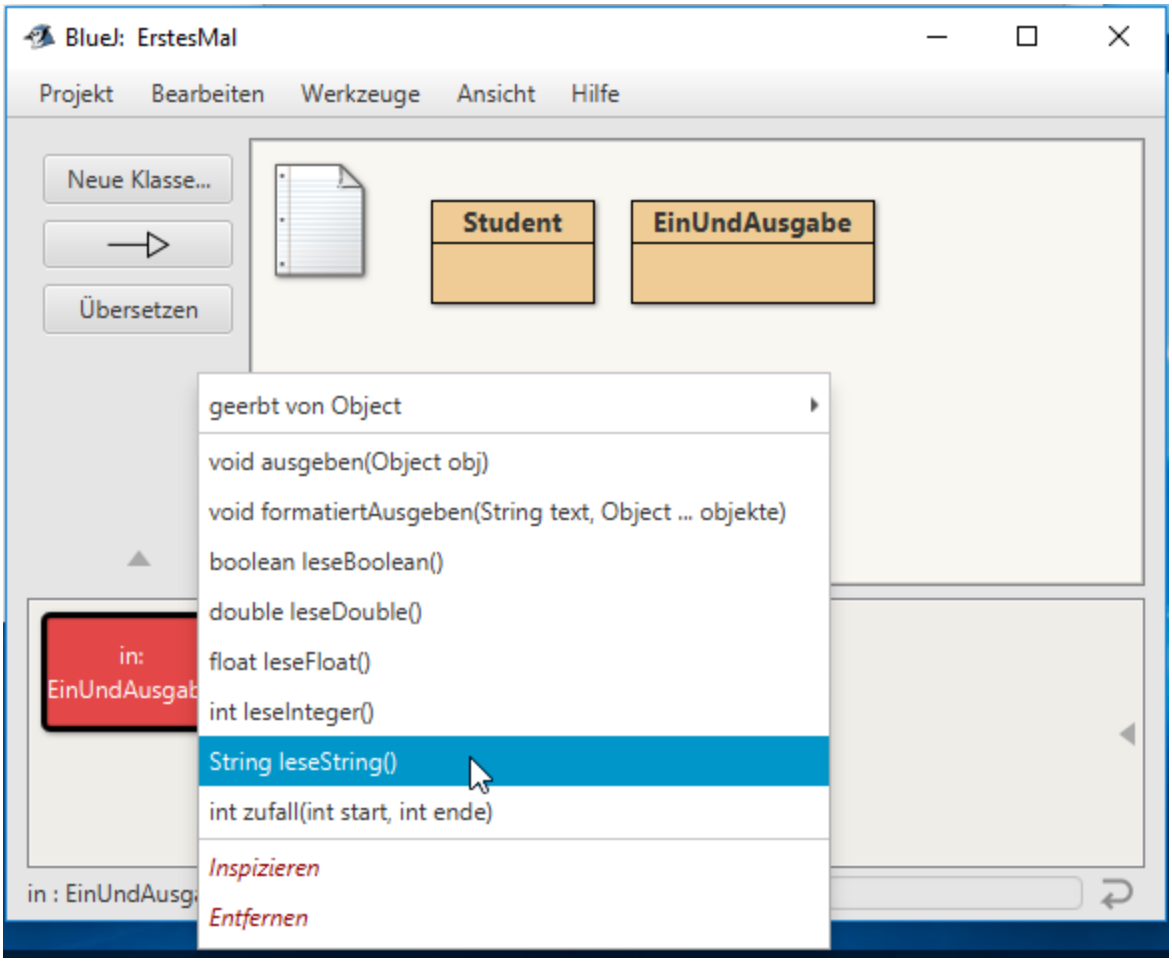

Da in diesem Fall die Methode eine Konsoleneingabe erwartet, kann jetzt über die Konsole ein Wert eingetippt werden, um dann die Eingabe mit der „Return“-Taste abzuschließen.



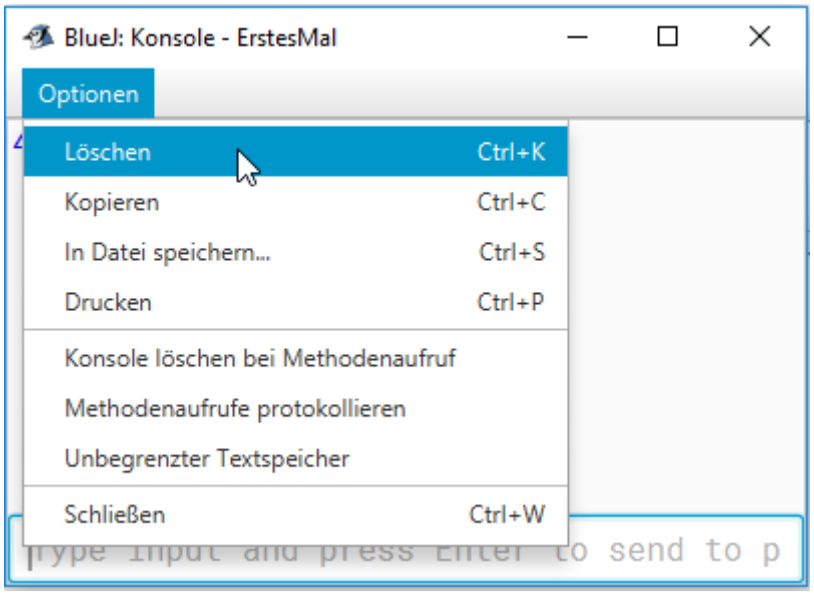

Nach der Eingabe wird die Rückgabe der Methode als untersuchbarer Wert ausgegeben. Im aktuellen Fall ist es ein einfacher Integer-Wert. Der oben mit ausgegebene Kommentar kann ignoriert werden.



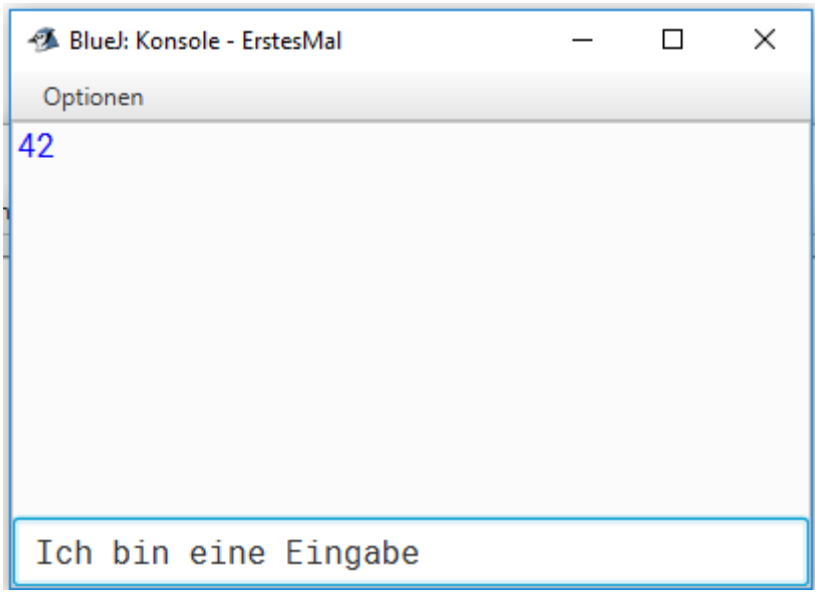

Nun wird eine Methode aufgerufen, die ein Objekt, hier einen String zurückliefert.



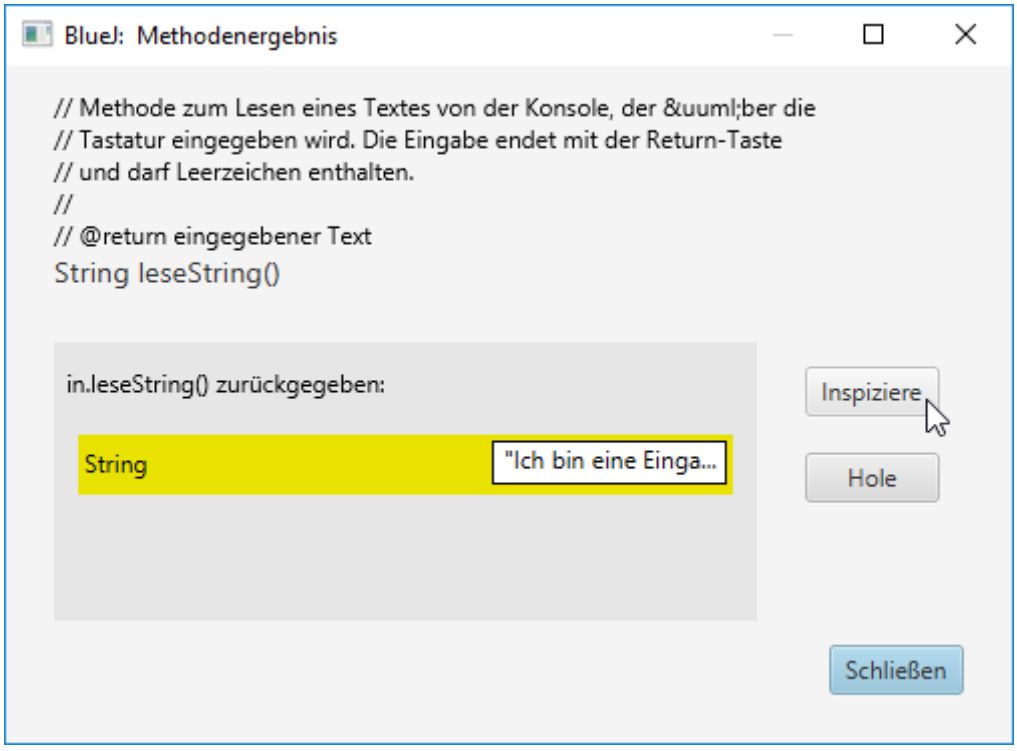

Die Eingabe erfolgt wieder über die Konsole. Sollten alte Eingaben stören, können diese vorher über „Optionen -> Löschen“ gelöscht werden. Soll bei jedem Methodenaufruf das Ausgabefenster gelöscht werden, ist „Clear screen at method call“ anzuklicken. Werden sehr lange Ausgaben erwartet, ist „Unilimited Buffering“ auszuwählen.



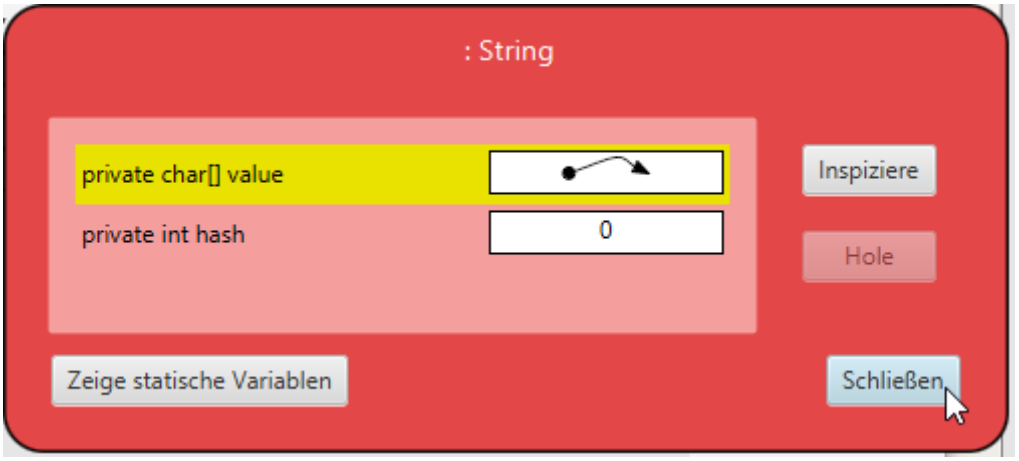
Ohne Löschen wird jetzt ein String eingegeben, der auch Leerzeichen enthalten kann.



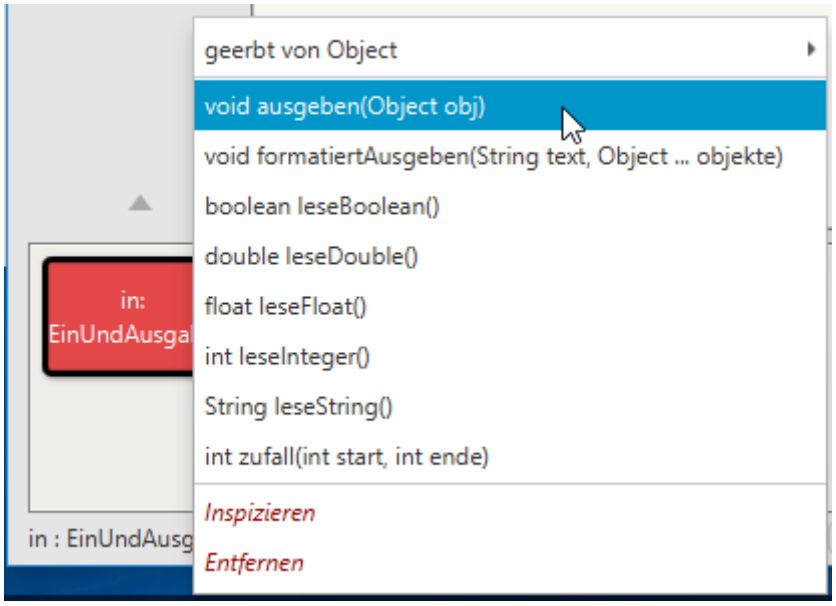

Das Ergebnis-Objekt, kann mit „Inspiziere“ untersucht werden.



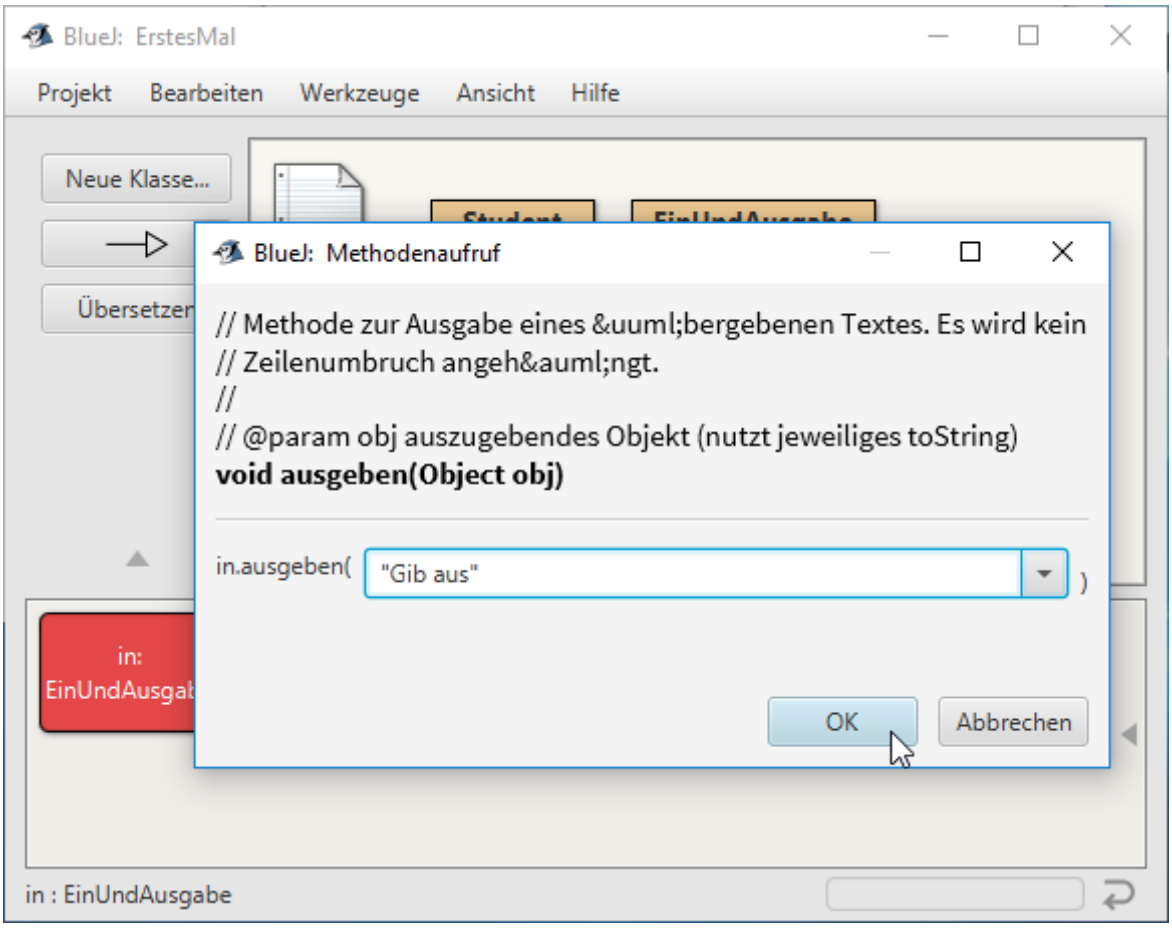

Es ist wieder möglich die innere Objektstruktur zu betrachten, deren Details hier aber nicht interessieren.



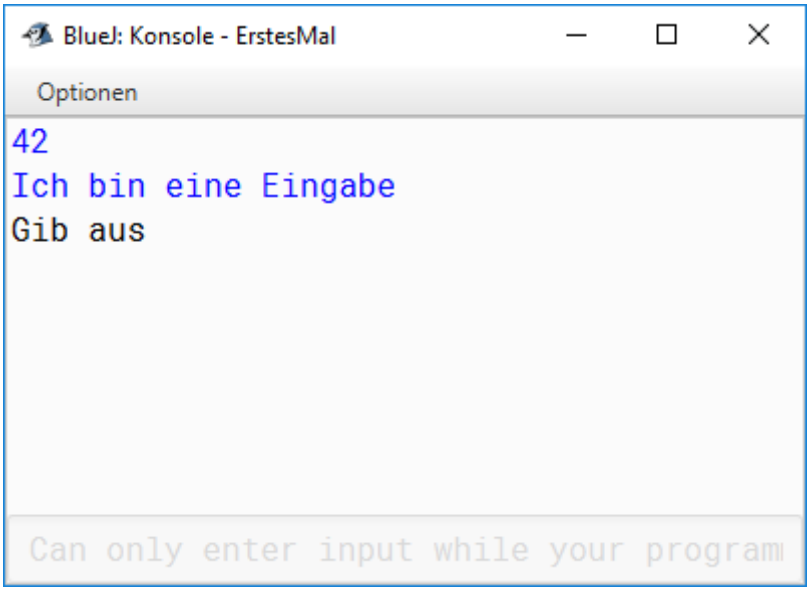

Ähnlich zur Eingabe funktioniert auch die Ausgabe, hier wird die Methode ausgeben() aufgerufen.




Als Parameter wird das auszugebende Objekt angegeben, hier ein String-Objekt.



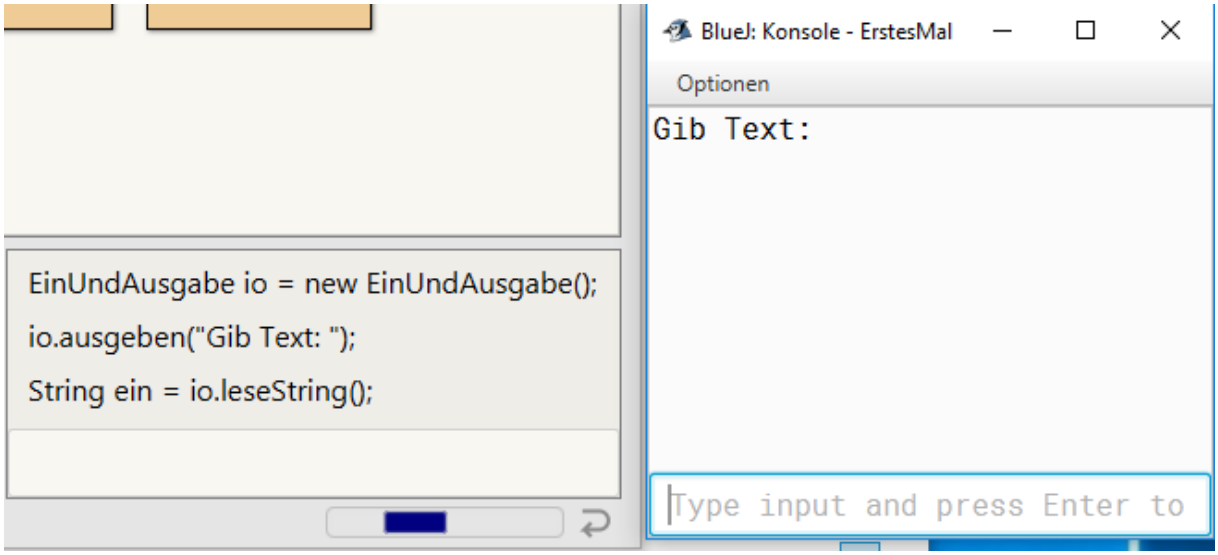

Die Ausgabe erfolgt in der Konsole, dabei können Eingaben in blau und Ausgaben in schwarz unterschieden werden.



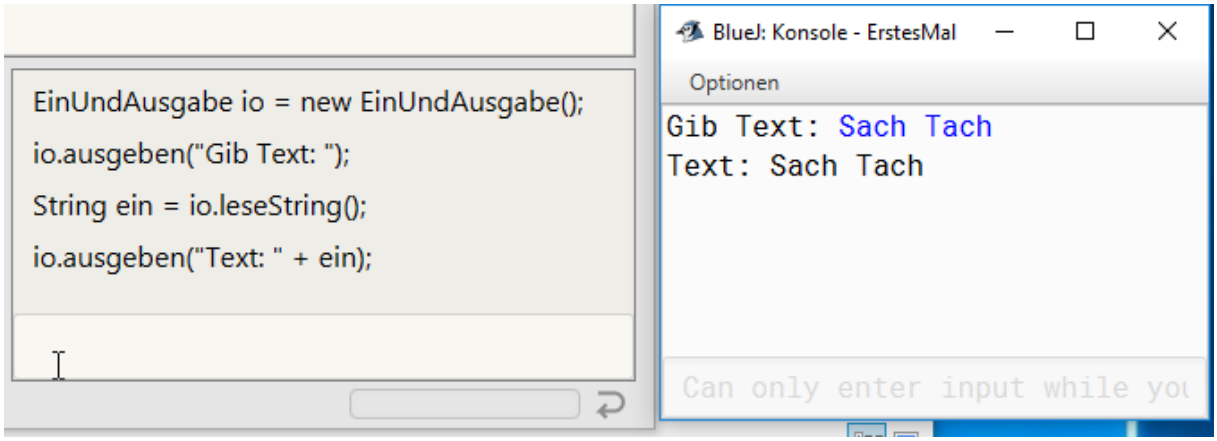

Ein kleines Beispielprogramm im Code Pad kann wie folgt aussehen.

```
EinUndAusgabe io = new EinUndAusgabe();
io.ausgeben("Gib Text: ");
String ein = io.leseString();
io.ausgeben("Text: " + ein);
```

Zunächst wird ein Text als Eingabeaufforderung ausgegeben und dann soll ein String-Objekt eingelesen werden, dessen Referenz mit der Variable ein verknüpft wird.




