Einführung in die Informatik
– Teil 1 –
Java-Editor und Applets

Inhalt:

1 Worum es geht
2 Die benötigten Programme
2.1 Installation des Java Development Kit (JDK)
2.2 Installation des Java-Editors
3 Erstellen eines Applets
3.1 Erstellen eines Applets mit dem Java-Editor
3.2 Mögliche Fehlerquellen
3.3 Textfelder in Applets
3.4 Grafik in Applets
4 Tipps und Tricks
4.1 Hinweise zum Java-Editor
4.1.1 Debuggen mit dem Java-Editor
4.1.2 Import von Java-Quelltexten mit dem Java-Editor
4.1.3 Weitere Hinweise zum Java-Editor
4.2 Hinweise zu Java
4.2.1 Datentypen
4.2.2 Type-Casting
4.2.3 Zeichenketten-Operationen
4.2.4 Label – TextField – TextArea
4.2.5 Zufallszahlen
4.2.6 Suchen nach Begriffen in Quelltexten
5 Wenn Du es genau wissen willst...

1 Worum es geht

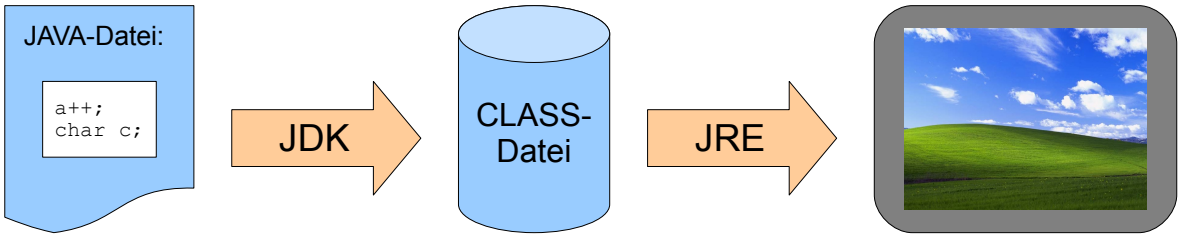
Du hast Dich entschlossen, Java zu lernen, und möchtest das Programmieren auch zu Hause ausprobieren? Dann bist Du hier richtig!
In diesem Dokument erfährst Du, wie Du Dir eine eigene Programmier-Umgebung zu Hause installieren kannst, wie Du im Wesentlichen damit umgehst und welche grundsätzlichen Tricks Du kennen solltest. Damit der Text für Anfänger und Fortgeschrittene geeignet ist, habe ich an einigen Stellen sogenannte Endnoten verwendet, also kleine hochgestellte Ziffern. Die zugehörigen Informationen stehen dann am Ende des Dokuments im Kapitel „5 Wenn Du es genau wissen willst“.[1]

Bevor wir richtig loslegen aber noch ein wichtiger Hinweis: Informatik und Programmieren sind nicht das Gleiche! Eigene Programme können helfen, das zu verstehen und auszuprobieren, worum es eigentlich in der Informatik geht. Insofern sind Programmierkenntnisse hilfreich, aber eben nicht alles. Dieses Dokument ist daher keine Informatik-Schulung, sondern soll die Probleme verringern, die beim Erstellen von Java-Programmen häufig auftreten.

Dieses Dokument behandelt Computer mit Windows-Betriebssystem. Apple- oder Linux-Rechner werden nicht gesondert behandelt.[2]

2 Die benötigten Programme

Java ist eine Programmiersprache, die ein Mensch als Text lesen kann. Das gilt jedoch nicht für den Computer. Für den Computer muss das Java-Programm, der sogenannte Quelltext, erst einmal übersetzt werden. In der Fachsprache heißt dieser Vorgang „kompilieren“. Dafür benötigst Du das **JDK (Java Development Kit)**. Im Fall von Java reicht dies jedoch noch nicht aus. Damit der Computer das Programm tatsächlich ausführen kann, wird ein weiteres Programm benötigt, nämlich die **JRE (Java Runtime Environment)**. Der ganze Vorgang sieht dann folgendermaßen aus:



Das Programm JRE ist in der Regel bereits auf einem Computer installiert. (Sonst würden einige Anwendungen im Internet-Browser nicht funktionieren.) Das Programm JDK ist in der Regel noch nicht installiert.
Am besten ist es, sich unabhängig von den tatsächlich installierten Java-Programmen auf dem eigenen Computer einfach die neueste Version vom JDK aus dem Internet herunterzuladen und zu installieren (siehe Kapitel „2.1 Installation des Java Development Kit (JDK)“). Dadurch wird auch gleich die JRE auf den neuesten Stand gebracht.

Im Prinzip würde diese Installation schon reichen, um einfache Java-Programme zu schreiben und ablaufen zu lassen. Viel bequemer ist es jedoch, wenn man ein Zusatzprogramm hat, mit dem man komfortabel den Quelltext erstellen kann, und das automatisch die Programme JDK und JRE aufruft. Ein solches Programm heißt IDE (Integrated Development Environment).