Nach Eingabe des Textes kann das Programm mit einer Ausgabe des vorher eingegebenen Textes verlängert werden.

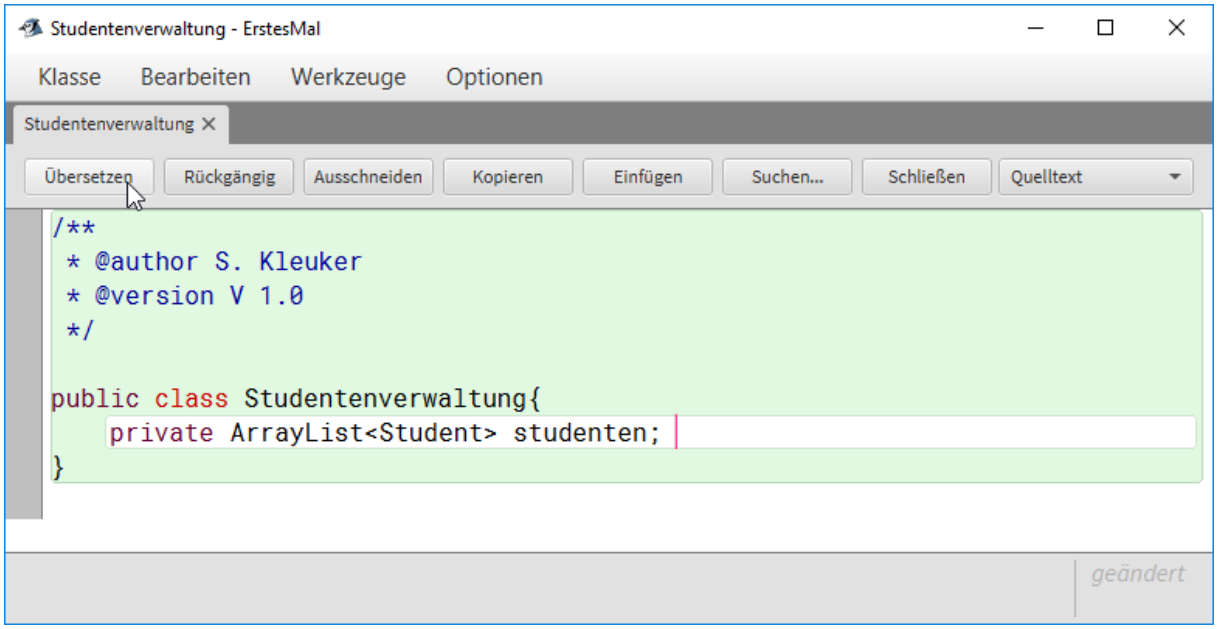



## *5.3 Ein einfaches Programm mit Nutzung einer Sammlung*

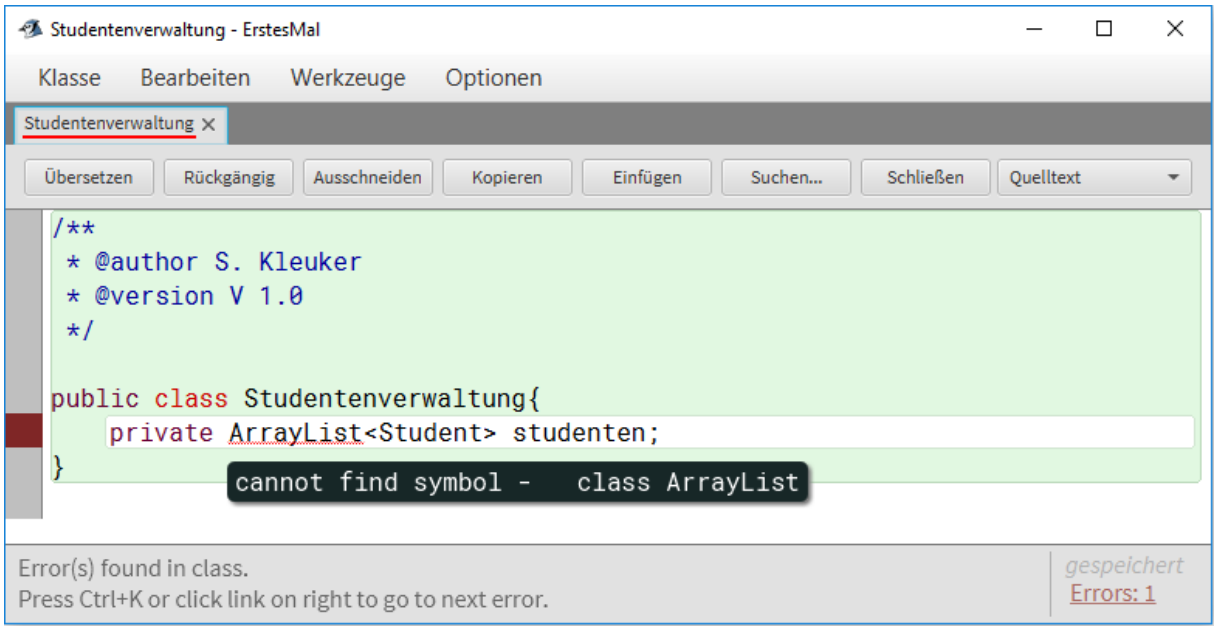
Der folgende Abschnitt beschreibt die Erstellung eines Programms zur Verwaltung neuer Studenten-Objekte. Dabei werden die bisher beschriebenen Klassen Student und Eingabe genutzt. Die Grundidee ist, dass es eine weitere Verwaltungsklasse gibt, die alle Studenten-Objekte kennt und einen Dialog anbietet, mit dem Studierenden-Objekte bearbeitet werden können. Es ist zu beachten, dass die Programmierung hier auf relativem Anfänger-Niveau

passiert, da die Eingabe nicht von der Verwaltung der Objekte getrennt wird und statt einer wahrscheinlich sinnvolleren Nutzung des Sammlungstyps Map auf eine eventuell einfacher verständliche List zurückgegriffen wird.

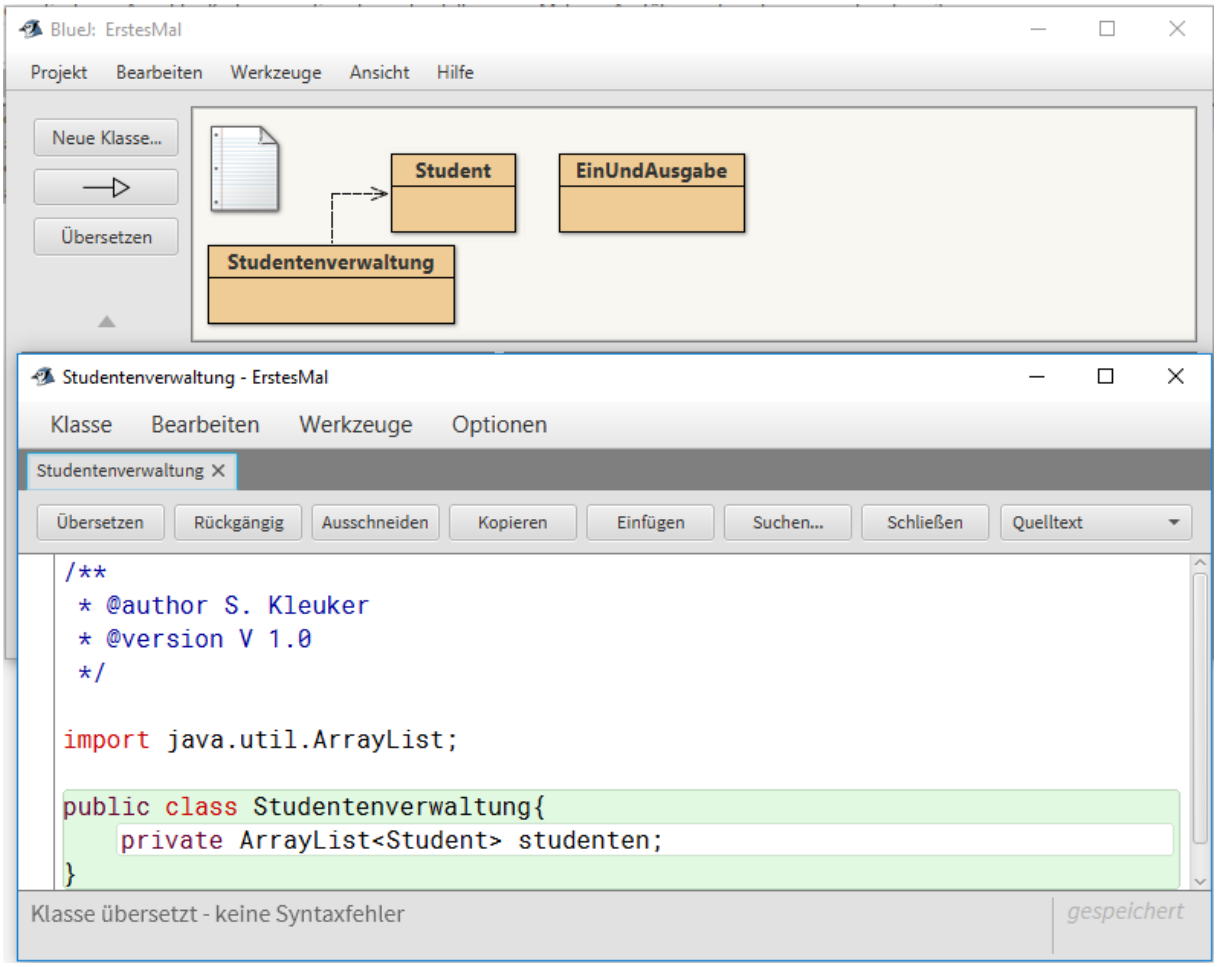
Zunächst wird eine neue Klasse Studentenverwaltung angelegt und der Programmeditor mit einem Doppelklick geöffnet, der Beispieltext gelöscht, einige Zeilen programmiert und der „Übersetzen“-Knopf gedrückt.



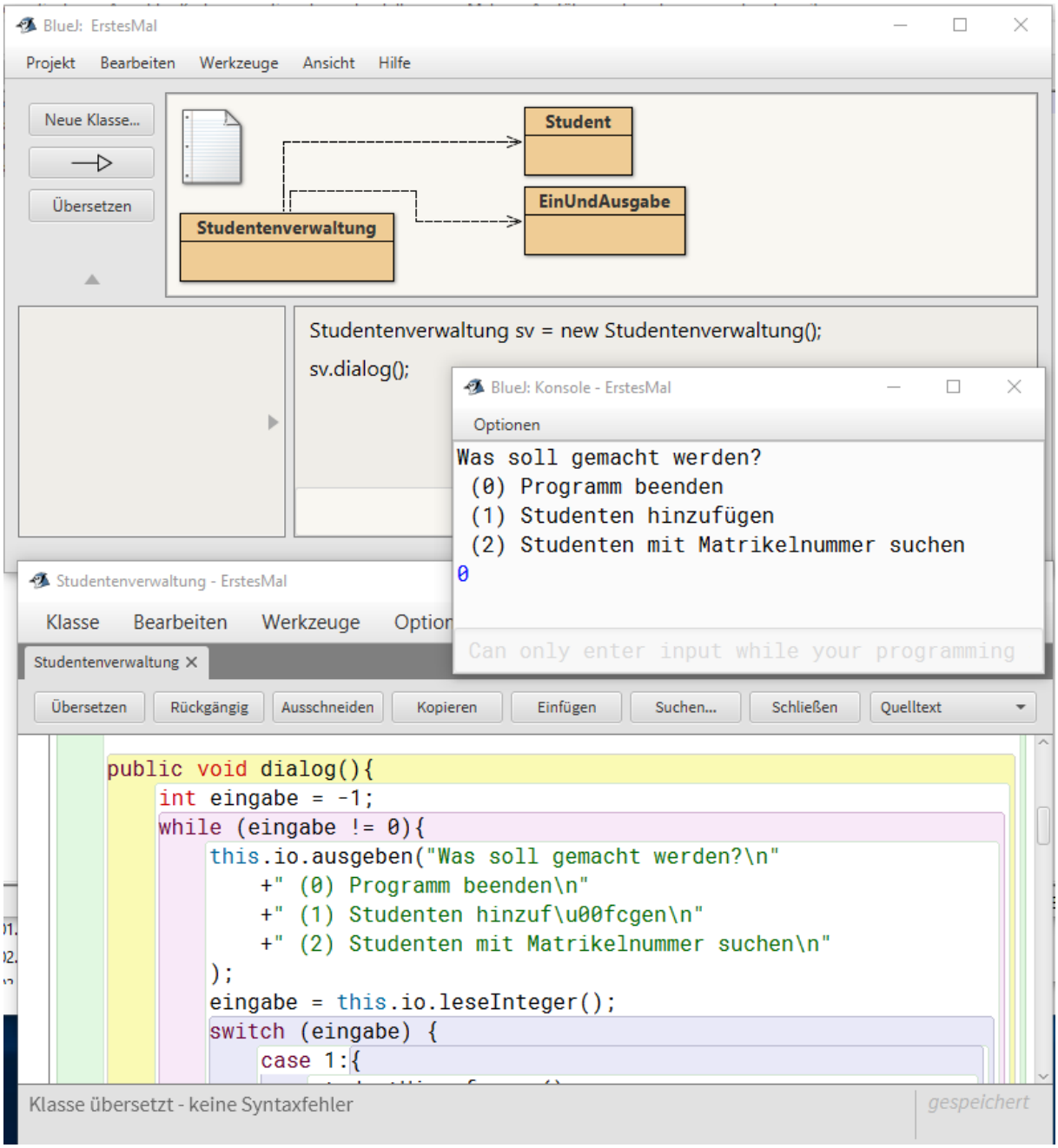

Es gibt eine Fehlermeldung, da die Klasse ArrayList noch nicht bekannt ist. Leider bietet der Editor keine Unterstützung, möglichst einfach bekannte Klassen aus der Klassenbibliothek einzubinden.



Die benötigte import-Zeile wird eingetippt und nach erfolgreicher Kompilierung fällt auf, dass im Klasseneditor eine gestrichelte Linie ergänzt wird. Diese Linie gibt an, dass die Klasse Studentenverwaltung die Klasse Student nutzt, also von dieser Klasse abhängig ist. Konkreter gilt, dass ein Objekt der Klasse Studentenverwaltung beliebig viele Objekte der Klasse Student nutzt. Das Layout der Pfeile kann leider nicht beeinflusst werden. Weiterhin hat der Klasseneditor neben der später beschriebenen Vererbung keine weiteren Möglichkeiten, Klassendiagramme als UML-Klassendiagramme darzustellen, die die in der Software-Entwicklung standardmäßig eingesetzte Notation ist.



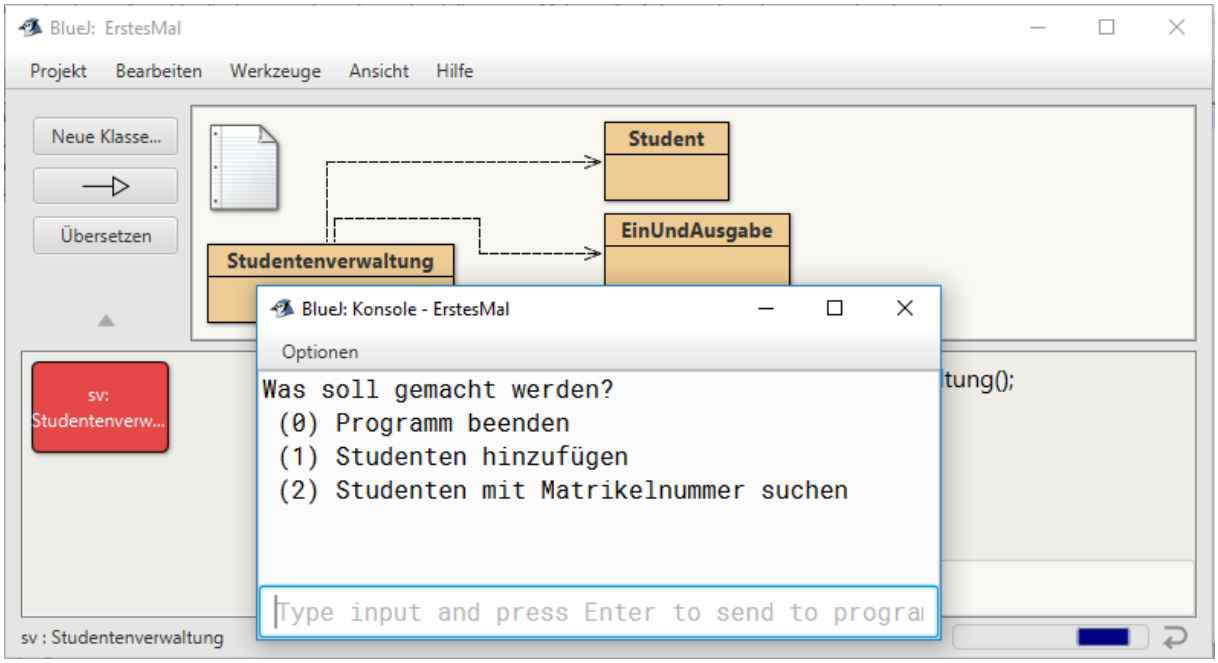

Das Programm wird jetzt schrittweise, durch mehrfaches Kompilieren und Weiterschreiben entwickelt. Im folgenden Beispiel wurde der Nutzungsdialog eingegeben und es ist manuell prüfbar, dass das Programm mit der Eingabe der Zahl Null terminiert.



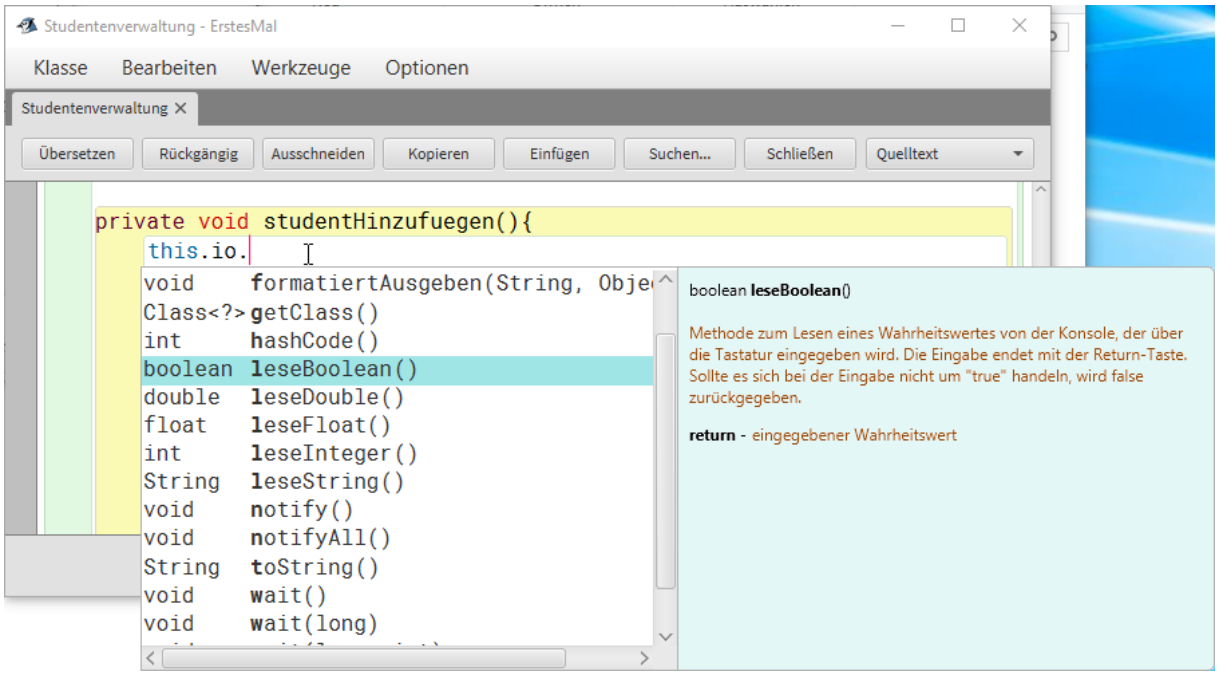

Als Hinweis am Rande sei vermerkt, dass in Java in Texten zwar deutsche Umlaute verwendbar sind, dieses aber vermieden werden sollte. Das Problem kann auch direkt auftreten, wenn Dateien zwischen Windows- und Unix-Systemen (Solaris, Linux, Mac OS) oder BlueJ und Eclipse ausgetauscht werden. Folgende Unicode-Zeichen sind zu verwenden (http://javawiki.sowas.com/doku.php?id=java:unicode).

| Zeichen | Unicode |
|---|---|
| Ä, ä | \u00c4, \u00e4 |
| Ö, ö | \u00d6, \u00f6 |
| Ü, ü | \u00dc, \u00fc |
| ß | \u00df |

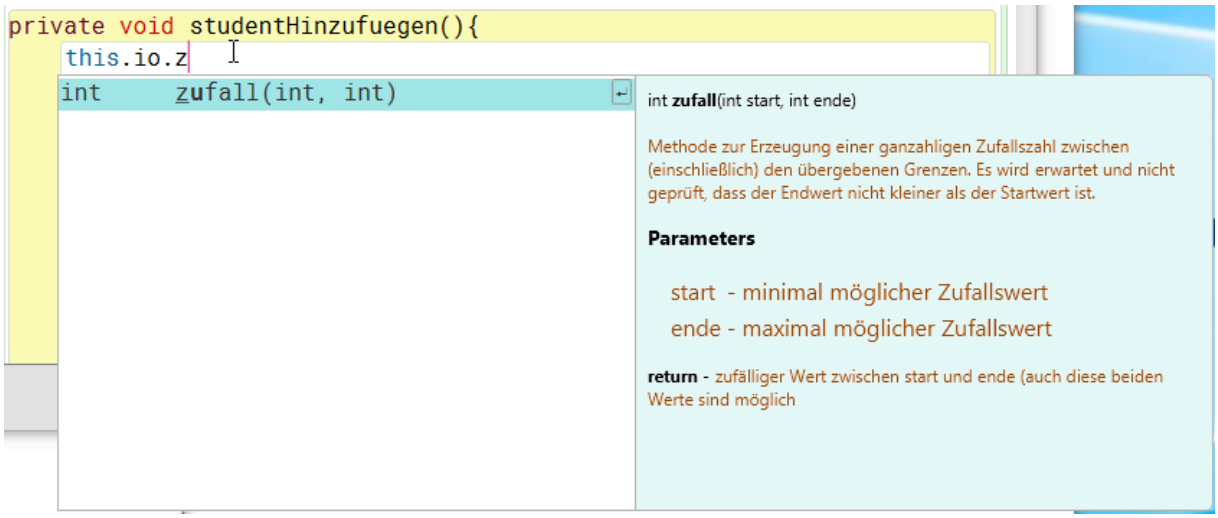
Zum Ausprobieren kann natürlich alternativ ein Objekt der Klasse Studentenverwaltung angelegt und im Code Pad die Methode dialog() aufgerufen werden. Der laufende blaue Balken am unteren Rand des Bildes macht deutlich, dass auf eine Eingabe gewartet wird