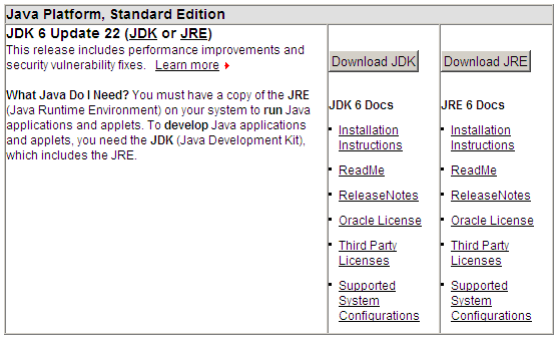
Es gibt viele, sehr unterschiedliche IDE-Programme für das Schreiben von Java-Programmen; für die Schule hat sich vor allem der Java-Editor bewährt.

Vielleicht kann Dir Deine Lehrerin bzw. Dein Lehrer die Installationsdateien auch auf Deinem USB-Stick zur Verfügung stellen, dann bist Du sicher, dass Du die richtigen Dateien und Programmversionen hast.

Grundsätzlich gilt für alle im Folgenden aufgeführten Installationen, dass diese mit Administrator-Rechten durchgeführt werden müssen.

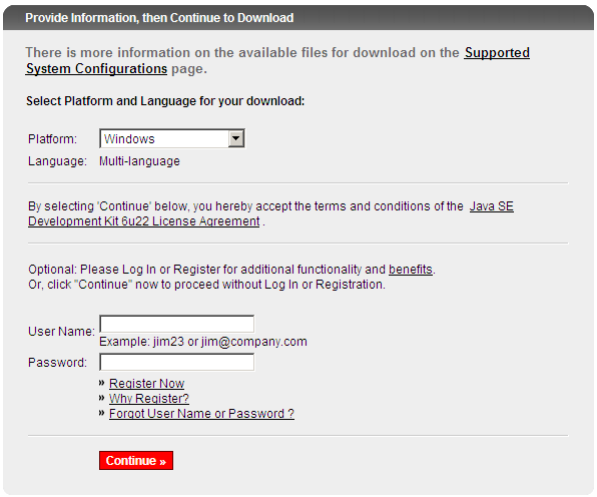
2.1 Installation des Java Development Kit (JDK)

Bevor irgendeine IDE installiert wird, muss unbedingt das Java Development Kit (JDK) installiert werden. Es wird von der Firma Sun Microsystems kostenlos zur Verfügung gestellt und kann über den Link http://www.oracle.com/technetwork/java/javase/downloads/index.html downgeloadet werden. Auf dieser Website stehen verschiedene Versionen des JDK zum Download bereit (siehe Abbildung), es reicht die Version ohne „Bundles“. Es muss aber unbedingt das JDK und nicht die JRE sein.
Wie Du an der Abbildung sehen kannst, war zum Zeitpunkt der Erstellung dieses Dokuments die Version JDK 6 Update 22 aktuell. Die Version 7 stand bereits „vor der Tür“, so dass Du vermutlich eine andere Darstellung erhalten wirst.



Wenn Du auf den Download-Button klickst, musst Du Dein Betriebssystem auswählen und auf „Continue“ klicken.
Im Folgebildschirm klickst Du auf die angebotene Datei.

Java SE Development Kit 6u22



Die Installation wird durch Aufrufen der heruntergeladenen Datei gestartet. Dabei kann einfach den Anweisungen gefolgt werden.

2.2 Installation des Java-Editors

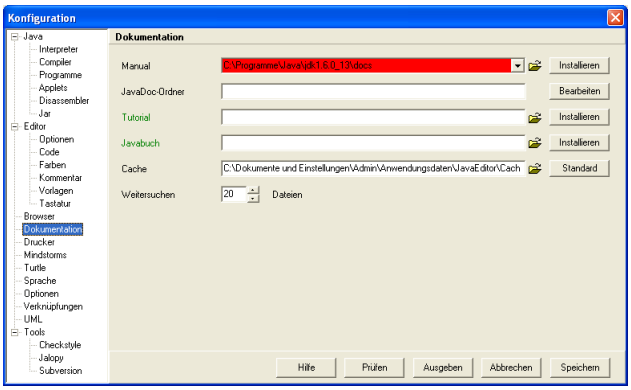
Die aktuelle Version des Java-Editos kannst Du – selbstverständlich kostenlos – über den Link http://javaeditor.org/index.php?title=Download herunterladen. Da Du das Programm zu Hause auf einem einzelnen PC installierst, solltest Du die „Personal Version“ wählen. Zum Zeitpunkt der Erstellung dieses Dokuments war die Version 9.15f aktuell. Die abgebildeten Screenshots beziehen sich auf diese Version.