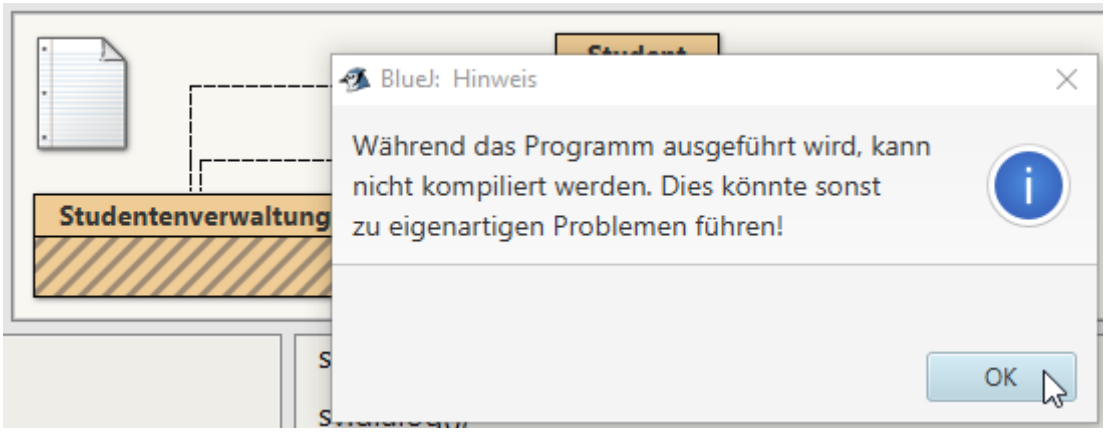

Wenn Klassen dem Editor bekannt sind, unterstützt er die sogenannte Code Completion, bei der er Vorschläge generiert, welche Methode genutzt werden soll. Nach dem letzten Programmstück ist die Verknüpfung zur Klasse EinUndAusgabe bekannt, von dessen Typ es eine Objektvariable io gibt, so dass nach der Eingabe von „io.“ und dem Drücken von „Strg“+“Leertaste“ ein Angebot an nutzbaren Methoden angezeigt wird, die dann mit einem Mausklick oder durch die Tasten Pfeil-hoch und Pfeil-runter ausgewählt werden können. Dieser Ansatz kann die Anzahl von Tippfehlern deutlich minimieren, weiterhin müssen so bei Klassen wie ArrayList nicht alle Methoden im Detail gemerkt oder permanent nachschlagen werden.



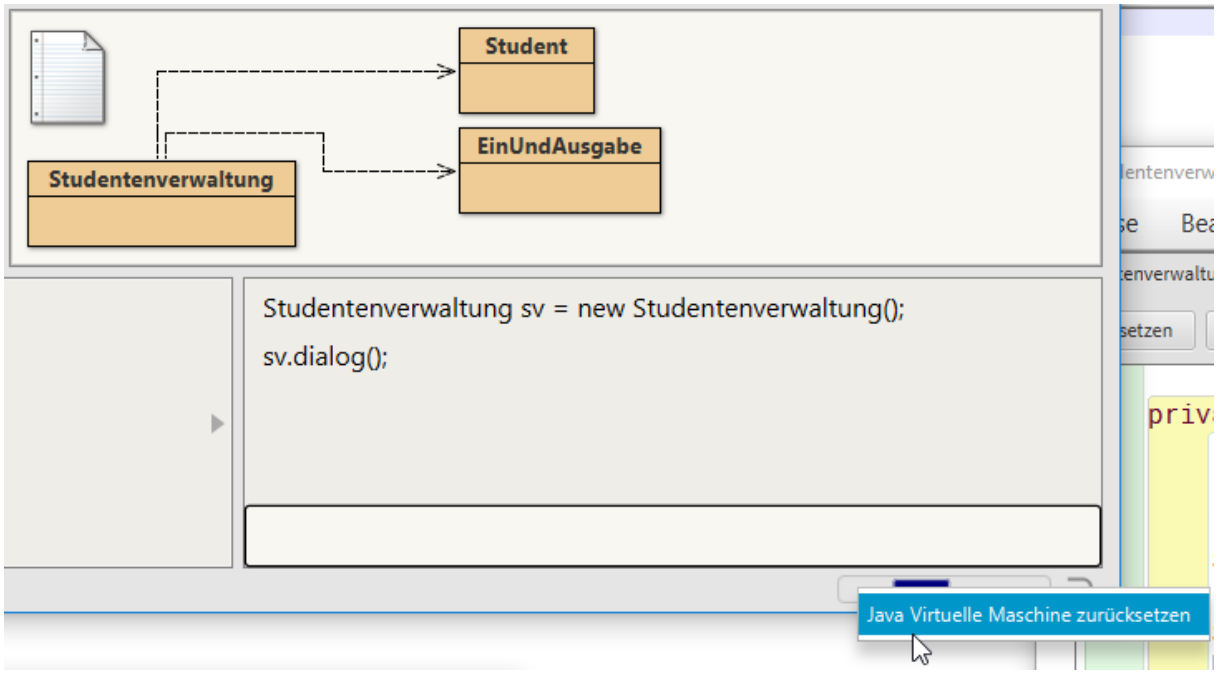

Durch das Eintippen einzelner Buchstaben als Anfang des Methodennamens wird die Auswahl dann eingeschränkt. Falls zur Methode eine Dokumentation in JavaDoc vorliegt, wird diese rechts mit angezeigt.




Neben der möglichen großen Anzahl an Fenstern in BlueJ, bei denen der Überblick behalten werden muss, ist zu beachten, dass eine Klasse nicht übersetzt werden kann, wenn das Programm gerade läuft.




Das laufende Programm ist an dem animierten Querbalken erkennbar. Mit einem Rechtsklick auf diesen Balken und der einzig möglichen Auswahl „Java Virtuelle Maschine zurücksetzen“, kann die Java Virtual Machine, also das für die Ausführung von Java-Programmen zuständige Programm stoppen und neu starten. Dies ist ein recht drastischer Schritt und sollte nur genutzt werden, wenn keine anderen Möglichkeiten bestehen



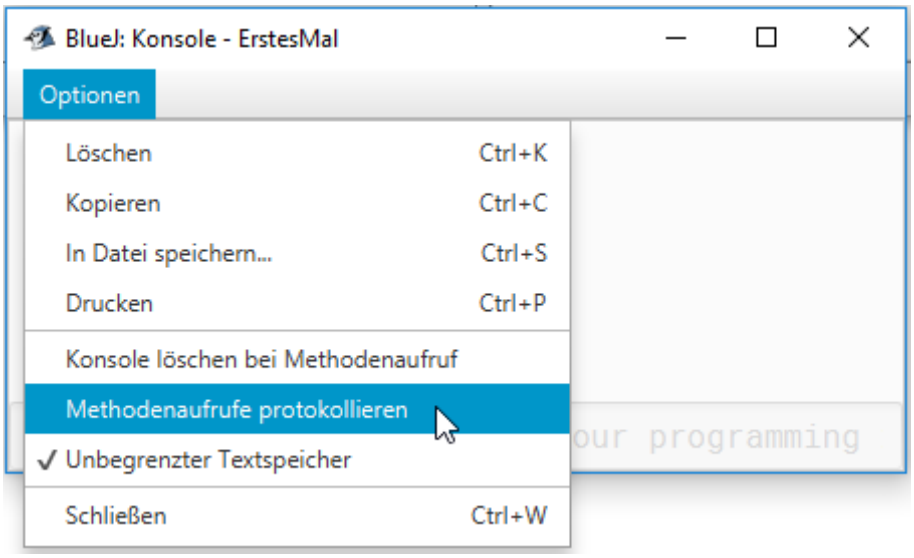

Das resultierende Programm sieht dann wie folgt aus.

```
/**
 * @author S. Kleuker
 * @version V 1.0
 */

import java.util.ArrayList;

public class Studentenverwaltung{
    private ArrayList<Student> studenten;
    private EinUndAusgabe io;
```

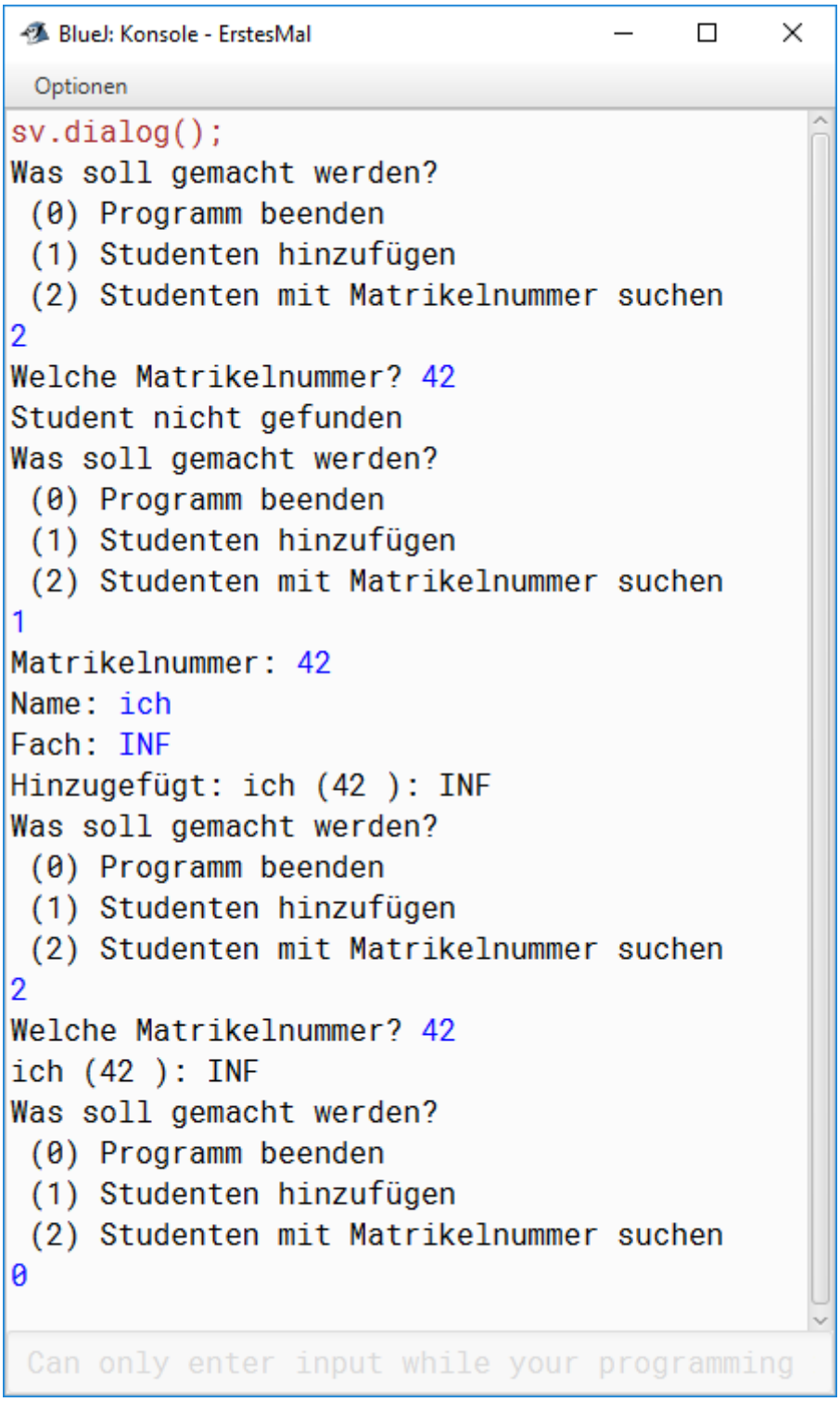
```
public Studentenverwaltung(){
    this.studenten = new ArrayList<Student>();
    this.io = new EinUndAusgabe();
}

public void dialog(){
    int eingabe = -1;
    while (eingabe != 0){
        this.io.ausgeben("Was soll gemacht werden?\n"
            +" (0) Programm beenden\n"
            +" (1) Studenten hinzuf\u00fcgen\n"
            +" (2) Studenten mit Matrikelnummer suchen\n"
        );
        eingabe = this.io.leseInteger();
        switch (eingabe) {
            case 1:{
                studentHinzufuegen();
                break;
            }
            case 2:{
                studentSuchen();
                break;
            }
        }
    }
}

private void studentHinzufuegen(){
    this.io.ausgeben("Matrikelnummer: ");
    int mat = this.io.leseInteger();
    this.io.ausgeben("Name: ");
    String name = this.io.leseString();
    this.io.ausgeben("Fach: ");
    String fach = this.io.leseString();
    if(matrikelnummerExistiert(mat)){
        this.io.ausgeben("Keine doppelten Matrikelnummern\n");
    } else {
        Student neu = new Student(mat, name, fach);
        studentHinzufuegen(neu);
        this.io.ausgeben("Hinzugef\u00fcgt: "+neu+"\n");
    }
}

public void studentHinzufuegen(Student s){
    this.studenten.add(s);
}

public Boolean matrikelnummerExistiert(int mat){
    Boolean ergebnis = false;
    Student tmp = studentSuchen(mat);
    if(tmp != null){
        ergebnis = true;
    }
    return ergebnis;
}

public void studentSuchen(){
    this.io.ausgeben("Welche Matrikelnummer? ");
    int mat = this.io.leseInteger();
    Student std = studentSuchen(mat);
```

```
            if (std == null){
                this.io.ausgeben("Student nicht gefunden\n");
            } else {
                this.io.ausgeben(std+"\n");
            }
        }

        public Student studentSuchen(int matrikelnummer){
            Student ergebnis = null;
            for(Student s:studenten){
                if (s.getMatrikelnummer() == matrikelnummer){
                    ergebnis = s;
                }
            }
            return ergebnis;
        }
    }
```

Das Programm kann dann interaktiv mit Nutzereingaben getestet werden. Optional kann bei der Konsole unter „Optionen -> Methodenaufrufe protokollieren“ einstellen, dass die durchgeführten Methodenaufrufe, dann in einem weiteren Terminal Window, mit angezeigt werden. Ob dies hilfreich ist, da z. B. keine internen Methodenaufrufe sichtbar werden, muss jeder für sich selbst entscheiden.



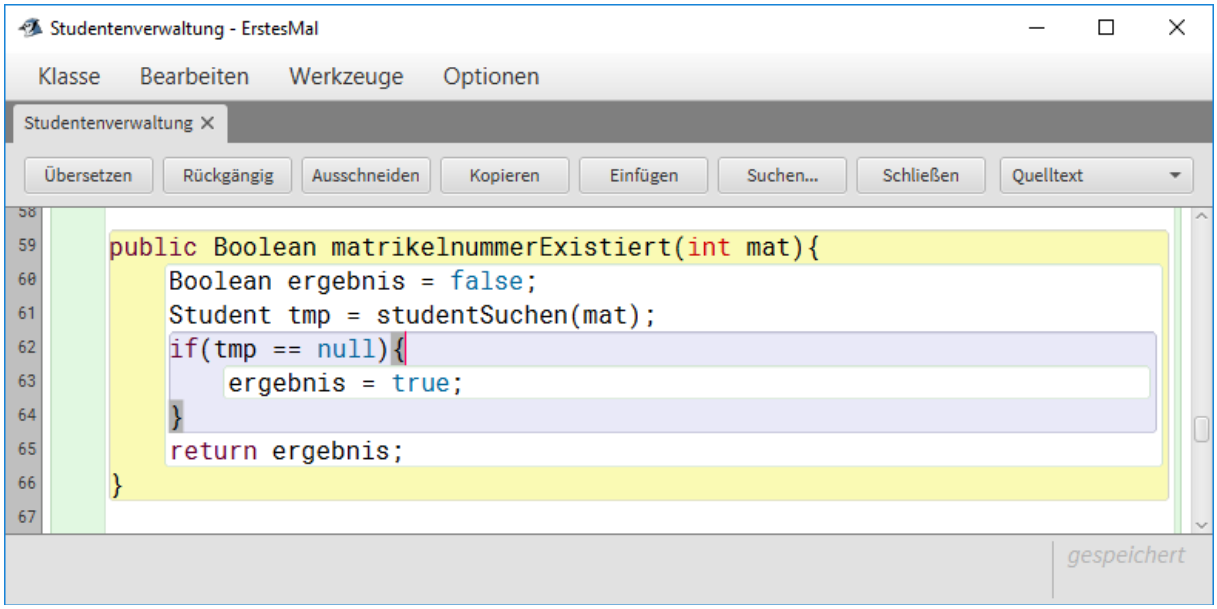

Ein Beispielablauf mit den zugehörigen Methodenaufrufen, nachdem ein Objekt sv der Klasse Studentenverwaltung erstellt wurde, kann wie folgt aussehen.


BlueJ: Konsole - ErstesMal
Optionen
sv.dialog();
Was soll gemacht werden?
(0) Programm beenden
(1) Studenten hinzufügen
(2) Studenten mit Matrikelnummer suchen
2
Welche Matrikelnummer? 42
Student nicht gefunden
Was soll gemacht werden?
(0) Programm beenden
(1) Studenten hinzufügen
(2) Studenten mit Matrikelnummer suchen
1
Matrikelnummer: 42
Name: ich
Fach: INF
Hinzugefügt: ich (42 ): INF
Was soll gemacht werden?
(0) Programm beenden
(1) Studenten hinzufügen
(2) Studenten mit Matrikelnummer suchen
2
Welche Matrikelnummer? 42
ich (42 ): INF
Was soll gemacht werden?
(0) Programm beenden
(1) Studenten hinzufügen
(2) Studenten mit Matrikelnummer suchen
0
Can only enter input while your programming

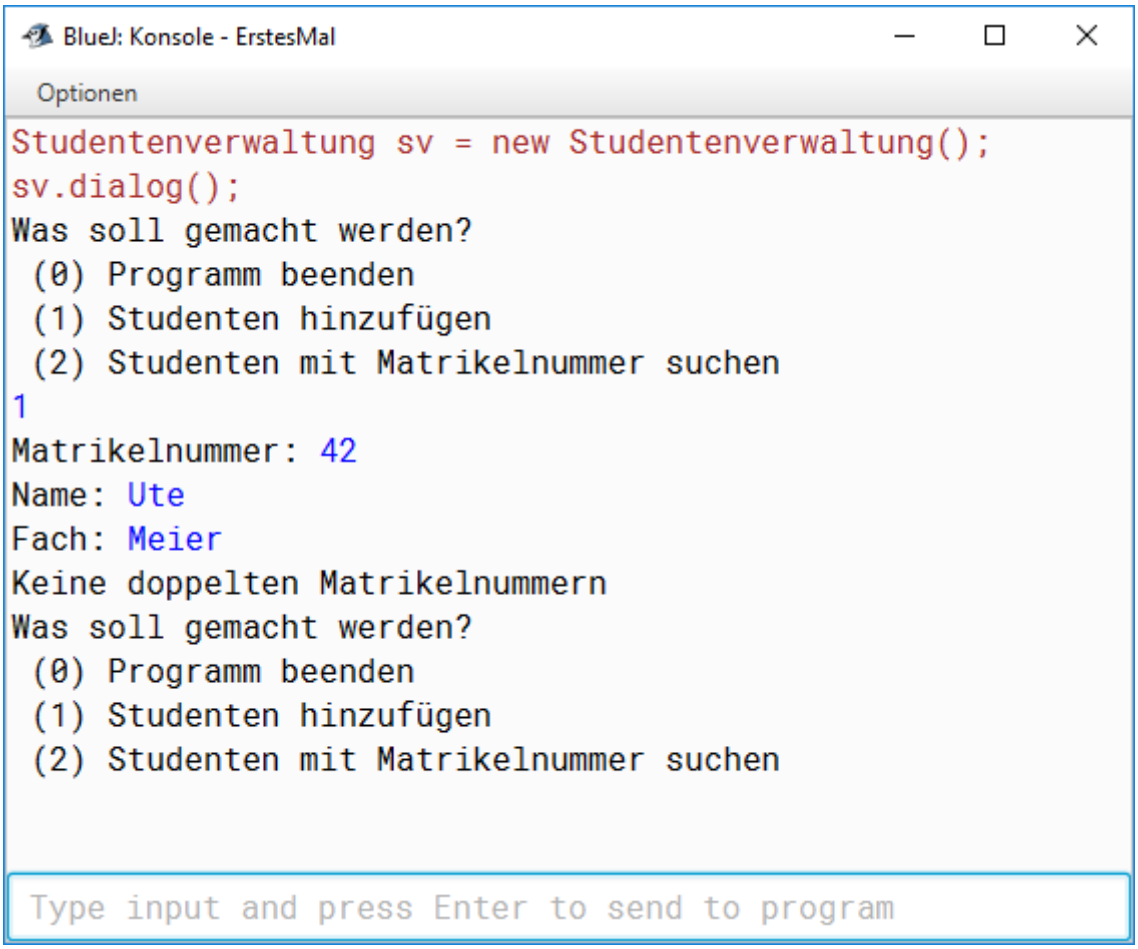
# 6 Nutzung des Debuggers

Oftmals stellt ein Entwickler in der Programmausführung fest, dass sich das Programm nicht wie gewünscht verhält. Zunächst sollte dann versucht werden durch kritisches Lesen des Programmcodes auf den Fehler zu stoßen. Wird der Fehler nicht gefunden, ist es unbedingt zu vermeiden, durch unsystematische Versuche doch noch zum laufenden Programm zu kommen. Dies ist gerade bei Anfängern ein gern gemachter Fehler und führt dann zu meist nur vermeintlich korrekten Programmen, deren Ablauf der Entwickler selbst nicht beschreiben kann. Eine Lösung ist das Einfügen von zusätzlichen Ausgabezeilen, die die momentanen Werte der aktuellen Variablen anzeigen, dieser Ansatz kann später durch Logging-Frameworks noch weiter systematisiert werden. Noch systematischer ist die Nutzung eines Debuggers, mit dem genau der Ablauf eines Programms mit allen Werten der Variablen verfolgt werden kann. Typischerweise wird dazu eine Zeile so markiert, dass die virtuelle Maschine bei der Ausführung des Java-Programms „weiß“, dass jetzt der Debugger gestartet werden soll.

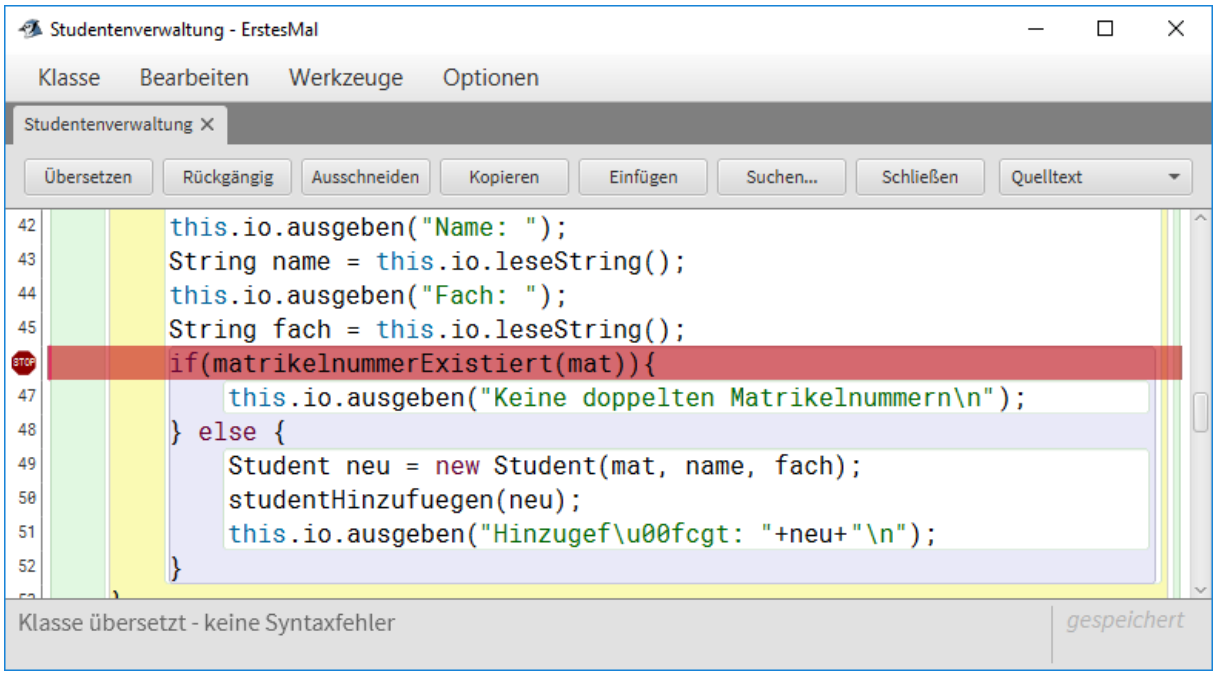
Als Beispiel dient das Programm aus dem Kapitel 5.3 bei dem in der Zeile 63 statt tmp!=null nun tmp == null steht.




Bei der Programmausführung scheitert bereits das erste Einfügen eines Studenten und es wird behauptet, dass es schon einen Studenten mit diese Matrikelnummer gibt..



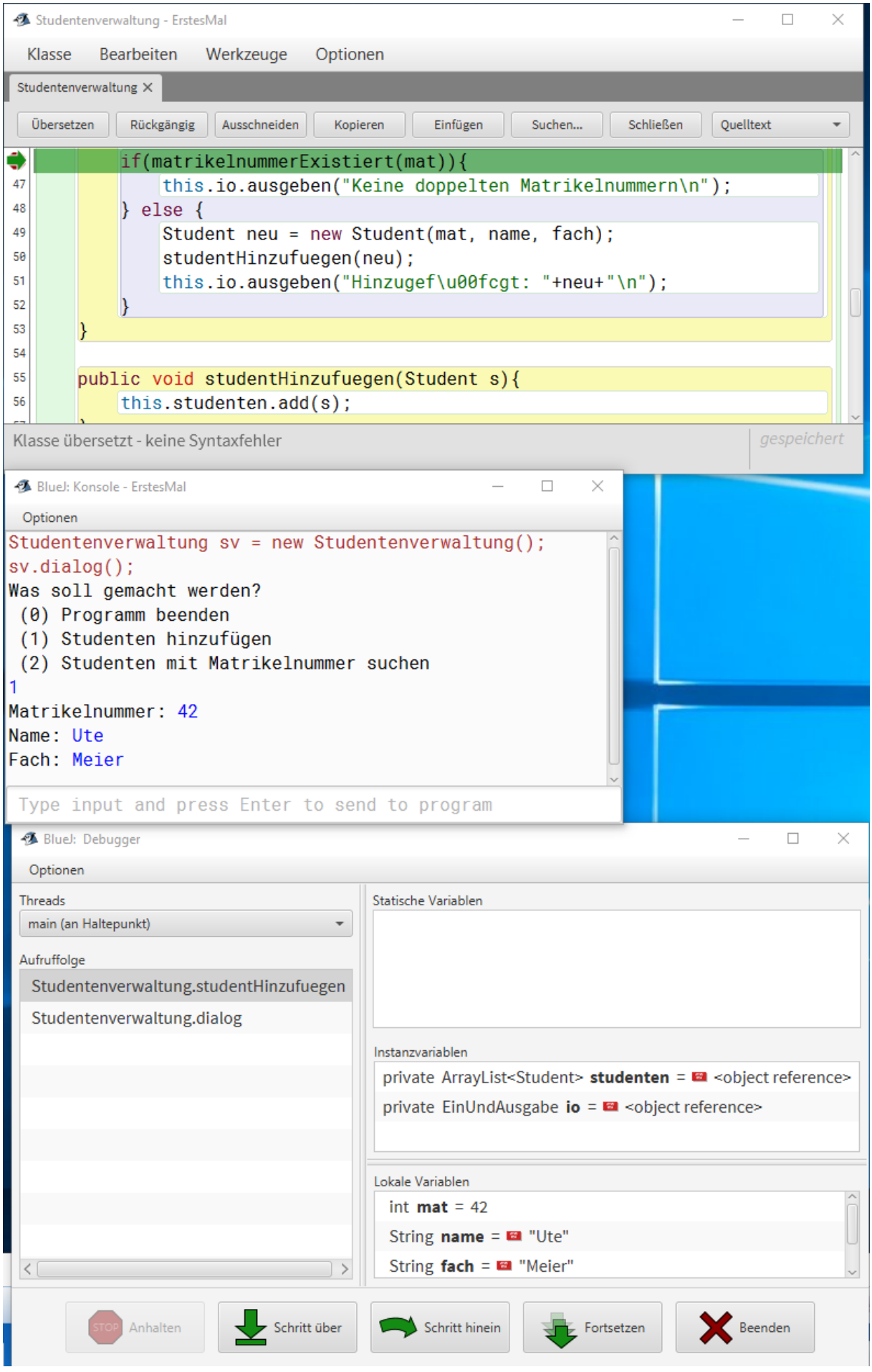

Nach erfolgloser Fehlersuche wird festgestellt, dass die Eingabe wohl noch funktioniert, danach aber irgendwann der Fehler auftritt. Aus diesem Grund wird unmittelbar nach der Eingabe ein sogenannter Breakpoint (Unterbrechungspunkt) gesetzt, der bei der nächsten Ausführung dazu führt, dass der Debugger aufgerufen wird. Der Breakpoint wird durch einen einfachen Klick auf der Zeilenummer (oder dem leeren Feld neben der Zeile, falls keine Zeilennummern angezeigt werden) gesetzt, was nur bei einem kompilierten Programm möglich ist. Grundsätzlich kann es durchaus mehrere Breakpoints geben, dass Programm unterbricht dann, wenn der erste Breakpoint erreicht wird. Später wird deutlich, dass der Debugger die Möglichkeit hat, das Progtramm einfach weiterlaufen zu lassen, so dass es beim nächsten erreichten Breakpoint wieder anhält.




Nun wird ein Objekt der Klasse Studentenverwaltung erzeugt und der Dialog aufgerufen. Unmittelbar nach der Eingabe des neuen Studenten öffnet sich der Debugger und im Programmcode ist die Zeile, die gerade ausgeführt werden soll, grün markiert.

# Nutzungshinweise für BlueJ

HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES



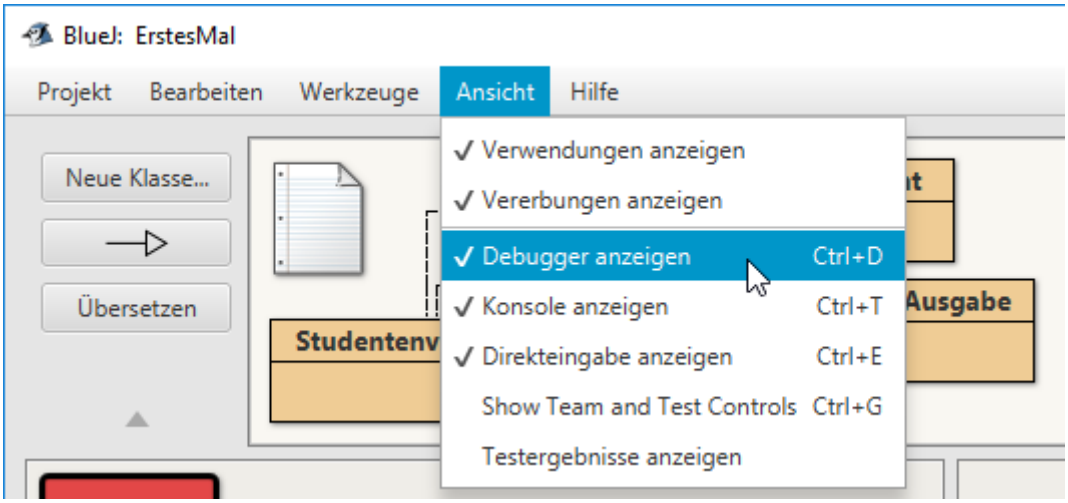
Der Debugger bietet in der unteren Zeile fünf verschiedene Funktionen an. Die ausgegraute ganz links heißt „Anhalten“ mit der z. B. eine länger laufende Ausgabe abgebrochen werden kann.

Mit „Schritt über“ („Step“) wird der nächste Schritt ausgeführt, dabei werden anstehende Methodenaufrufe einfach ausgeführt, der Debugger bleibt also in der gerade ausgeführten Methode und bekommt vom eventuell anstehenden Methodenaufruf nur mit, dass diese ausgeführt wurde und eventuell ein Ergebnis geliefert hat. Im konkreten Fall würde die Methode matrikelnummerExistiert(mat) ausgeführt und in die nächste Zeile 47 gegangen.

Mit „Schritt hinein“ („Step Into“) wird bei einem in der Zeile befindlichen Methodenaufruf in diese Methode gesprungen, so dass als nächster Schritt der erste Schritt dieser Methode ausgeführt wird. Befindet sich in der Zeile kein Methodenaufruf entspricht das Verhalten dem von „Schritt über“.

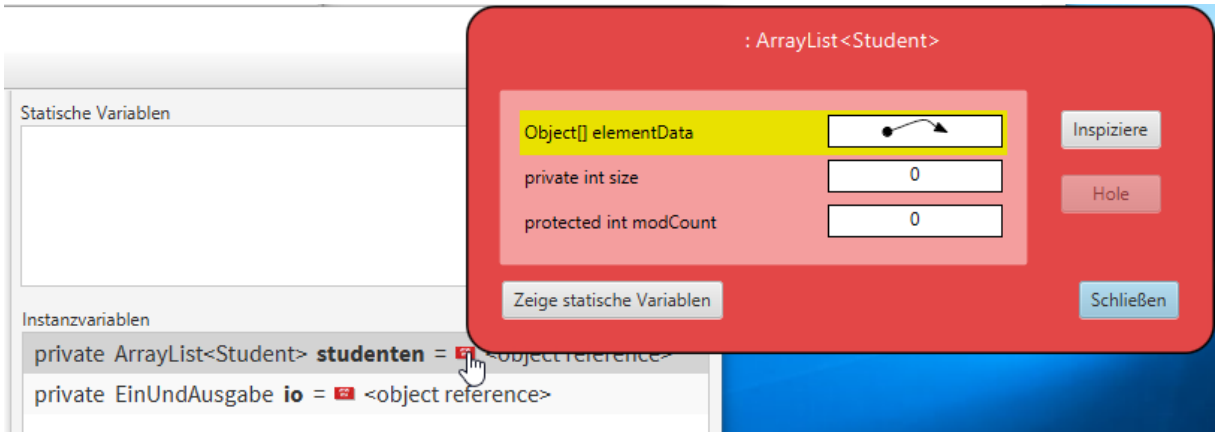
Mit „Fortsetzen“ („Continue“) läuft das Programm weiter, bis ein nächster Breakpoint erreicht wird oder das Programm endet.

Mit „Beenden“ („Terminate“) kann das gesamte Programm z. B. nach erfolgreicher Fehlersuche abgebrochen werden. Weiterhin können hiermit auch Programme abgebrochen werden, wenn der Debugger nicht läuft. Dafür kann dieser mit „Ansicht -> Debugger anzeigen“ sichtbar gemacht werden.




Im Debugger stehen links unter „Aufruffolge („Call Sequence“) die Methodenaufrufe, die zur aktuellen Zeile geführt haben. Die Aufrufe sind als Stapel organisiert, bei dem der neuste Aufruf oben auf den Stapel gelegt wird. Wird eine Methode beendet, wird sie vom Stapel entfernt, es wird also immer nur oben im Stapel hinzugefügt oder entfernt. Die Abbildung mit dem Debugger zeigt, dass für ein Objekt der Klasse Studentenverwaltung zunächst dialog() und darin studentHinzufuegen() aufgerufen wurde. Durch Anklicken einer Methode wird auf der rechten Seite die momentane Situation in der Methode, d. h. die Werte der Variablen, sichtbar. Durch einen Klick auf die Methode, wird diese auch im Editor markiert. In dem leeren Feld „Statische Variablen“ werden alle Klassenvariablen diese Klasse angezeigt, in diesem Fall gibt es keine mit „static“ deklarierten Variablen. Der irreführende Name „Instanzvariablen“ der korrekter „Objektvariablen“ oder „Exemplarvariablen“ heißen sollte, zeigt alle objekteigenen Variablen an. Bei den lokalen Variablen handelt es sich um Variablen die nur in der Methode

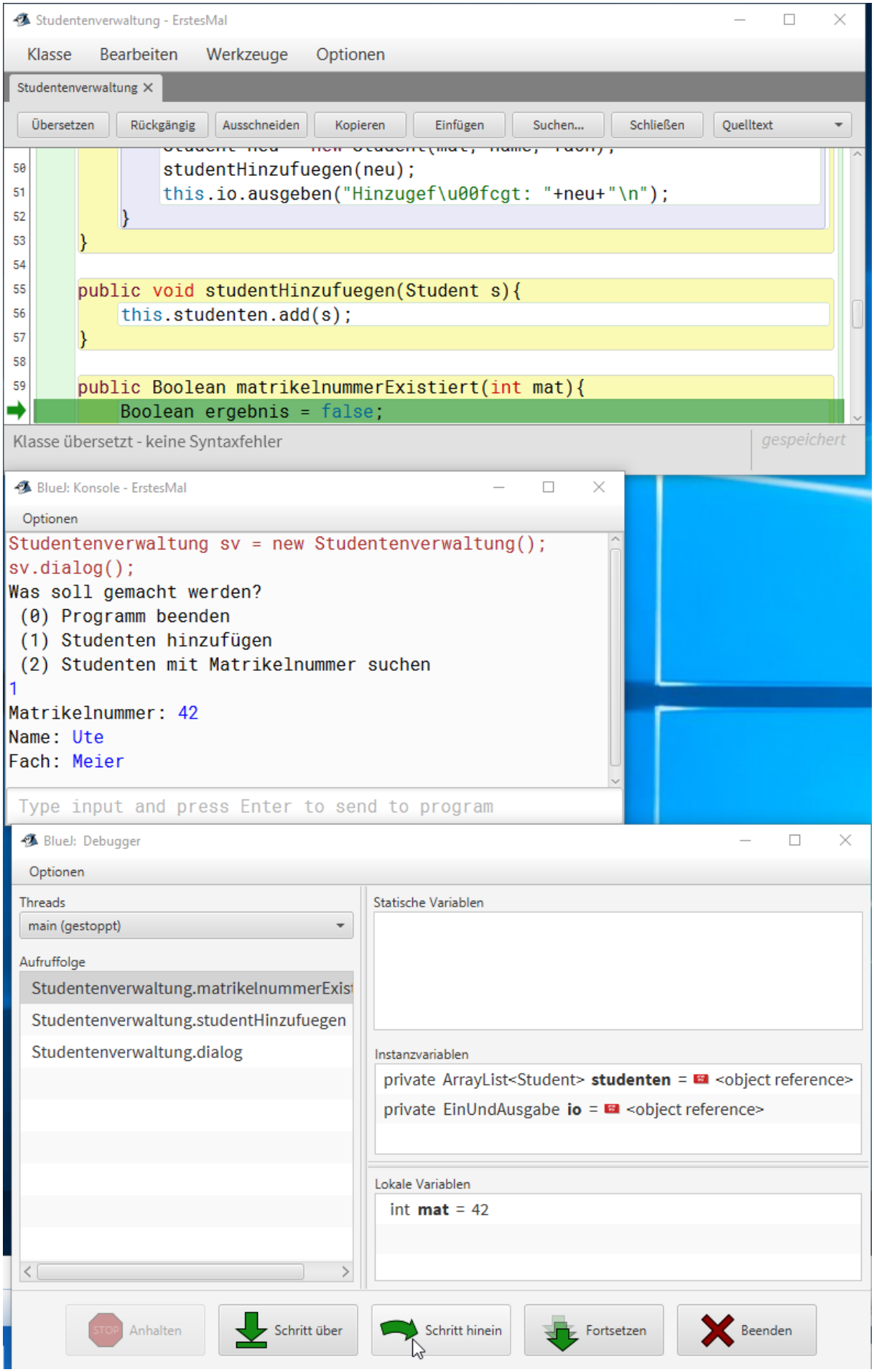
deklariert wurden und so auch nur dort nutzbar sind. Bei den lokalen Variablen werden weiterhin Parameter des Methodenaufrufs angezeigt. Enthalten die Variablen Objekte, wird dies wieder mit einem kleinen roten Kasten gekennzeichnet. Durch einen einfachen Linksklick auf einen solchen Kasten wird die bereits bekannte Detailansicht („inspiziere“) des Objektes geöffnet.



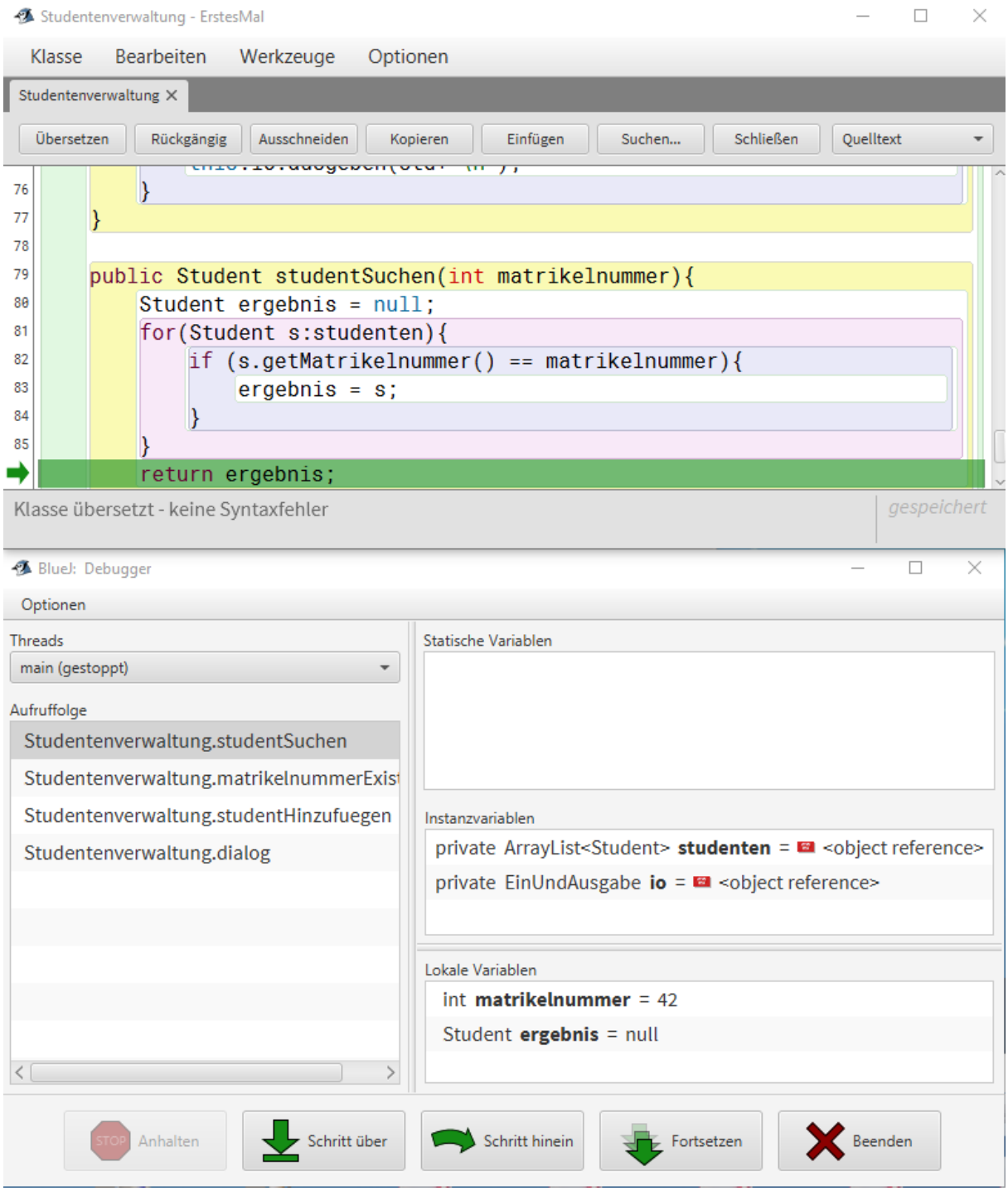

Im Debugger wird jetzt „Schritt hinein“ aufgerufen. Was an der „Aufruffolge“, den neuen lokalen Variablen und der jetzt markierten Zeile erkennbar ist.