Bei der Installation erkennt der Java-Editor automatisch das installierte JDK. Im Folgenden sind noch weitere Konfigurationseinstellungen für den Java-Editor erforderlich bzw. sinnvoll[3]:

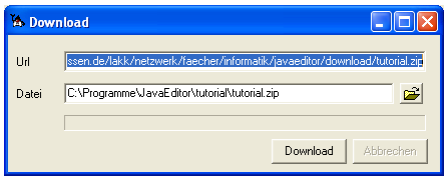
Java-Editor einrichten

Für die bessere Verwendung des Java-Editors solltest Du Dir unbedingt einige Erweiterungen installieren[4]. Dazu musst Du das Programm zunächst als Administrator ausführen. Unter windows XP genügt die Anmeldung als Administrator, unter Windows Vista und Windows 7 musst Du das Programm-Icon zusätzlich mit der rechten Maustaste anklicken und „Als Administrator ausführen“ auswählen. Außerdem muss für das weitere Vorgehen eine Internet-Verbindung bestehen.

Nach dem Start des Programms wählst Du „Fenster | Konfiguration“ oder klickst auf die Schaltfläche und klickst dann auf „Dokumentation“. Es öffnet sich nebenstehendes Fenster.



Zuerst installieren wir das Tutorial. Dazu klickst Du auf den Button „Installieren“ hinter „Tutorial“. Es öffnet sich ein weiteres Fenster, bei dem Du auf „Download“ klickst:



Nach Abschluss des Downloads klickst Du rechts auf „OK“, so dass die Installation abgeschlossen wird.

Entsprechend verfährst Du beim „Javabuch“[5], bei dem automatisch nacheinander zwei Dateien heruntergeladen und installiert werden.

Die Dokumentation installierst Du über den Eintrag „Manual“. Wenn Du nur über eine langsame Internet-Verbindung (weniger als DSL2000) verfügst, musst Du vor dem Anklicken von „Download“ bei „Url“ den Link „http://javadoc.allimant.org/dist/j2se6.zip“ von Hand eingeben.[6] Der Rest der Installation läuft wie bei den anderen Erweiterungen.

Zum Abschluss „Speichern“ anklicken nicht vergessen!

(Eine weitere Konfigurationseinstellung wird in Endnote [7] behandelt.)

Damit Du den Überblick behältst, solltest Du Dir unbedingt ein Verzeichnis anlegen, in dem Du Deine Java-Editor-Projekte ablegen kannst, bei mir z. B. „D:\Computer\VLIN\Java-Editor-Projekte“.

Außerdem empfiehlt es sich, den Windows-Explorer so einzustellen, dass Dateierweiterungen grundsätzlich angezeigt werden.[8]
Dazu wählst Du im Windows-Explorer (oder im Arbeitsplatz) unter Windows-XP „Extras | Ordneroptionen...“ bzw. unter Windows Vista „Organisieren | Ordner- und Suchoptionen“, anschließend das Register „Ansicht“ und entfernst den Haken bei „Erweiterungen für bekannte Dateitypen ausblenden“.

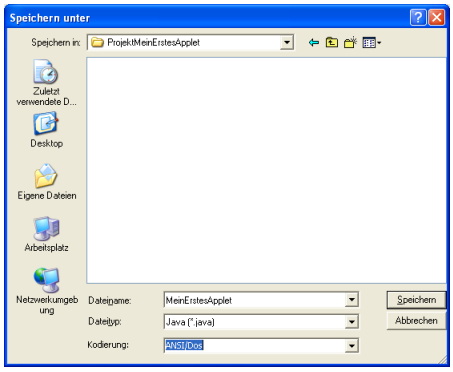
Wie Du das erste Programm mit dem Java-Editor erstellen kannst, erfährst Du in Kapitel „3.1 Erstellen eines Applets mit dem Java-Editor“.

3 Erstellen eines Applets

Ein Applet ist ein kleines Java-Programm, das in einem Browser angezeigt werden kann. Das ist zwar nicht mehr so modern, führt aber schnell zu Ergebnissen – und daran sind wir zunächst interessiert.
Auf weitere Möglichkeiten des Java-Editors wird in diesem Dokument nicht eingegangen.[9]