# Nutzungshinweise für BlueJ

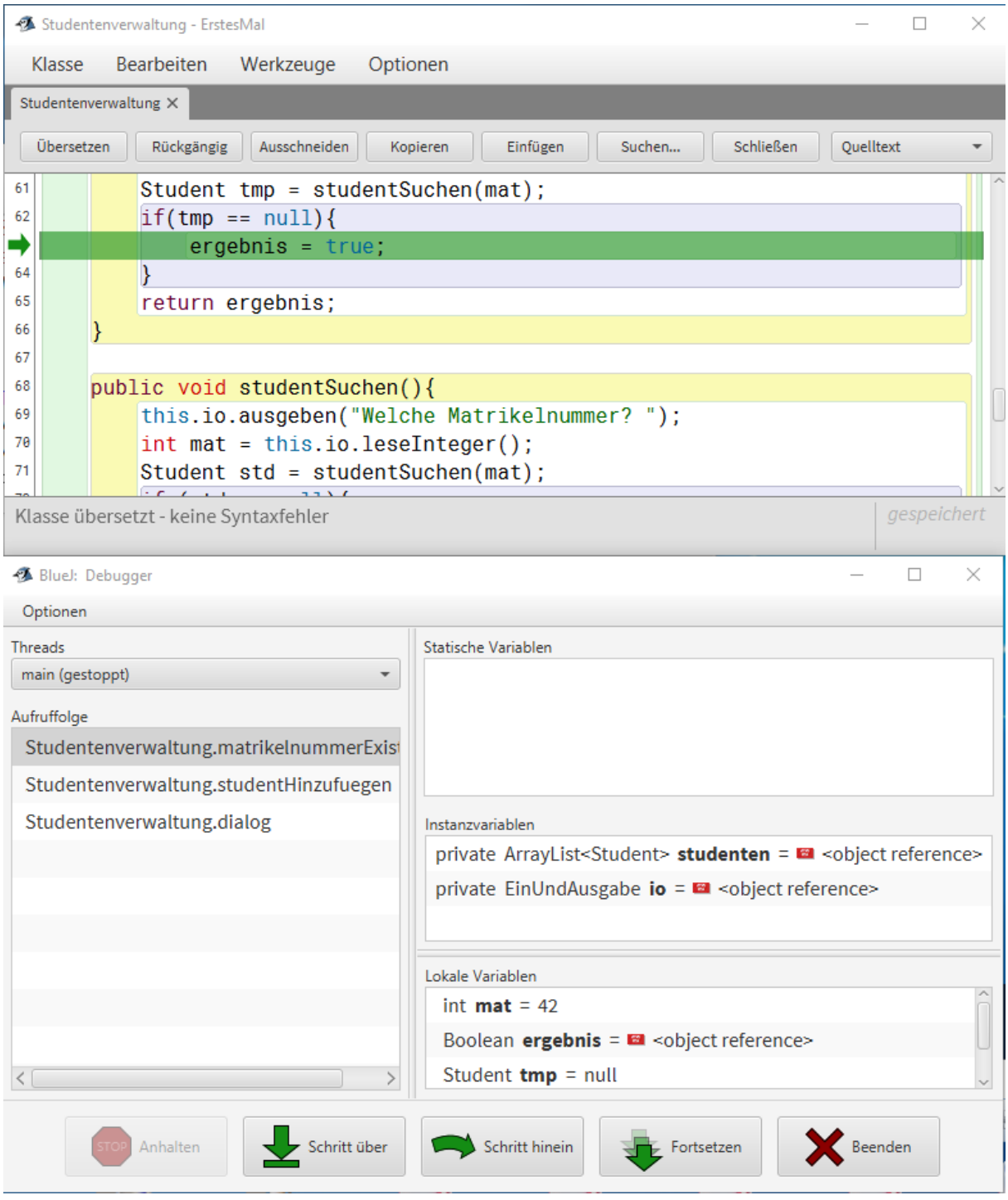
HOCHSCHULE OSNABRÜCK
UNIVERSITY OF APPLIED SCIENCES



Da jetzt keine Methodenaufrufe anstehen, haben „Schritt über“ und „Schritt hinein“ die gleiche Bedeutung. Nach einigen Schritten mit „Schritt hinein“ ist erkennbar, dass die Methode studentSuchen mit ergebnis=null das gewünschte Ergebnis liefert, da kein Student mit der übergenen Matrikernummer gefunden wird und das Ergebnis dann null sein soll.




Wahrscheinlich wird an folgender Stelle erkannt, dass hier das falsche Ergebnis durch die falsche Boolesche Bedingung zurückgeliefert wird.




Danach kann das Debuggen beendet werden. Das Fenster wird einfach geschlossen und das Programm neu übersetzt. Der Breakpoint sollte vor der nächsten Ausführung gelöscht werden.

# 7 Testen

Um zu prüfen, ob eine Software das gewünschte Verhalten hat, gibt es neben dem reinen Ausprobieren und der Erstellung einer neuen Klasse, die die zu testende Klasse nutzt, auch die Möglichkeit, Testfälle direkt in Java so zu programmieren, dass sie später bei Änderungen auch weiter nutzbar sind. Dieses Kapitel zeigt die Nutzung von JUnit zur Prüfung von Methoden. Es sei angemerkt, dass die systematische Qualitätssicherung ein komplexes Aufgabenfeld mit vielen Möglichkeiten auch im Testbereich ist. Weiterhin muss sich ein Entwickler immer verdeutlichen, dass nur gezeigt wird, dass die Testfälle funktionieren, er daraus nie auf ein vollständig korrektes Verhalten schließen darf. Die Erstellung von Testfällen ist zentraler Bestandteil der systematischen Software-Entwicklung.

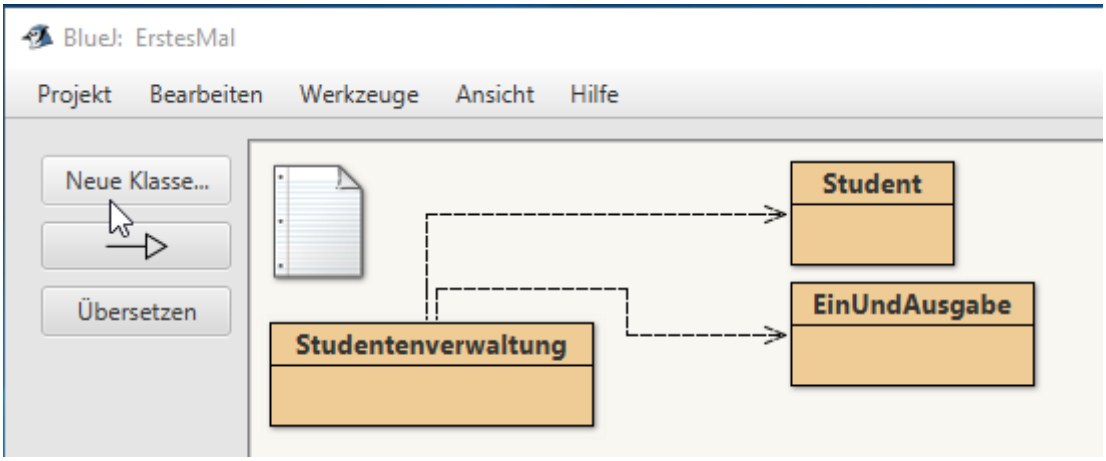
BlueJ unterstützt neben dem hier beschriebenen Weg auch die Möglichkeit, Aktionen, wie das Erzeugen von Objekten und das Aufrufen von Methoden aufzuzeichnen, um dann daraus JUnit-Testfälle zu erzeugen. Dieser für einige Schritte auch interessante Weg kann in anderen BlueJ-Dokumentationen nachgelesen werden und wird hier nicht weiter betrachtet, da er recht BlueJ-spezifisch ist.

Als Beispiel sollen die Methoden der Klasse Studentenverwaltung aus Kapitel 5.3 geprüft werden, in denen keine Nutzereingaben erfolgen. Der auch mögliche Test von Methoden mit Nutzereingaben wird hier nicht weiter verfolgt.

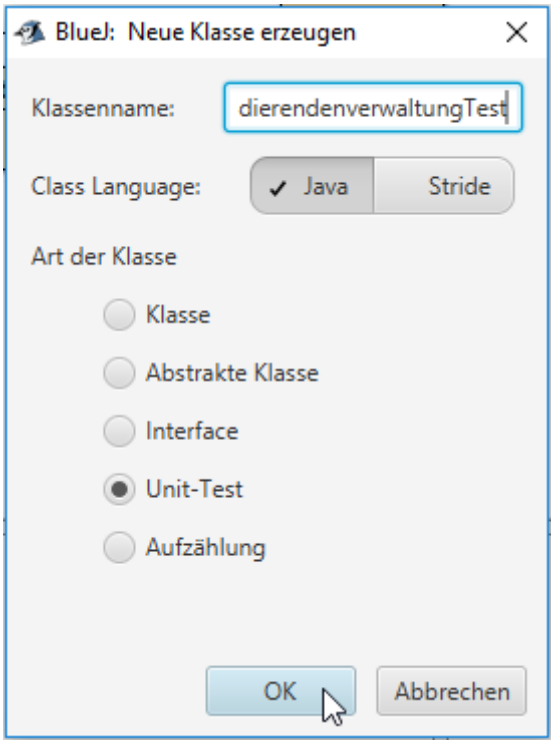
Um effizient testen zu können, muss dies in der Software berücksichtigt werden. Dies ist in der bisher erstellten Software der Fall, da z. B. wesentliche Methoden matrikelnummerExistiert(.) und studentSuchen(.) die Sichtbarkeit public haben und deshalb von außen aufgerufen werden können. Es gilt, dass generell in einem Projekt entschieden werden muss, wie die die Testbarkeit der Software garantiert wird.

## *7.1 Erzeugen einer Testklasse*

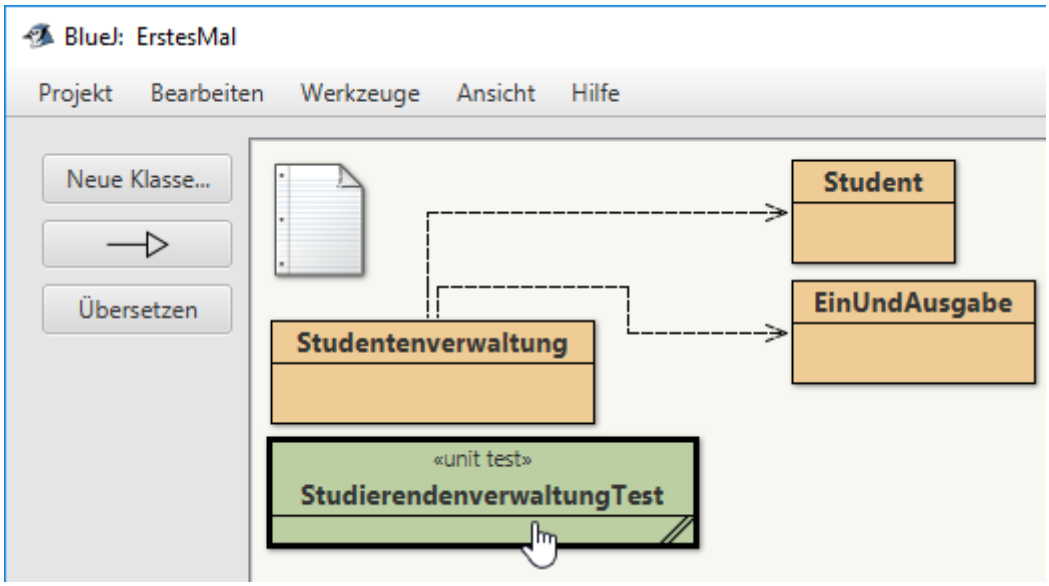
Zum Erzeugen einer Testklasse wird z. B. „Neue Klasse…“ angeklickt.




Es wird jetzt „Unit Test“ gewählt. Der Name der Klasse ist prinzipiell beliebig, es ist aber üblich, den Namen der zu testenden Klasse um das Wort „Test“ zu ergänzen.



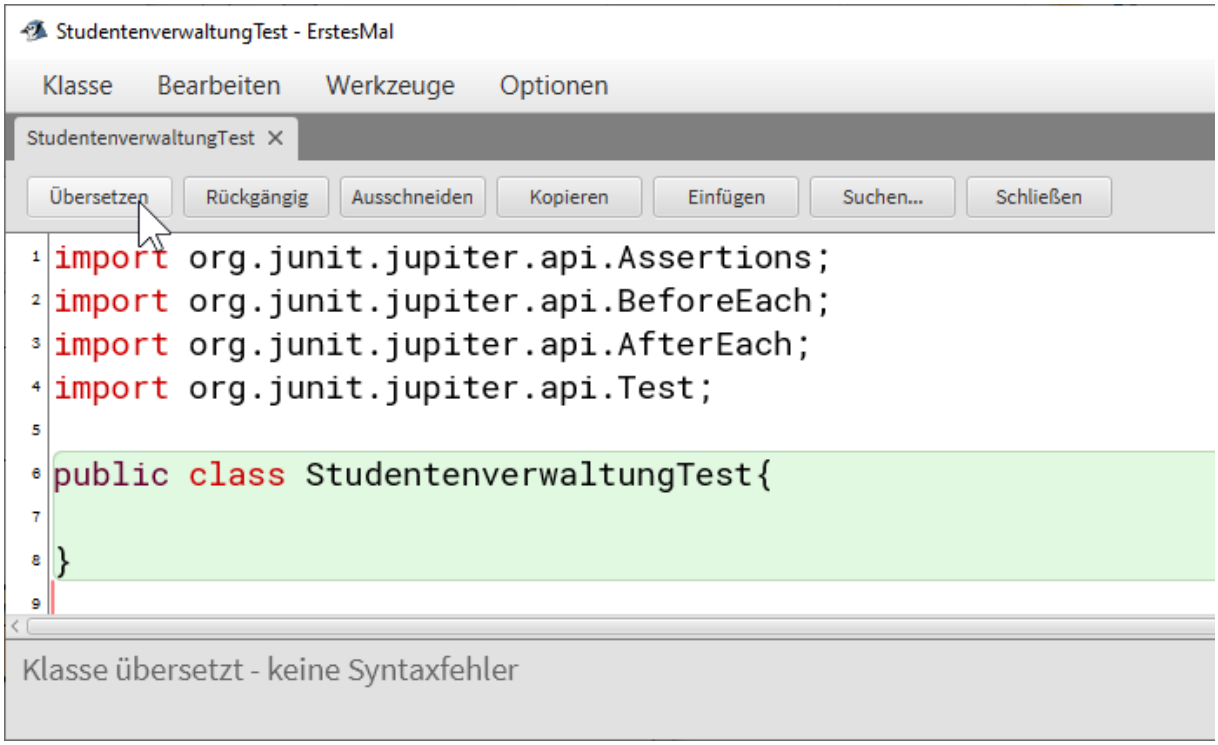

Es entsteht eine neue Klasse, die z. B. mit einem Doppelklick geöffnet wird.




Die Klasse ist bereits mit einigem Programmcode gefüllt. Hier wird geraten, diesen Code vollständig zu löschen und durch folgenden Code zu ersetzen, bzw. die erste Zeile zu ändern und den Klasseninhalt zu löschen. Es wird mit dem Knopf „Übersetzen“ geprüft, dass die Klasse laufffähig ist. In der KleukersSEU ist ein vergleichbarer Text bereits enthalten.

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
```

```
import org.junit.jupiter.api.Test;

public class StudentenverwaltungTest{

}
```



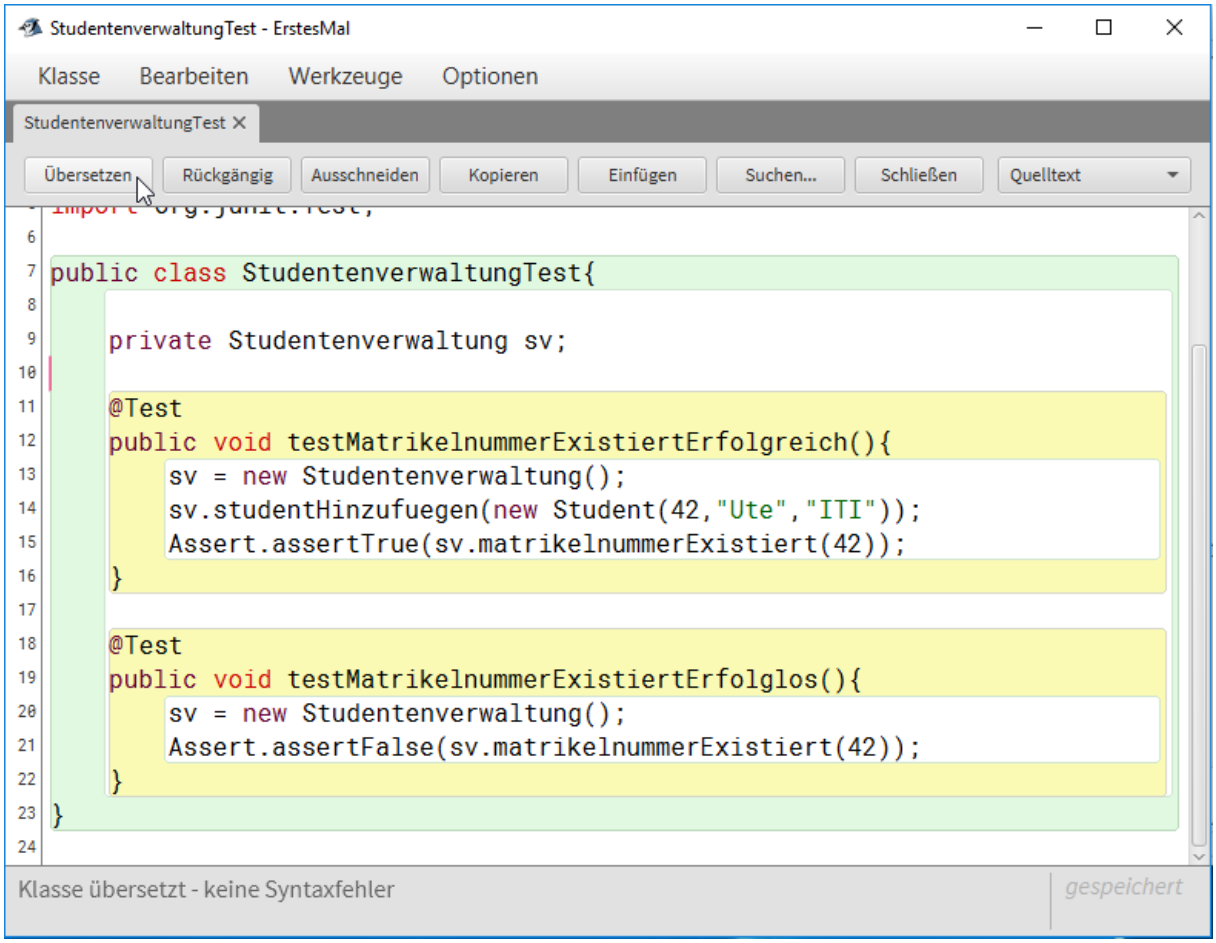

## 7.2 Ausführung von Tests

Danach werden schrittweise Testfälle ergänzt. Dies sind ganz normale Methoden, die Zusätzlich mit @Test am Anfang annotiert werden. Annotationen sind von anderen Programmen nutzbar, um bestimmte Aktionen auszuführen. Mit Annotationen wird deklarativ programmiert, da nur beschrieben wird was gemacht werden soll, aber nicht wie es gemacht werden soll. Die Tests werden dann von JUnit ausgeführt, das Bestandteil von BlueJ und allen größeren Entwicklungsumgebungen ist.

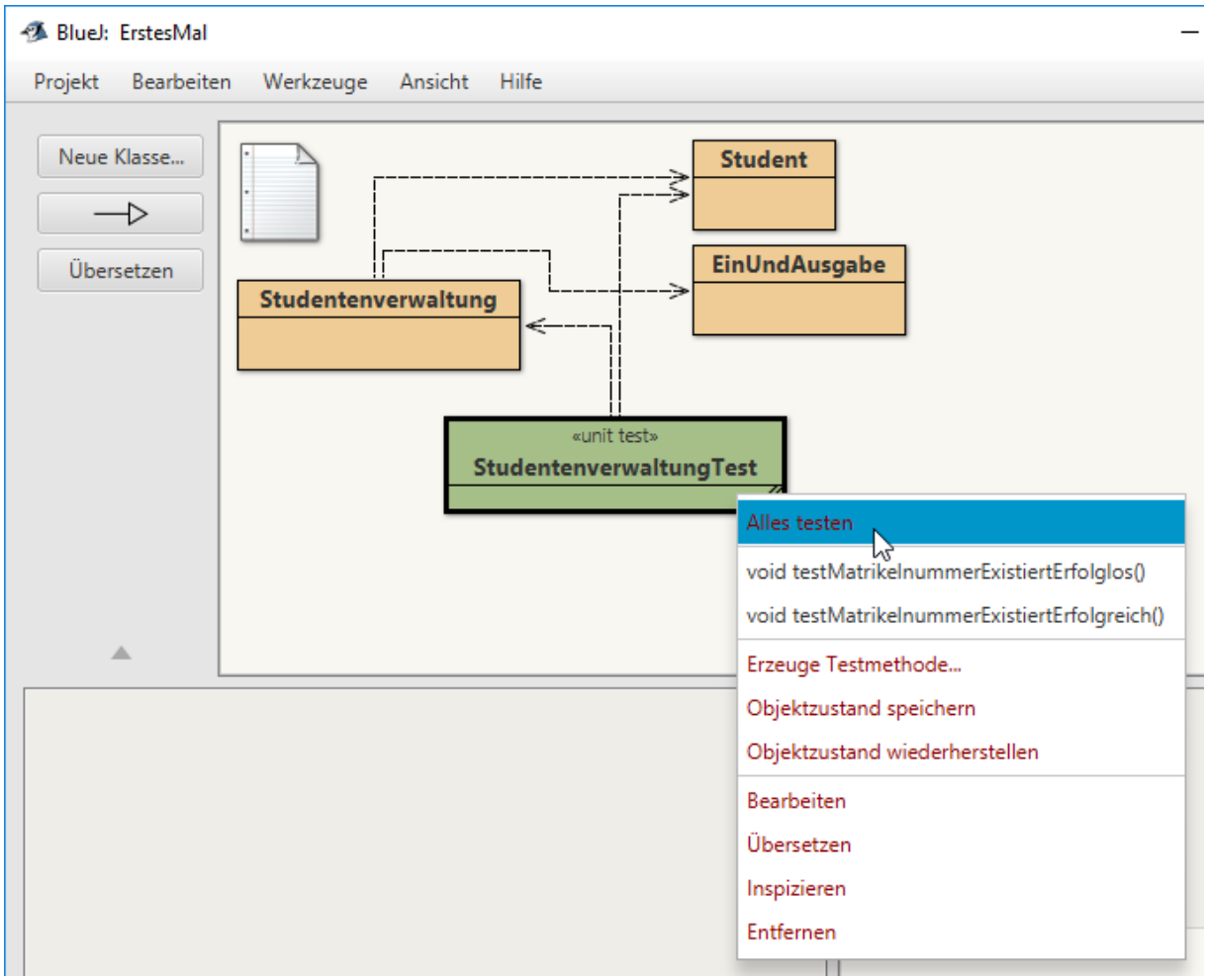
In den Tests beschreibt die Methode Assert.assertTrue(<Bed>) Boolesche Bedingungen <Bed> von denen geprüft wird, ob Sie nach true ausgewertet werden. Bei einer Auswertung nach „false“, wird dies als Fehler von JUnit vermerkt. Die folgende Abbildung zeigt zwei Testfälle, die Methodennamen sind frei wählbar, beginnen aber üblicherweise mit „test“ und beinhalten den Namen der zu testenden Methode und gegebenenfalls das erwartete Verhalten. Neben assertTrue() gibt es einige weitere Methoden, die Prüfungen zur Verfügung stehen.

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;

public class StudentenverwaltungTest{
```