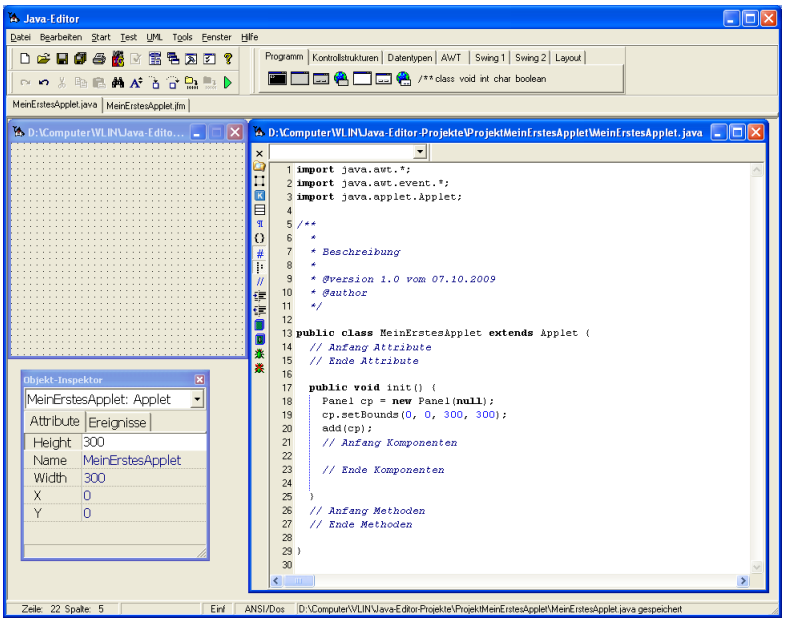
3.1 Erstellen eines Applets mit dem Java-Editor

Ein Applet erstellst Du, indem Du im Register „Programm“ auf die Schaltfläche  (die Weltkugel mit der *grauen* Fläche) klickst. Es öffnet sich eine Abfrage zum Speichern der zugehörigen Dateien dieses Projekts. Als Verzeichnis wählst Du das von Dir in Kapitel 2.2 erstellte Verzeichnis für die Java-Editor-Projekte aus, bei mir also „D:\Computer\VLIN\Java-Editor-Projekte“.



Auch wenn der Java-Editor alle Dateien mit dem gleichen Anfang speichert, solltest Du zuerst mithilfe der Schaltfläche  ein eigenes Verzeichnis für dieses Projekt anlegen, damit nicht alle Projekte in einem Verzeichnis stehen. Ich habe den Namen „ProjektMeinErstesApplet“ gewählt.[10]
Die Datei für das Applet bekommt von mir den Namen „MeinErstesApplet“. Der Java-Editor ergänzt automatisch die Dateierweiterung „.java“ und erstellt eine Formular-Datei mit dem Namen „MeinErstesApplet.jfm“.

Der Bildschirm sieht jetzt wie abgebildet aus.



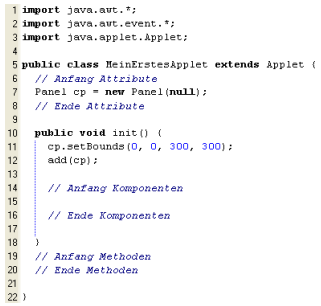
Unten links siehst Du den Objekt-Inspektor, den Du mithilfe des Kreuzes oben rechts schließen und jederzeit über „Fenster | Objekt-Inspektor ein/aus“ oder über die Schaltfläche  wieder anzeigen lassen kannst. Allerdings benötigen wir den Objekt-Inspektor weiter unten noch einmal, so dass Du ihn zunächst geöffnet lassen solltest.

Über dem Objekt-Inspektor befindet sich das Fenster für das eigentliche Applet. Die Größe des Applets kannst Du mit der Maus durch Ziehen am Rahmen unten rechts verändern, die Größe von 300×300 Bildschirmpunkten ist aber für den Anfang in Ordnung.[11]
Wenn Du das Applet-Fenster versehentlich geschlossen haben solltest oder wenn Du beim Öffnen zuerst die JAVA-Datei aufgerufen hast, kannst Du es durch Anklicken der Schaltfläche  im rechten Quelltext-Fenster wieder aufrufen.

Das wichtigste Fenster ist das für den Quelltext. Im Quelltext siehst Du eine senkrechte Linie, eine sogenannte Strukturlinie, die Dir grafisch anzeigt, welche Teile des Quelltextes eine

strukturelle Einheit bilden. Du kannst Diese Strukturlinien mithilfe der Schaltfläche  aus- und wieder einschalten.

Zuerst wollen wir den vom Java-Editor angelegten Quelltext etwas „verschlanken“: Kommentare, die nur der Erläuterung des Quelltextes dienen, sind dunkelblau und kursiv dargestellt. Sie beginnen mit „`/*`“[12] und enden mit „`*/`“. Den Rest einer einzelnen Zeile kannst Du mithilfe von „`//`“ auskommentieren. Da wir die ersten Kommentarzeilen nicht benötigen, kannst Du den ersten Kommentar bis zum „`public class`“ löschen.[13]



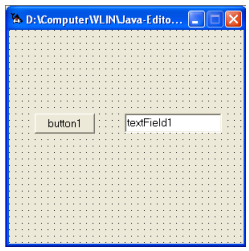
Außerdem ist es für das Folgende etwas unpraktisch, dass der Java-Editor das verwendete Panel[14] nur innerhalb des init-Methode definiert hat. Deshalb solltest Du die Zeile

```
Panel cp = new Panel(null);
```

zwischen die beiden Kommentar-Zeilen mit den Attributen verschieben.[15] Dazu erzeugst Du zunächst mithilfe von „Enter“ eine leere Zeile zwischen den bisherigen Zeilen 6 und 7 und verschiebst dann die genannte Zeile hierhin.[16] Der Quelltext sollte jetzt wie abgebildet aussehen.

Übrigens: Hinter dem Dateinamen des Quelltextes im Register steht jetzt ein Kreuz,
das anzeigt, dass irgendetwas am Quelltext verändert wurde. Damit diese Veränderungen nicht durch einen Computerabsturz o. ä. zunichte gemacht werden, solltest Du Deine Datei schleunigst speichern: „Datei | Alle speichern“ oder Anklicken der Schaltfläche  .

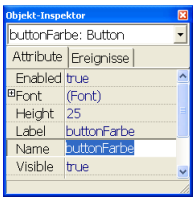
Klicke jetzt oben auf die Registerkarte „AWT“[17], dort auf den Button und anschließend in das Feld des Applets: Ein Button erscheint auf dem Applet. Verfahre ebenso mit einem TextField, das Du rechts daneben setzen kannst.



Im Quelltext hat sich parallel zu unseren Klicks einiges getan:

- Zu Anfang des Quelltextes (bei den Attributen) wurden die beiden Objekte eingeführt. Hier kannst Du jetzt auch gut sehen, wie der Java-Editor die automatisch gewählten Namen für die Objekte vergibt: Im Prinzip wird die Bezeichnung des Objekts übernommen, wobei der erste Buchstabe kleingeschrieben wird[18] und hinten eine fortlaufende Ziffer angehängt wird: Der nächste Button bekäme also den Namen „button2“.
  Wichtig sind die Namen der Objekte für den Programmtext, den wir uns gleich genauer ansehen werden.
  Der Name eines Objekts hat aber nichts mit dem zu tun, wie das Objekt beschriftet ist. (Die Beschriftung bekommt ein Anwender bei der Ausführung des Programms zu sehen.) Eine Ernie-Handpuppe mit einem Pullover, auf dem „Bert“ steht, ist immer noch eine Ernie-Handpuppe und wird nicht durch einen falschen Pullover zu einer Bert-Handpuppe.
  Die Standardbeschriftung der eingefügten Objekte stimmt zunächst mit dem Namen des jeweiligen Objekts überein. Wie Du die Beschriftung und weitere Eigenschaften der Objekte ändern kannst, klären wir weiter unten. An dieser Stelle aber noch einmal (weil es häufig zu Fehlern führt): Der Name eines Objekts ist für den Programmtext von Bedeutung, die Beschriftung des Objekts nur für den Anwender, der die Beschriftung liest!
- In der init-Methode werden alle Eigenschaften der Objekte festgelegt, also wo sie sich befinden und wie sie beschriftet sind.
- Im unteren Bereich hat der Java-Editor gleich die sogenannte Standard-Methode für den Button ergänzt, die aufgerufen wird, wenn der Button angeklickt wird. Sie heißt „button1_ActionPerformed“. Diese werden wir weiter unten noch benutzen.

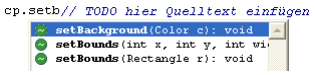
Namen von Objekten sollten so gewählt sein, dass Du im Quelltext schnell erkennst, welches Objekt mit welcher Funktion gerade gemeint ist. Deswegen solltest Du den Button umbenennen, da wir mit diesem gleich die Farbe des Applet-Hintergrundes ändern werden. Für das Umbenennen klickst Du im Applet-Fenster den Button an[19] und änderst im Objekt-Inspektor die Eigenschaft „Name“ auf den neuen Namen „buttonFarbe“. („Enter“ drücken!) Im Quelltext hat der Java-Editor jetzt automatisch alle Einträge, in denen bisher „button1“ stand, in „buttonFarbe“ geändert.



Die Beschriftung (Eigenschaft „Label“) wurde gleich mit geändert, weil sie mit dem bisherigen Namen identisch war. Trage in das zugehörige Feld als neue Bezeichnung „mache rot!“ ein.[20] Der Button ist jetzt gerade noch groß genug für diese lange Beschriftung, so dass Du ihn durch Ziehen mit der Maus vergrößern solltest.