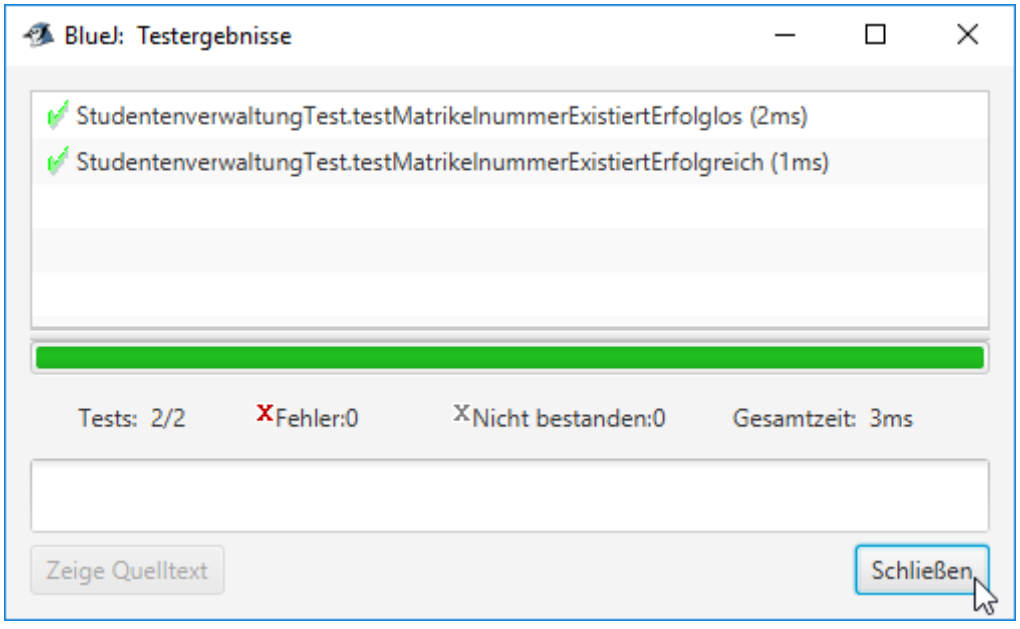
```
    @Test
    public void testMatrikelnummerExistiertErfolgreich(){
        sv = new Studentenverwaltung();
        sv.studentHinzufuegen(new Student(42,"Ute","ITI"));
        Assertions.assertTrue(sv.matrikelnummerExistiert(42)
            , "Studi mit MatNr 42 muss existieren");
    }

    @Test
    public void testMatrikelnummerExistiertErfolglos(){
        sv = new Studentenverwaltung();
        Assertions.assertFalse(sv.matrikelnummerExistiert(42)
            , "kein Studi mit Matnr 42");
    }
}
```




Zur Ausführung der Testfälle wird ein Rechtsklick auf der kompilierten Klasse gemacht und „Alles testen“ ausgewählt. Es ist erkennbar, dass auch einzelne Tests ausgeführt werden können.



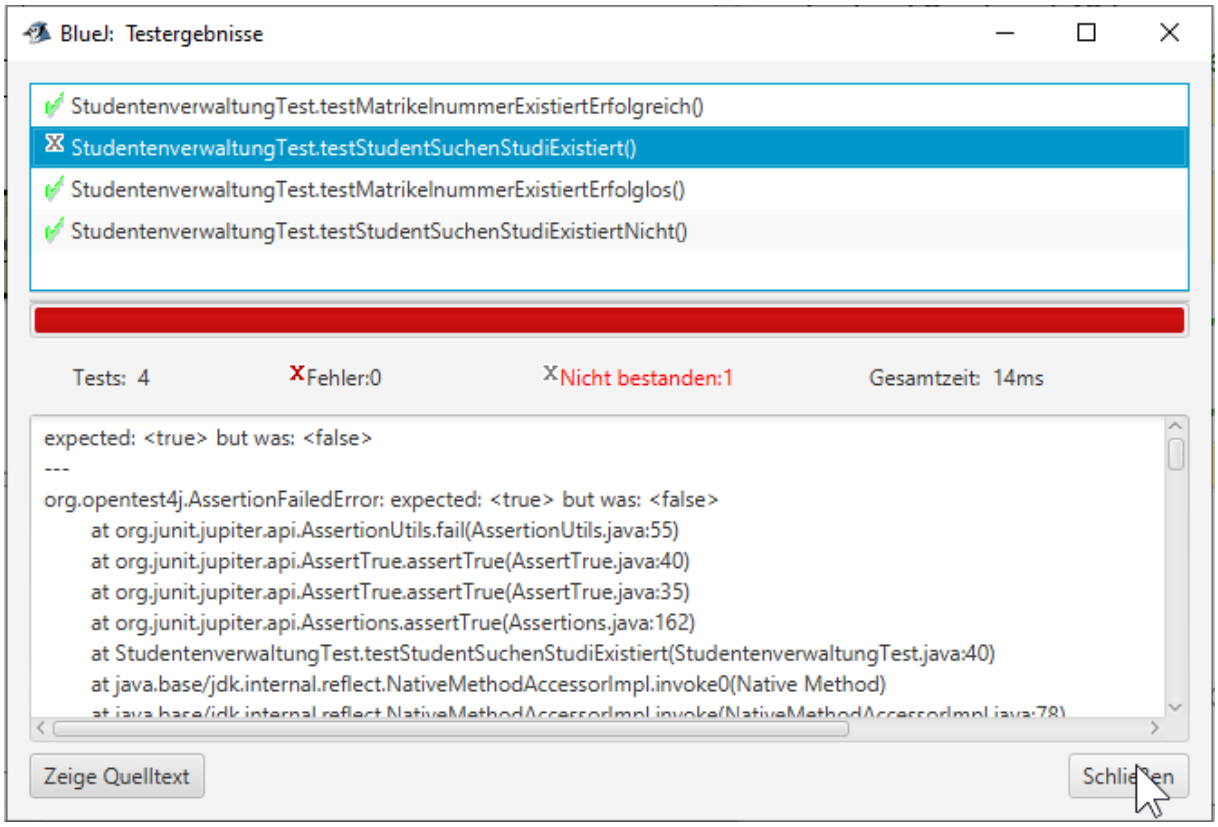

Nach der Ausführung öffnet sich ein Ergebnisfenster, in der für jeden Test angezeigt wird, ob er erfolgreich war. Wichtig ist der grüne Balken in der Mitte, der zeigt, dass alle Tests ohne Fehler durchgelaufen sind.



Nun werden noch zwei weitere Testfälle ergänzt, wobei der eine Testfall scheitern wird, da in diesem Fall der Testfall nicht in Ordnung ist. Generell gilt, dass bei einem gescheiterten Test immer zu prüfen ist, ob die getestete Software oder der Test Fehler enthält. Im konkreten Beispiel werden zwar zwei inhaltlich gleiche Studenten erzeugt, da aber mit == auf Identität geprüft wird, muss dieser Test scheitern.

```
@Test
public void testStudentSuchenStudiExistiertNicht(){
     sv = new Studentenverwaltung();
     Assertions.assertNull(sv.studentSuchen(42)
         , "Objekt mit Matnr 42 darf nicht existieren");
}

@Test
public void testStudentSuchenStudiExistiert(){
    sv = new Studentenverwaltung();
    sv.studentHinzufuegen(new Student(42,"Ute","ITI"));
    Assertions.assertTrue(
        sv.studentSuchen(42) == new Student(42,"Ute","ITI"));
}
```

Das Ergebnis sieht wie folgt aus, der gescheiterte Test muss angeklickt werden, um unten Details angezeigt zu bekommen.



Weiterhin kann in JUnit eine einheitliche Testumgebung, eine Test-Fixture angeben werden, die für jeden Test benutzt wird. Die Grundidee ist, dass vor jedem Test die mit @BeforeEach annotierte Methode, dann der Test und dann die mit @AfterEach annotierte Methode ausgeführt werden. Die korrigierte vollständige Testklasse sieht wie folgt aus.

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;

public class StudentenverwaltungTest{

    private Studentenverwaltung sv;
    private Student dummy;

    @BeforeEach
    public void setUp(){
      this.sv = new Studentenverwaltung();
      this.dummy = new Student(42,"Ute","ITI");
    }

    @Test
    public void testMatrikelnummerExistiertErfolgreich(){
        this.sv.studentHinzufuegen(dummy);
        Assertions.assertTrue(sv.matrikelnummerExistiert(42)
            , "Studi mit MatNr 42 muss existieren");
    }

    @Test
    public void testMatrikelnummerExistiertErfolglos(){
        Assertions.assertFalse(sv.matrikelnummerExistiert(42)
            , "kein Studi mit Matnr 42");
    }

    @Test
    public void testStudentSuchenStudiExistiertNicht(){
         Assertions.assertNull(sv.studentSuchen(42)
             , "Objekt mit Matnr 42 darf nicht existieren");
    }

    @Test
    public void testStudentSuchenStudiExistiert(){
        sv.studentHinzufuegen(dummy);
        Assertions.assertSame(sv.studentSuchen(42), dummy
            , "identische Objekte erwartet");
    }
}
```

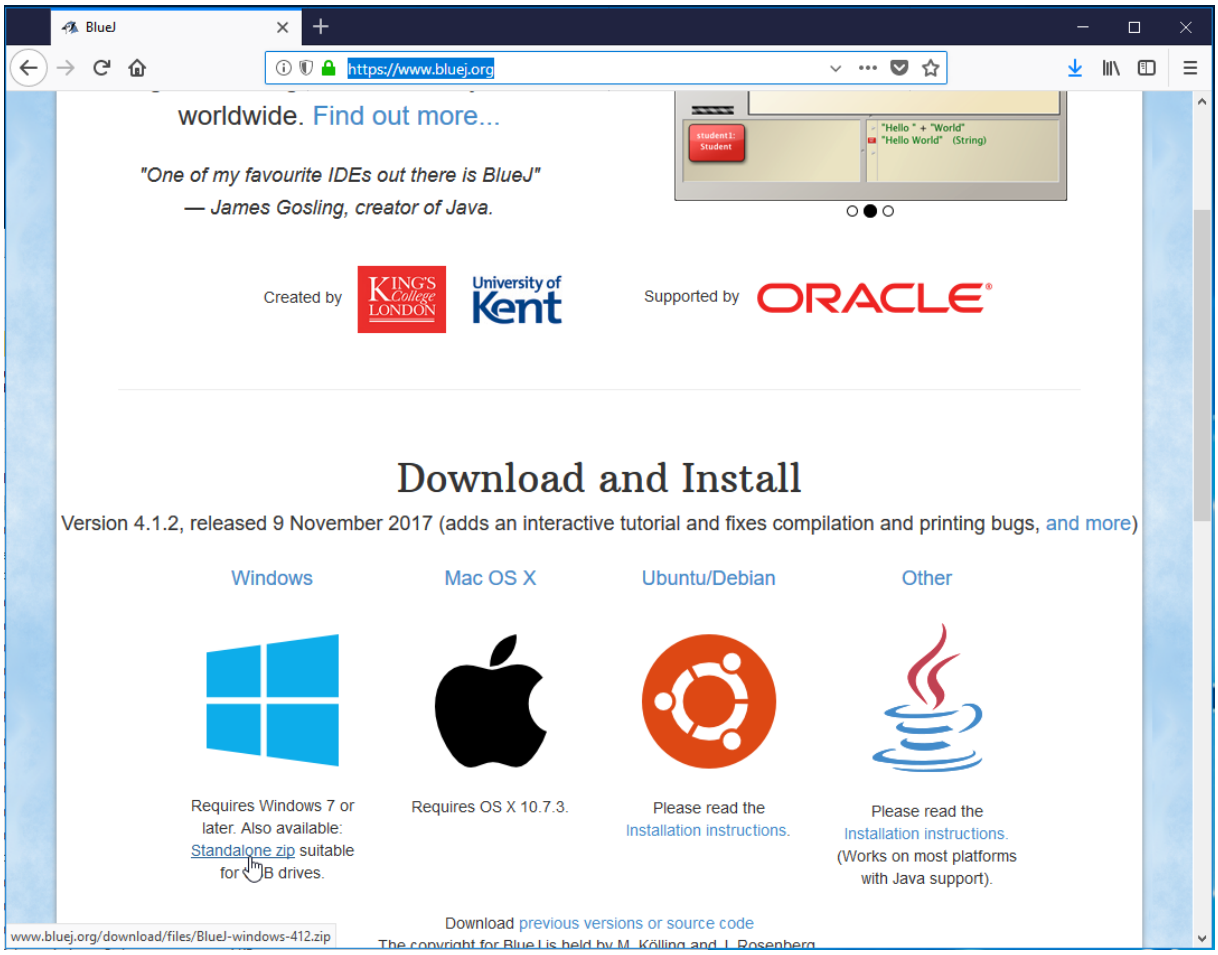
Eine wesentlich detailliertere Einführung in das Testen mit JUnit befindet sich in:

S. Kleuker, Qualitätssicherung durch Softwaretests, 2. aktualisierte und erweiterte Auflage, Springer Vieweg, Wiesbaden, 2019

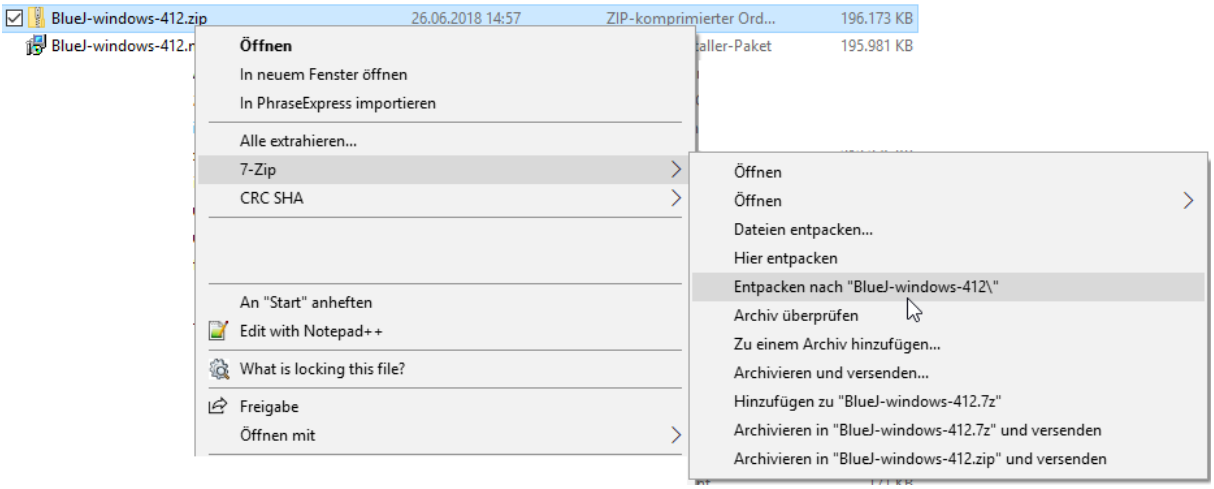
# 8 Portable Version von BlueJ

Portable Programme haben gegenüber anderen die Vorteile, dass sie keine Änderungen in den Systemeinstellungen vornehmen und auf allen Rechnern prinzipiell nutzbar sind, die das gleiche Betriebssystem nutzen. Ein Nachteil ist, dass solche Programme ohne weitere Einstellungen keine Administrationsrechte haben und so nicht auf alle Dateien zugreifen können. Ein Einsatzszenario einer portablen Installation ist die Nutzung von BlueJ auf einem Rechner auf dem weder BlueJ noch Java installiert sind. Da die Installation portabel ist, kann sie leicht andere Rechner kopiert und dort genutzt werden.

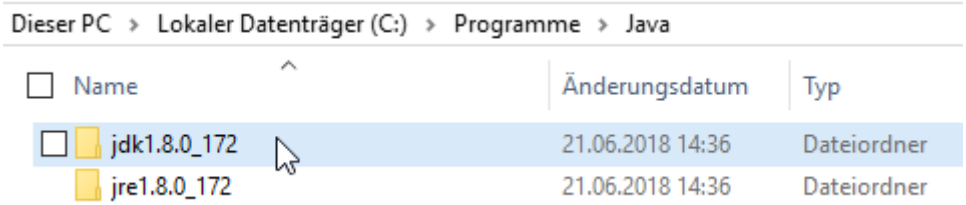
Die benötigte Datei wird von https://www.bluej.org/ über den Link „Standalone zip“ heruntergeladen.




Die heruntergeladene Zip-Datei wird in den Ordner kopiert, in dem ein Unterordner mit der BlueJ-Installation entstehen soll. Abhängig vom installierten Zip-Programm wird über einen Rechtsklick und die passende Auswahl das Auspacken im passenden Ordner gerstartet.




In dem entstehenden BlueJ-Ordner befindet sich nutr eine bluej.exe-Datei mit der BlueJ gerstartet werden kann, insofern sich auf dem benutzten Rechner eine halbwegs aktuelle Java-Installation befindet. Ist dies nicht der Fall oder soll eine eigene Java-Version genutzt werden, kann z. B. der Ordner der JDK-Installation einfach in den BlueJ-OPrdner kopiert werden. Es ist zu beachten, dass eine JDK-Version und die zu den Zielrechnern passende 32- oder 64-Bit-Version kopiert wird. Da aktuelle alle eine 64-Bit-Version des Betriebssystems nutzen und neuere Java-Version nut für 64-Bit existieren, ist generell diese Variante vorzuziehen. Da die 32-Bit-Variante aber auch auf 64-Bit-Systemen läuft, was anderesherum nicht der Fall ist, wäre auch dies eine Alternative. Natürlich können auch beide Versionen kopiert werden, wobei dann die Ordnernamen gegebenenfalls anzupassen sind. Die nachfolgende Abbildung zeigt einen typischen Installationsort, von dem aus die Java-Version kopiert werden kann.

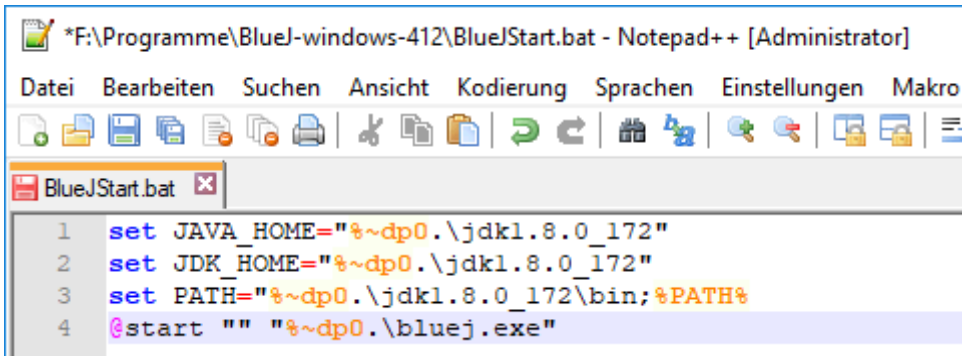



Zum Starten des Programm wird eine Batch-Datei mit einem beliebigen Editor, z. B. Notepad++ (https://portableapps.com/apps/development/notepadpp_portable) angelegt. Die Datei hat die Endung „.bat“, der Name am Anfang ist beliebig, hier wird „BlueJStart“ gewählt.

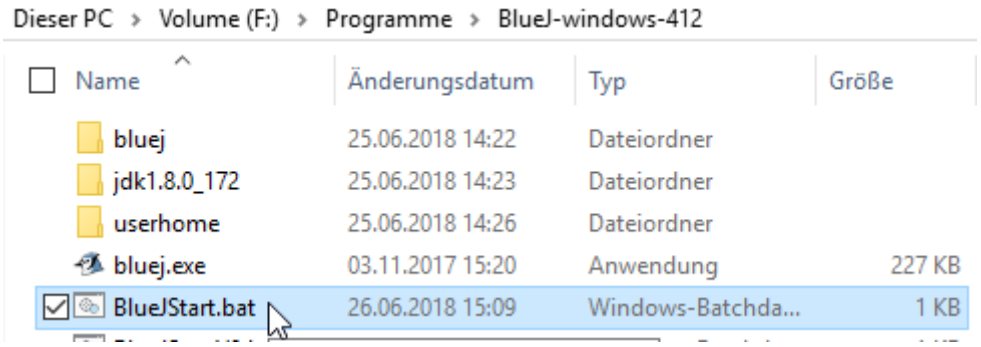
Es werden folgende Befehle genutzt.

```
set JAVA_HOME="%~dp0.\jdk1.8.0_172"
set JDK_HOME="%~dp0.\jdk1.8.0_172"
set PATH="%~dp0.\jdk1.8.0_172\bin;%PATH%
@start "" "%~dp0.\bluej.exe"
```

Mit den Zeilen werden die beiden Systemvariablen JAVA_HOME und JDK_HOME lokal für eine Programmausführung gesetzt, der Pfad zur Java-Installation ergänzt und BlueJ aufgerufen.




Das vollständige Verzeichnis sieht wie folgt aus. BlueJ wird durch einen Doppelklick auf „BlueJStart.bat“ gestartet. Eine Verknüpfung mit dieser Datei kann z. B. auf die Oberfläche gelegt werden.



# 9 Installation des Screenshot-Werkzeugs Faststone Capture

Mit einem Screenshot-Werkzeug besteht die Möglichkeit, den aktuellen Monitorinhalt „abzufotografieren“ und das entstandene Bild abzuspeichern. Solche Programme sind in verschiedenen Situationen sehr hilfreich, zwei wesentliche sind:

- Die Erstellung einer Benutzungsdokumentation mit der die Bedienung der entstandenen Software beschrieben werden kann. Sehr hilfreich ist es dabei, wenn der Mauszeiger auch in den Bildern sichtbar wird.
- Die Dokumentation von Fehlersituationen, dabei wird neben dem Programmcode ein Foto der aktuellen Situation auf dem Bildschirm ergänzt. Das Foto erspart oft eine langwierige Beschreibung, wie es zur kritischen Situation kommt.

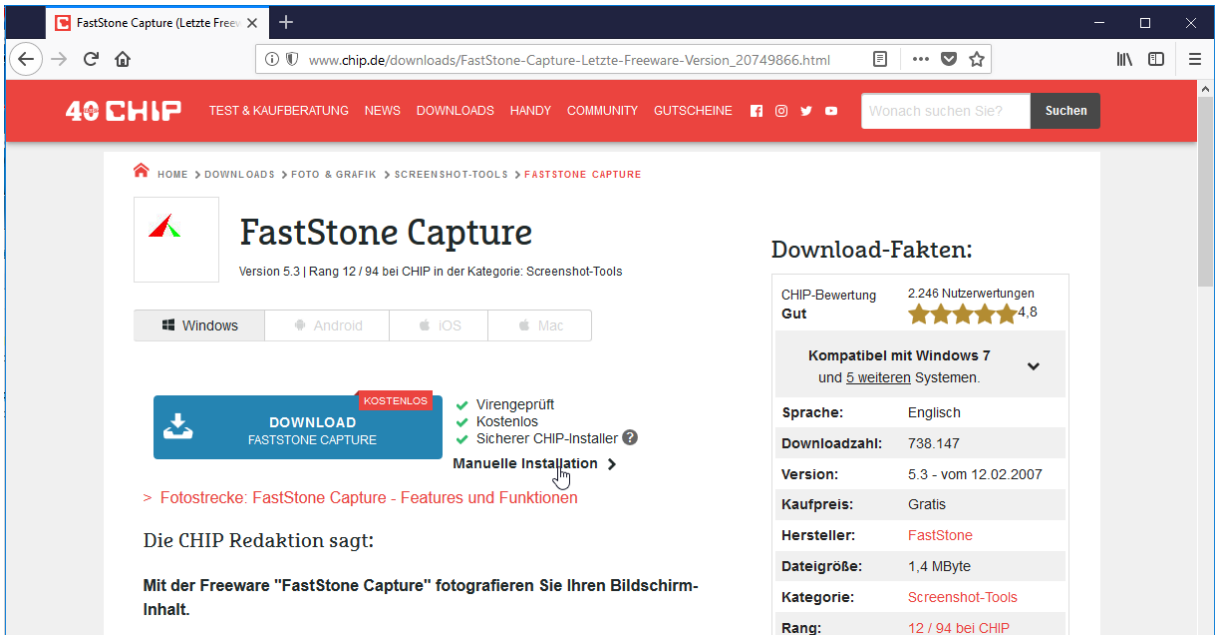
In der Programmierausbildung kann solch ein Werkzeug auch genutzt werden, um Fortschritte in der systematischen Programmerstellung zu dokumentieren.

Die konkrete Auswahl des Werkzeugs ist dabei irrelevant. Hier wird exemplarisch das Werkzeug FastStone Capture vorgestellt, das allerdings nur bis zur völlig ausreichenden Version 5.3 frei genutzt werden kann. Es werden weiterhin nur grundlegende Funktionen vorgestellt, weitere Möglichkeiten des Programms sind leicht selbst erarbeiten.

## *9.1 Herunterladen und Installieren*

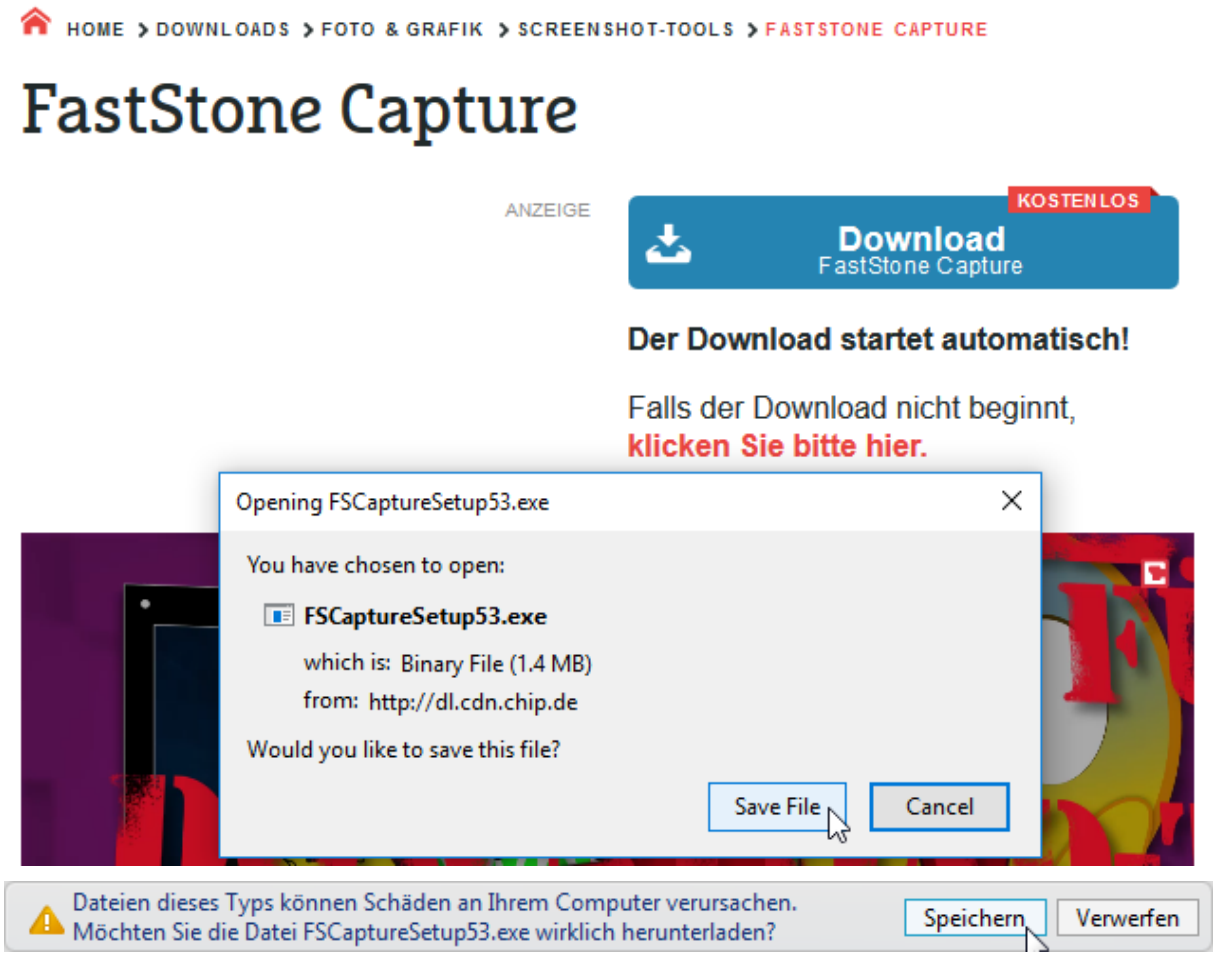
Da nicht die aktuellste, da kostenpflichtige, Version des Programms genutzt werden soll, muss die freie Version gefunden werden. Eine gute Quelle ist die Web-Seite der Zeitung Chip, dabei wird z. B. folgender Link http://www.chip.de/downloads/FastStone-Capture-Letzte-Freeware-Version_20749866.html genutzt. Falls dieser nicht mehr funktioniert, sollte die Google-Suche nach „faststone capture freeware“ Erfolg bringen.

Es wird der Link „Manuelle Installation“ gedrückt.

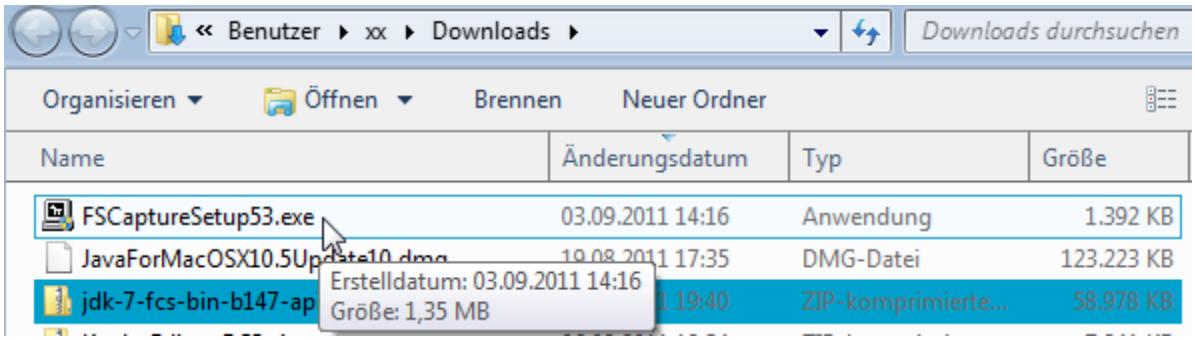



Es wird auf „Download-Server CHIP Online“ geklickt.

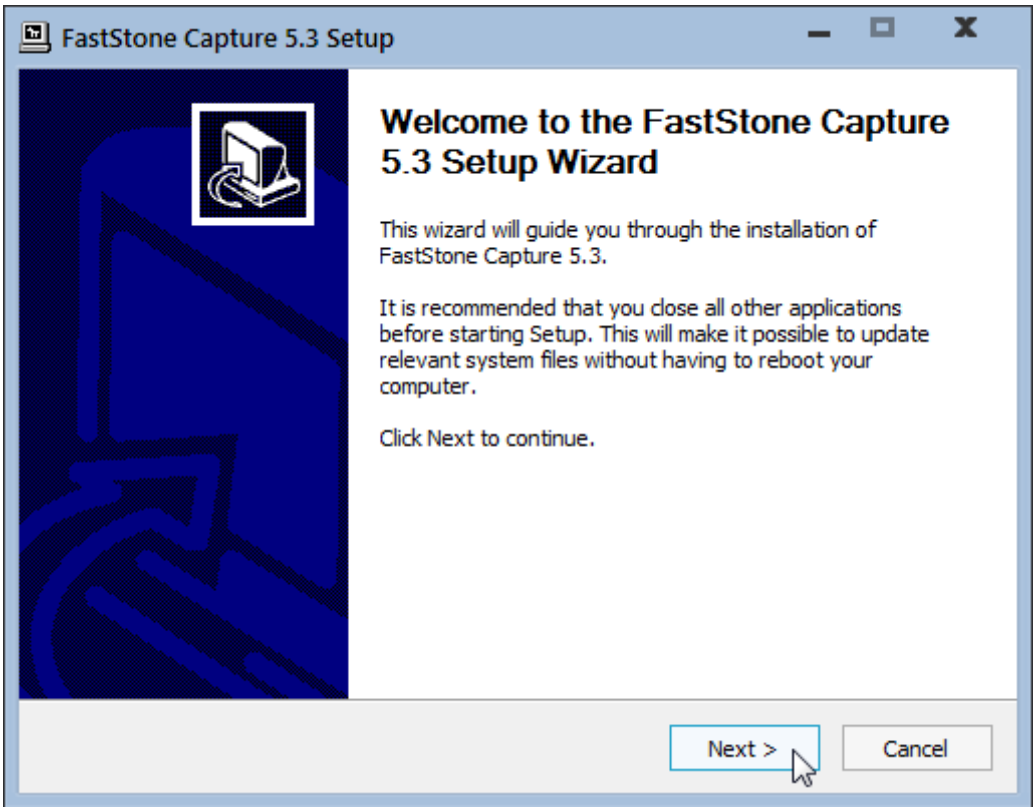

Dann hängt es etwas von der Einstellung des Browsers ab, ob der „klicken Sie bitte hier“-Link noch gedrückt werden muss, oder ob der Download automatisch startet.



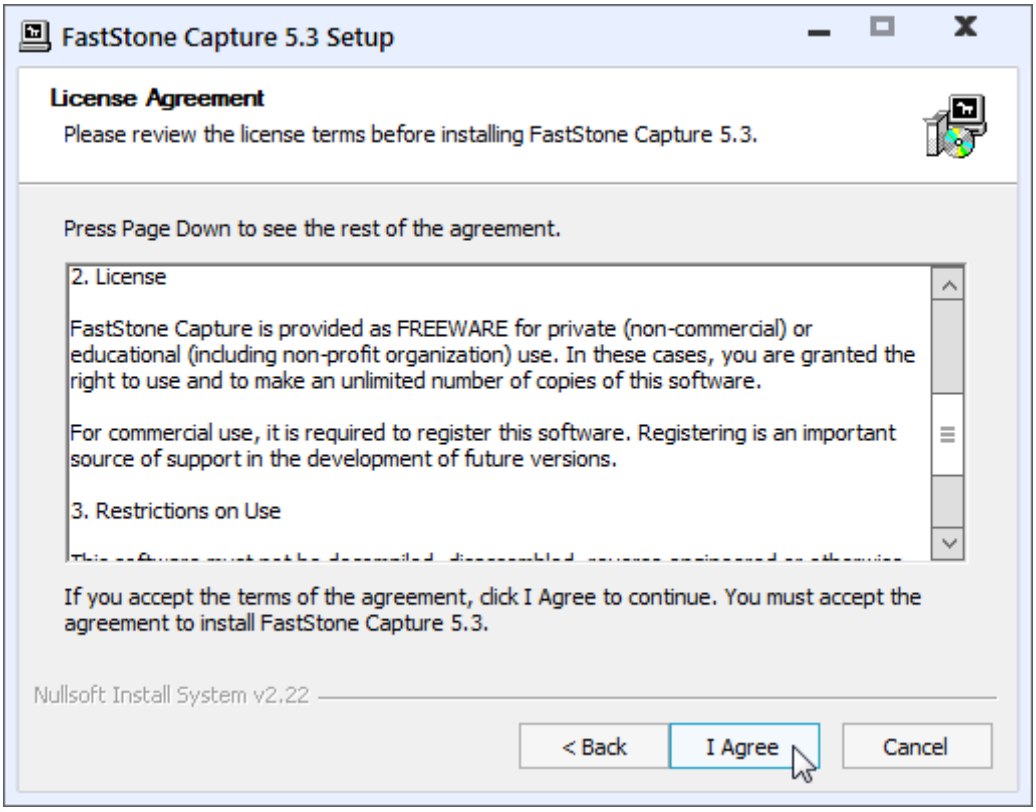

Die Installation wird durch einen Doppelklick auf der heruntergeladenen Datei gestartet.



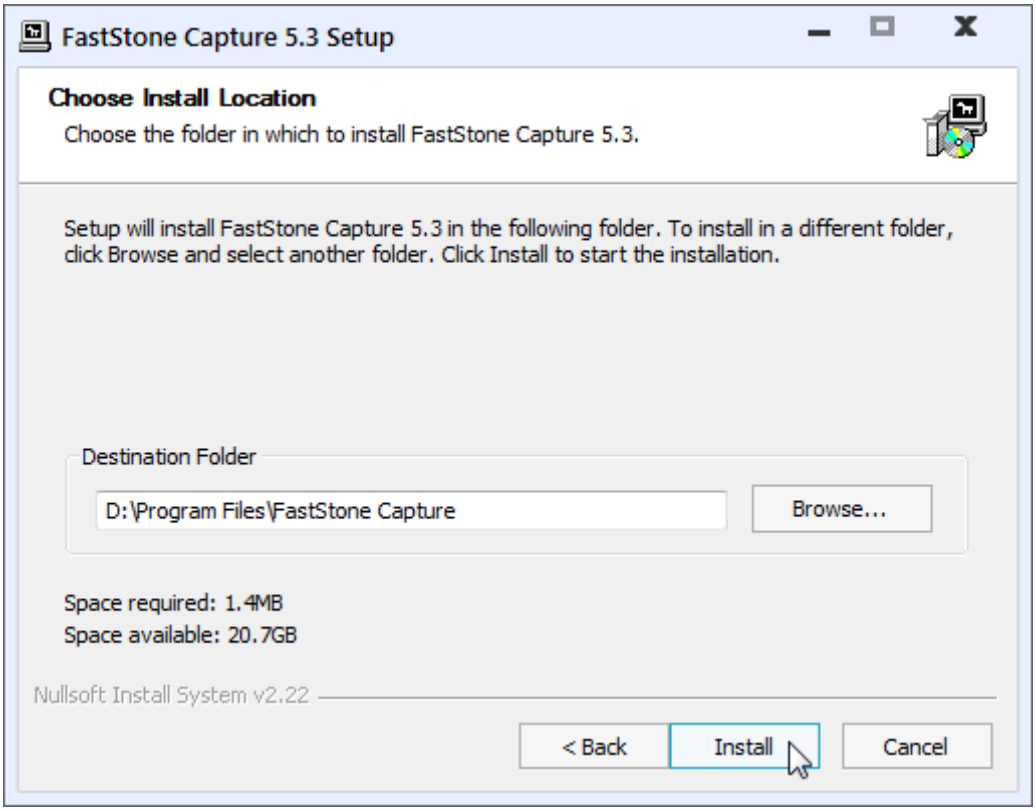

Das Startfenster wird mit „Next>“ verlassen.



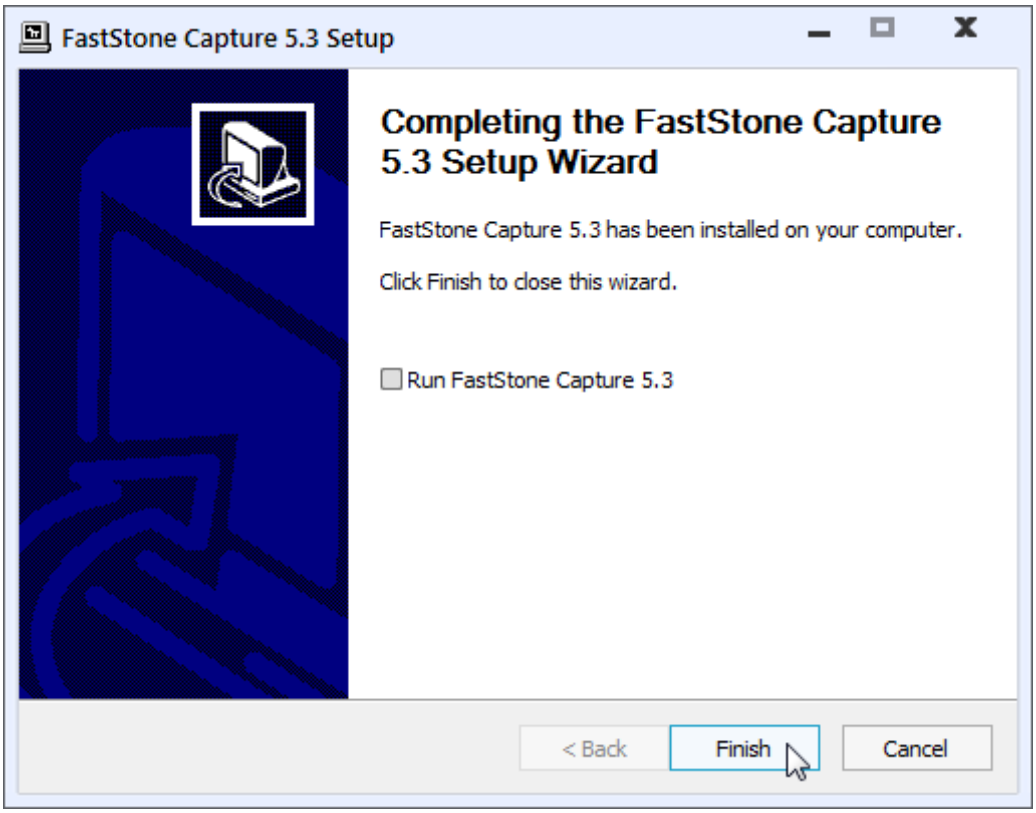

Die Lizenzbedingungen gelesen (u. a. Freeware für educational use) und mit „I Agree“ bestätigt.



Danach kann das Installationsverzeichnis gewählt werden und es wird „Install“ gedrückt.

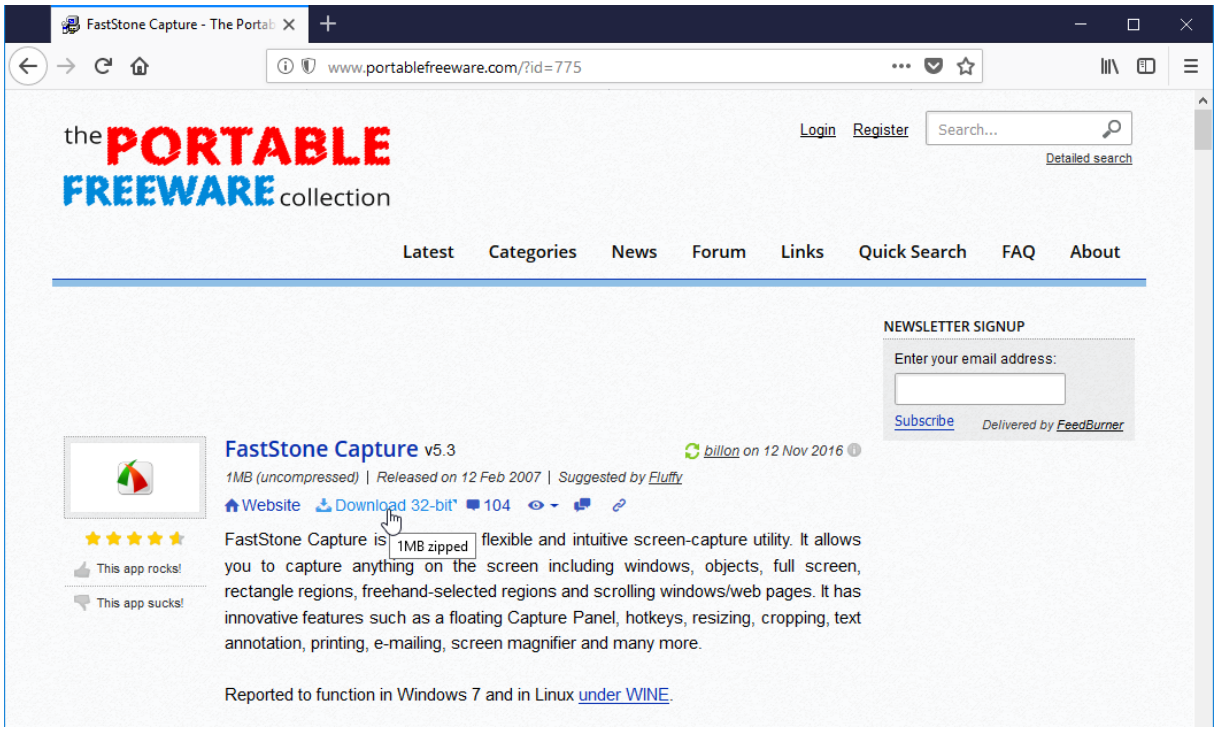



Die Installation wird mit „Finish“ abgeschlossen.



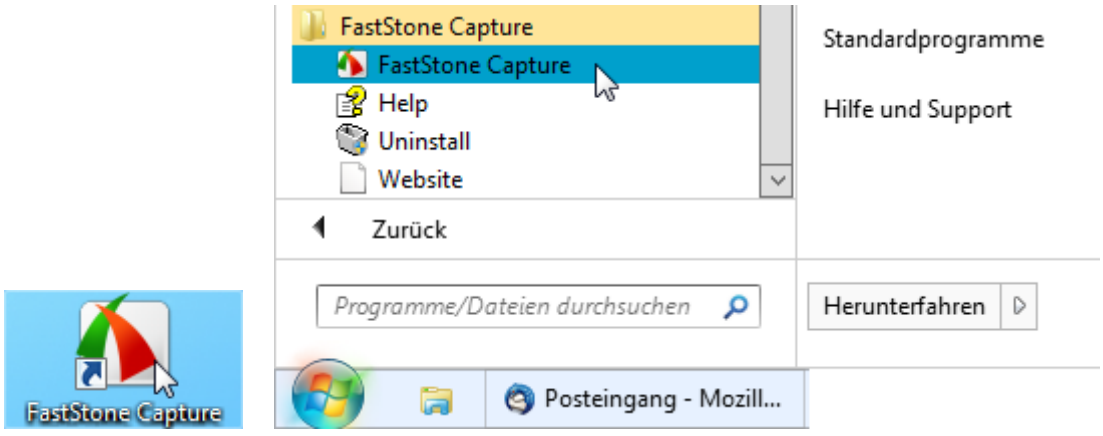

## 9.2 Portable Alternative

Das gleiche Programm gibt es auch als sogenannte „portable Software“. Diese hat den großen Vorteil nicht installiert werden zu müssen, so dass sie ohne jedwede Installationsrechte auf Rechnern nutzbar ist. Das Programm trägt sich dann natürlich nicht in das Startmenü ein und muss jeweils direkt aufgerufen werden. Wer sich intensiver mit portabler Software beschäftigt, was ratsam für Personen sein kann, die auf mehreren Rechnern arbeiten und „ihre“ Programme und Daten immer z. B. auf einem USB-Stick dabei haben wollen, wird schnell herausfinden, dass es auch Software zum Start und zur Verwaltung portabler Programme gibt. Ein Quelle zum Download ist z. B. http://www.portablefreeware.com/?id=775. Dort den „Download 32-bit“-Link benutzen.