Jetzt soll endlich ein tatsächliches Programm verfasst werden: Klicke dazu doppelt auf den Button. Der Java-Editor springt im Quelltext gleich an die Stelle, an der der gewünschte Befehl eingegeben werden kann. Die fertige Methode soll folgendermaßen aussehen:

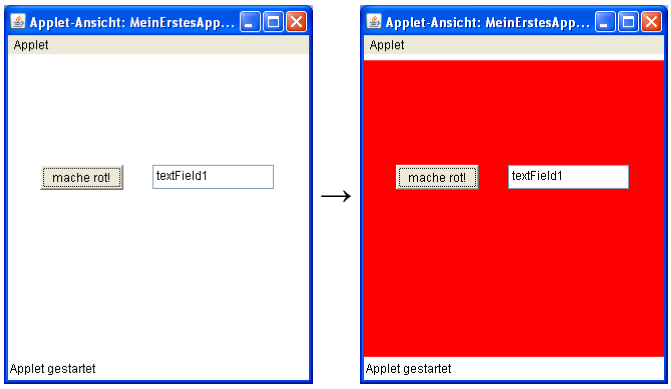
```
public void buttonFarbe_ActionPerformed(ActionEvent evt) {
  cp.setBackground(Color.red);
}
```

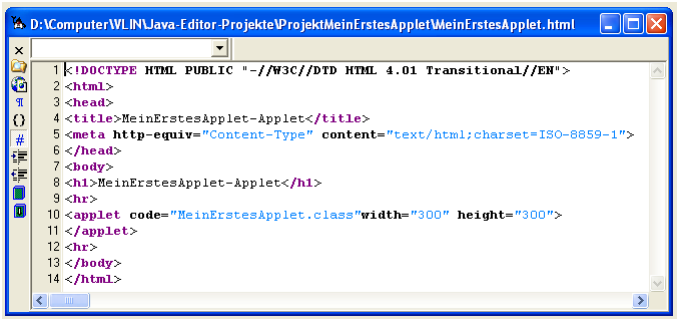


Sobald Du hinter „`cp`“ den Punkt tippst, öffnet sich automatisch die sogenannte Code-Vervollständigung. Es werden Dir alle möglichen Methoden, Eigenschaften und Objekte angeboten, die zu „cp“ passen. Sobald Du „`setb`“ tippst, wird der Eintrag „`setBackground`“ hervorgehoben, den Du jetzt mithilfe von „Enter“ automatisch in den Quelltext übernehmen kannst. Analog kannst Du hinter „`Color.`“ den Eintrag „`red`“ auswählen. Wenn Du die Code-Vervollständigung schließen möchtest, musst Du „Esc“ drücken.

Das „cp“ steht für das auf dem Applet befindliche Panel, auf dem der Button und das Textfeld liegen. Der Punkt hinter einem Objekt oder hinter einer Variablen gibt an, dass eine Methode aufgerufen oder eine Eigenschaft verwendet wird.

Wenn Du das Applet ausführen möchtest, hast Du mehrere Möglichkeiten: „Start | Starte Applet“ oder „F9“ oder die Schaltfläche . Nach dem Anklicken des Buttons ändert sich die Farbe des Applets wie gewünscht in Rot:



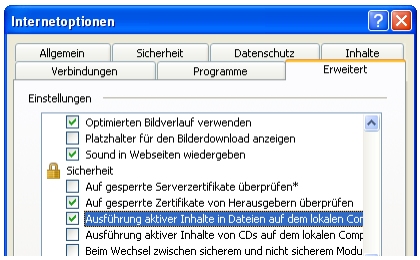


Neben dem Applet im Appletviewer hat sich ein weiteres Fenster geöffnet, das die zugehörige HTML-Datei zeigt. Die Zeile mit dem „`<applet>`“-Tag enthält die Abmessungen des Applets, die wir oben gewählt bzw. beibehalten haben. Wenn Du die Größe des Applets veränderst, werden automatisch die Werte in der HTML-Datei angepasst.
Das Fenster mit der HTML-Datei kannst Du schließen, damit Du nicht versehentlich durch Anklicken von „Start“ einen Browser aufrufst, der die HTML-Datei darstellen soll.[21] Auf der anderen Seite hast Du die Möglichkeit, aus dem HTML-Fenster heraus das Applet im Browser betrachten zu können, indem Du „Start“ anklickst.

Wenn Du das Applet ohne den Quelltext weitergeben möchtest, musst Du die HTML-Datei und die CLASS-Dateien weitergeben, die in Deinem Projekt-Verzeichnis stehen.

Alternativ zum Appletviewer kannst Du Dir das Applet auch ansehen, wenn Du im Windows-Explorer auf die HTML-Datei in Deinem Projekt-Verzeichnis doppelklickst. Das Applet wird dann in einem Browser geöffnet.

Beim Einsatz des Internet-Explorers von Microsoft könnte es dabei Probleme geben, weil Skripte etc. aufgrund der Sicherheitseinstellung nicht ausgeführt werden können. Es erscheint in diesem Fall oben eine gelbe Leiste mit folgendem Inhalt: „Das Ausführen von Skripts bzw. ActiveX-Steuerelementen, die auf den Computer zugreifen könnten, wurde für diese Webseite aus Sicherheitsgründen eingeschränkt. Klicken Sie hier, um weitere Optionen anzeigen zu lassen...“ Wenn Du auf die Leiste klickst, kannst Du über „Geblockte Inhalte zulassen...“ und einen weiteren Klick auf „Ok“ das Applet endlich anzeigen lassen. Diese Prozedur muss nach einem Neustart des Internet-Explorers jedes Mal erneut durchgeführt werden, und das nervt doch extrem.