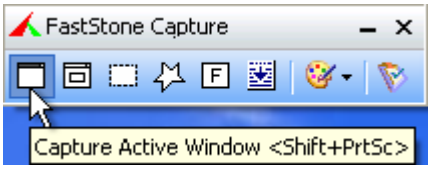



## 9.3 Erste Schritte in der Nutzung

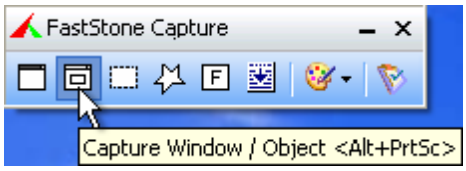
Der Start des Programms erfolgt z. B. durch einen Doppelklick auf das Icon oder über den Eintrag im Startmenü.



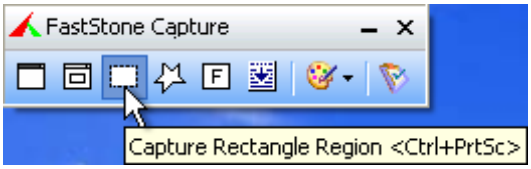

Die wesentlichen Knöpfe für verschiedene Snapshot-Varianten sind selbsterklärend. (Da sich das Werkzeug nicht selbst fotografieren kann, es blendet sich sinnvollerweise für Fotos aus, wurde ein virueller Rechner fotografiert.)

Erster Knopf von links (aktuelles Fenster): 

Zweiter Knopf von links (Fenster ist auswählbar):

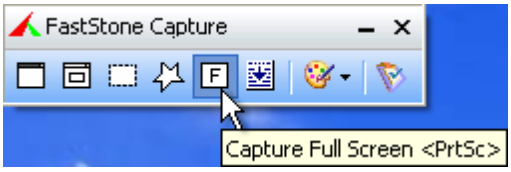



Dritter Knopf von links (beliebiges Rechteck):

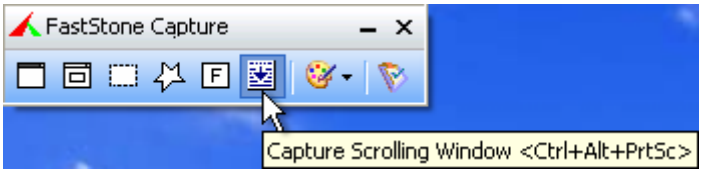



Vierter Knopf von links (beliebige Form):

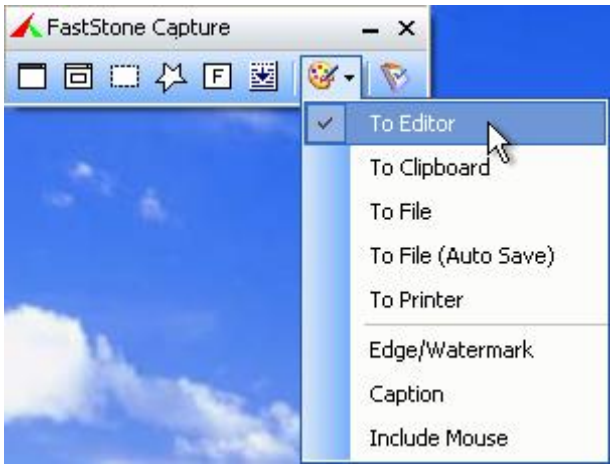

Fünfter Knopf von links (gesamter Bildschirm):



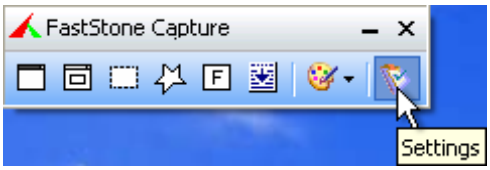

Sechster Knopf von links (Fenster auch mit nicht sichtbaren Bereich):



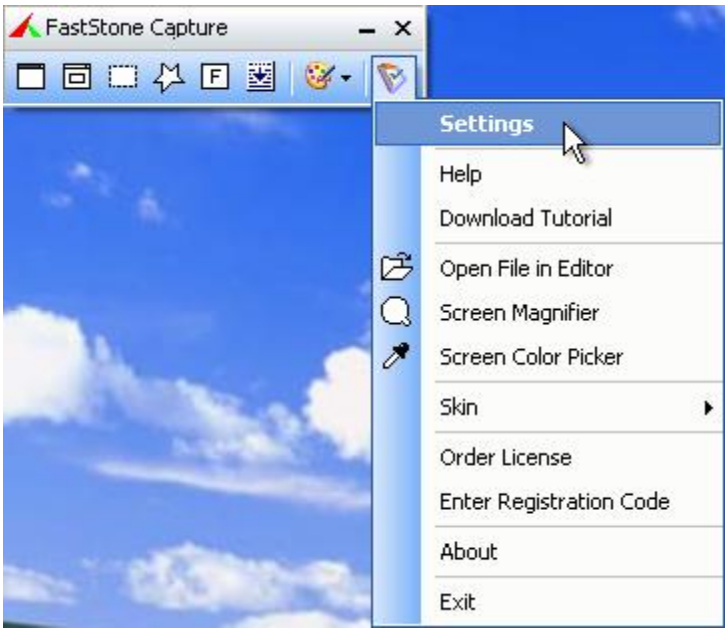

Siebter Knopf zum Herunterklappen ermöglicht die Auswahl, wo sich das Bild befinden soll. Die Standard-Einstellung „zum Editor“ kann so gelassen werden.




Der Knopf ganz rechts führt zu den Einstellungsmöglichkeiten.



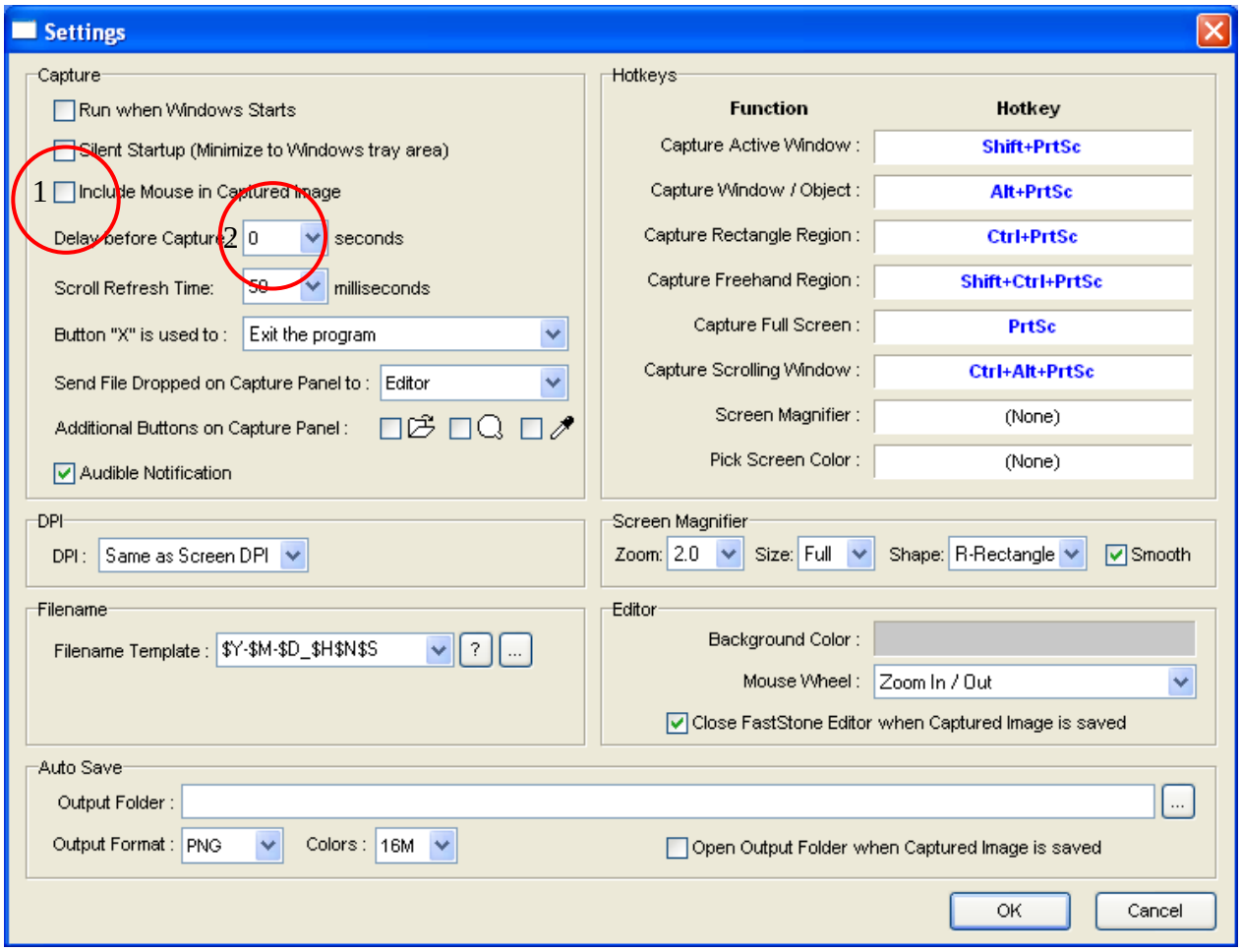

Ohne weitere Einstellungen, wird diese Funktionalität sofort ausgeführt. Um dieses zu ändern, muss der „Settings“-Knopf ganz rechts gedrückt und „Settings“ gewählt werden.



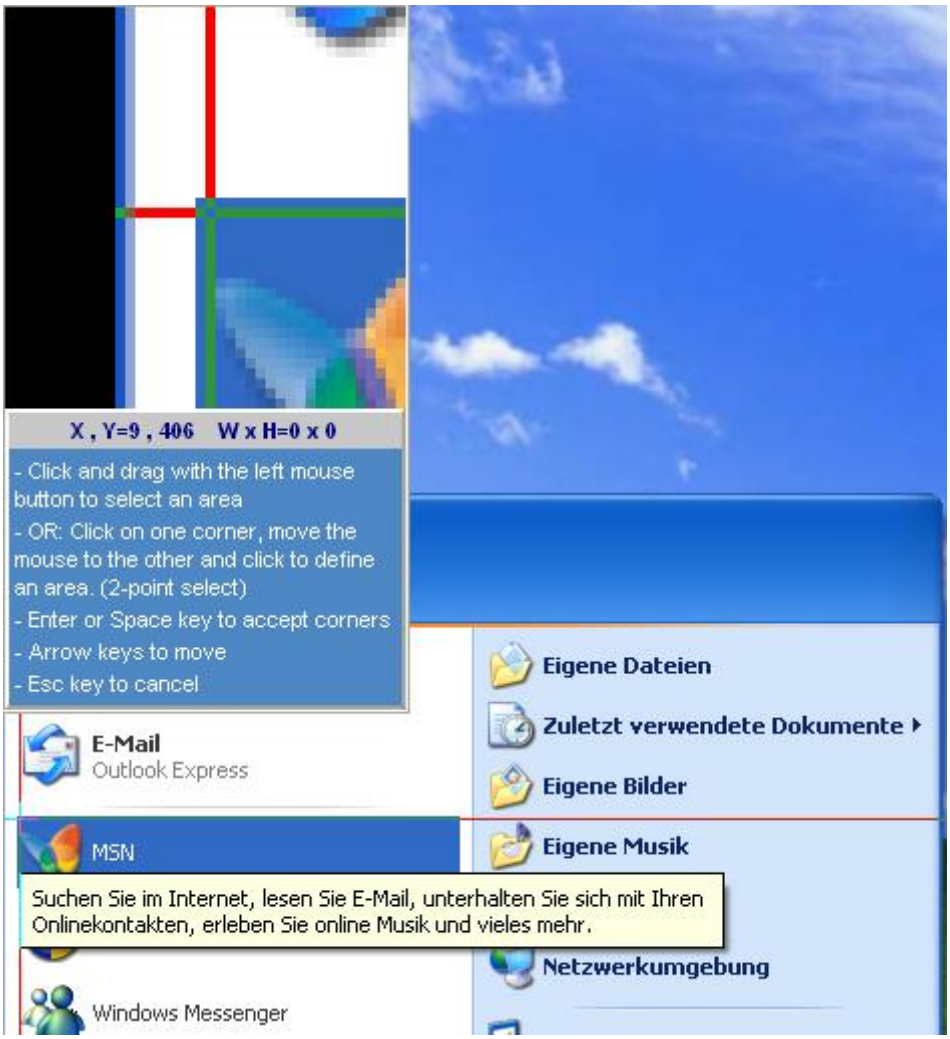

Von den vielen Einstellungsmöglichkeiten sind folgende besonders wichtig, die bei der Arbeit geändert werden sollte.

1: Es sollte ein Haken bei „Include Mouse in Captured Image“ gesetzt werden, damit der Mauszeigern auch in den Bildern erscheint.

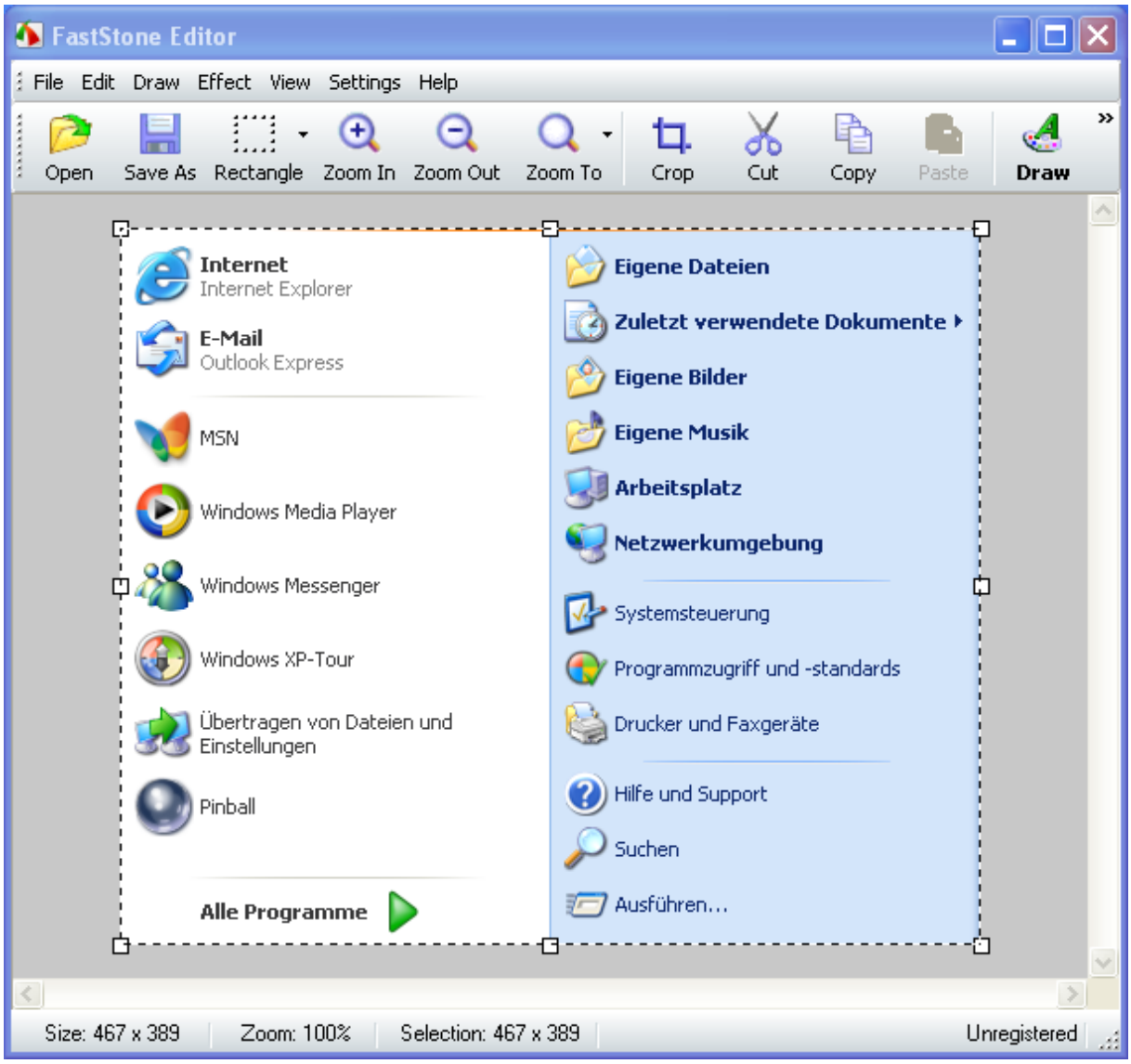
2: Mit „Delay before Capture“ kann angeben werden, wie lange nach dem Drücken eines Foto-Knopfes gewartet wird, bis das Foto gemacht wird. Dies ist sinnvoll, wenn zuerst die Maus in Position gebracht werden soll, wenn z. B. aufgeklappte Menüs fotografiert werden sollen. Erfahrungsgemäß ist ein Wert von 4 Sekunden sinnvoll. Alle Einstellungen müssen über den „OK“-Knopf rechts-unten übernommen werden.




Soll dann ein Teilbereich des Bildschrims fotografiert werden, wird auf einen der Knöpfe gedrückt, dabei ist meist der rechteckige Ausschnitt sehr sinnvoll. Danach verschwindet das „FastStone Editor“-Fenster und die zu fotografierende Situation muss zügig auf den Bildschirm gebracht werden. Bei einem Rechteck-Bild kann dann mit zwei einfachen Mausklicks die linke obere und die recht untere Ecke bestimmt werden. Dies soll im Beispiel ein einfaches Stück des aufgeklappten Start-Menü in Windows XP sein. Es ist im folgenden Bild erkennbar, dass eine zum gehörende automatisch sich öffnende Lupe genutzt wird, damit Pixel-genau bestimmt werden kann, wo die Eckpunkte des gewünschten Ausschnitts liegen.




Nachdem beide Punkte gewählt sind, öffnet sich ein einfacher graphischer Editor, der im Wesentlichen selbsterklärend ist. Meist soll einfach das Foto in ein Dokument kopiert werden, dazu genügt der klassische „Guttenberg-Griff“, mit Strg+A alles markieren, mit Strg+C das Bild kopieren und mit Strg+V im gewünschten Dokument einfügen. Das Bild kann so natürlich auch in komportablere Bildbearbeitungsprogramme kopiert werden.


FastStone Editor
File Edit Draw Effect View Settings Help
Open Save As Rectangle Zoom In Zoom Out Zoom To Crop Cut Copy Paste Draw
Internet
Internet Explorer
E-Mail
Outlook Express
MSN
Windows Media Player
Windows Messenger
Windows XP-Tour
Übertragen von Dateien und Einstellungen
Pinball
Alle Programme
Eigene Dateien
Zuletzt verwendete Dokumente
Eigene Bilder
Eigene Musik
Arbeitsplatz
Netzwerkumgebung
Systemsteuerung
Programmzugriff und -standards
Drucker und Faxgeräte
Hilfe und Support
Suchen
Ausführen...
Size: 467 x 389
Zoom: 100%
Selection: 467 x 389
Unregistered