Dauerhaft kannst Du es abschalten, wenn Du im Internet-Explorer auf „Extras | Internetoptionen“ klickst und dann im Register „Erweitert“ einen Haken bei „Ausführung aktiver Inhalte in Dateien auf dem lokalen Computer zulassen“ setzt.

Wenn Du das Applet ohne den Quelltext weitergeben möchtest, musst Du die HTML-Datei aus dem Unter-Verzeichnis „build“ und das Unter-Verzeichnis „classes“ weitergeben.[22]

3.2 Mögliche Fehlerquellen

Wenn ein Programm nicht starten will und der Übersetzungsvorgang mit einer Fehlermeldung beendet wird (Details dazu im Kapitel „4. Tipps und Tricks“), liegt das in der Regel daran, dass ein sogenannter Syntax-Fehler, also ein Verstoß gegen die korrekte Schreibweise des Programmtextes, vorliegt. Für eine schnelle Analyse könnte folgende Liste der typischen Fehler helfen:

- Eine Anweisung schließt nicht mit einem Semikolon ab.
- Objekte oder Methoden haben eine falsche Groß- bzw. Kleinschreibung (z. B. „`TextfIEld`“ anstelle von „`textField`“).
- Die Klammerstruktur ist falsch (Klammer fehlt z. B. häufig bei geschweiften Klammern).

- In einer Parameterliste, also dem eingeklammerten Bereich hinter einer Methodenbezeichnung, steht ein Wert zu viel oder zu wenig (z. B. „`setBackground(Color.red, Color.blue)`“). Dieser Fehler sollte bei Verwendung der Code-Vervollständigung eigentlich nicht auftreten.
- Es werden falsche Variablentypen miteinander verbunden (z. B. „`int c = "a";`“).
- Ein Befehl bzw. eine Methode ist unbekannt, weil die Mutterklasse nicht eingebunden wurde. Das Objekt „FileReader“ steht z. B. nur dann zur Verfügung, wenn am Anfang des JAVA-Dokuments die Zeile „`import java.io.*;`“ steht.

Wenn sich die Code-Vervollständigung nach dem Tippen eines Punktes nicht öffnet, liegt dies meistens daran, dass das Wort bzw. Objekt davor falsch geschrieben wurde oder unbekannt ist.

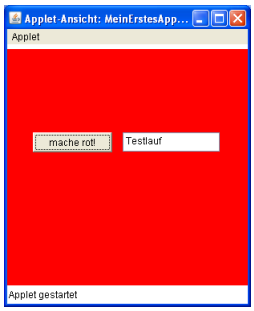
Von den Syntax-Fehlern zu unterscheiden sind die logischen Fehler, also ein (syntaktisch) korrekt geschriebenes Programm, das übersetzt werden kann, aber nicht das tut, was es eigentlich tun sollte. Solche Fehler kann man mithilfe des Debuggens finden (siehe Kapitel „4. Tipps und Tricks“).

3.3 Textfelder in Applets

Nun wollen wir auch noch etwas Sinnvolles mit dem eingefügten Textfeld anfangen. Verändere dazu zunächst analog zur Änderung der Beschriftung des Buttons (siehe oben) im Entwurfsmodus die Eigenschaft „text“ vom Textfeld auf „Test“ und verbreitere das Textfeld etwas, so dass auch längere Wörter hineinpassen.
Klicke nun wieder auf den Button doppelt, so dass Du wieder im Quelltext an die richtige Stelle kommst, an der der beim Anklicken des Buttons auszuführende Quelltext steht. Ergänze hinter der ersten Zeile mit der `setBackground`-Anweisung Folgendes:

```
String eintrag;
eintrag = textField1.getText();
eintrag = eintrag + "lauf";
textField1.setText(eintrag);
```

Achte bei der Eingabe darauf, dass „`textField`“ ein großgeschriebenes „F“ hat.



Wenn Du das Applet jetzt ausführst und auf den Button klickst, wird an das vorhandene „Test“ im Textfeld ein „lauf“ angehängt. Bei jedem weiteren Klicken wird ein weiteres „lauf“ angehängt.

Die Methode „`getText`“ liest also den Inhalt eines Textfeldes aus, die Methode „`setText`“ schreibt einen neuen Inhalt in das Textfeld hinein.
Weitere Informationen werden im Kapitel „4.3.4 Label – TextField – TextArea“ gegeben.

3.4 Grafik in Applets

Abschließend sollst Du noch erfahren, wie Du einfache Grafiken im Applet selbst erzeugen kannst. Nimm dazu bitte folgende Veränderungen an Deinem Applet vor (oder gestalte entsprechend ein neues Applet):

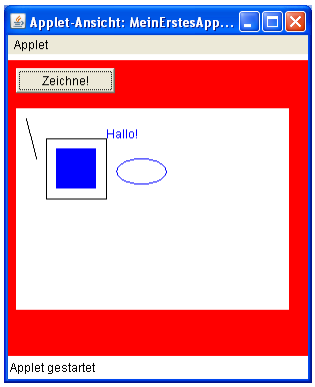
- Das Textfeld kannst Du löschen, indem Du es anklickst und „Entf“ drückst.
  Die von Dir selbst geschriebenen Zeilen in der ActionPerformed-Methode des Buttons, in denen das Textfeld vorkommt, werden vom Java-Editor nicht automatisch gelöscht, das musst Du selbst machen. Beim Kompilieren würdest Du allerdings darauf hingewiesen, dass hier ein Objekt angesprochen werden soll, das nicht (mehr) existiert.

- Den Button kannst Du nach oben links verschieben und ihm die Beschriftung „Zeichne!“ geben.
- Unterhalb des Buttons platzierst Du ein neues Objekt und zwar eine Canvas. Der Hintergrund (Eigenschaft „background“) der Canvas soll weiß sein, die Canvas soll fast den gesamten Bereich des Applets ausfüllen. (Ggf. kannst Du die Abmessungen des Applets vergrößern, wenn Du noch mehr Platz zum Zeichnen benötigst.)
- Der Hintergrund des Applets soll rot sein, damit Du die Abmessungen der Canvas besser erkennen kannst. Verschiebe dazu die Anweisung „`cp.setBackground(Color.red);`“ aus der ActionPerformed-Methode des Buttons hinter die `setbounds`-Anweisung für das Panel zu Beginn der init-Methode.

Klicke jetzt auf den Button doppelt und lösche ggf. noch verbliebene Zeilen in der ActionPerformed-Methode des Buttons. Gib dann folgenden Programmtext ein:

```
Graphics g=canvas1.getGraphics();
g.drawLine(10,10,20,50);
g.drawRect(30,30,60,60);
g.setColor(Color.blue);
g.fillRect(40,40,40,40);
g.drawString("Hallo!",90,30);
g.drawOval(100,50,50,25);
g.dispose();
```

Wenn Du das Programm ablaufen lässt, ergibt sich nebenstehender Bildschirm.



Alle Zeichenfunktionen werden in Java auf einem sogenannten Grafik-Kontext (Objekt „Graphics“) ausgeführt, der zu Beginn einer Zeichenoperation zugewiesen werden muss. In unserem Fall ist das der Grafik-Kontext unserer Canvas: `canvas1.getGraphics()`
Alle Zeichenoperationen sind dann Methoden dieses Grafik-Kontextes „`g`“.

Bei der Verwendung von Koordinaten bei Zeichenoperationen muss Du die nebenstehend abgebildete Orientierung des Koordinatensystem beachten: Der Ursprung eines Grafik-Kontextes liegt immer oben links. Die *y*-Koordinaten nehmen dann nach unten zu.

Weitere Informationen zur Verwendung von Grafik-Befehlen erhältst Du von Deiner Lehrerin bzw. Deinem Lehrer oder aus Büchern bzw. Quellen im Internet.[23]

Ein wichtiger Hinweis aber noch zum Schluss: Wenn eine Zeichnung in einem Applet erstellt wurde und das Applet hinterher z. B. von einem anderen Fenster überdeckt wurde, ist die Zeichnung „verschwunden“.

4 Tipps und Tricks

Beim Einsatz einer IDE treten vor allem folgende Routine-Aufgaben auf, zu denen Du noch einige Hinweise erhalten sollst:

- Finden von logischen Fehlern in einem Programm, das sogenannte „Debuggen“,
- Importieren von Java-Quelltexten, die als Datei oder Ausdruck vorliegen,
- Verwenden von Features aus der IDE-Bedienung[24].

Die Hinweise zu diesen Aspekten sind im Kapitel „4.1 Hinweise zum Java-Editor“ aufgelistet. Anschließend werden in Kapitel „4.2 Hinweise zu Java“ noch einige allgemeine Hinweise gegeben.

Zum Debuggen und zum Import von Java-Quelltexten können jedoch an dieser Stelle schon Hinweise gegeben werden.

Debuggen – ohne Verwendung der IDE-Möglichkeiten

Wenn Dein Programm nicht so funktioniert, wie es eigentlich beabsichtigt war, liegt es in vielen Fällen daran, dass Du an einer bestimmten Stelle im Programm davon ausgegangen warst, dass eine bestimmte Variable einen bestimmten Wert hat, was aber tatsächlich nicht der Fall ist. Es wäre also gut, wenn Du Dir einen Überblick darüber verschaffen könntest, welchen Wert eine bestimmte Variable hat bzw. ob eine bestimmte Stelle des Programms überhaupt durchlaufen wird. Hierfür stellt der Java-Editor ausgefeilte Werkzeuge zur Verfügung, die in dem folgenden Kapitel erläutert werden.
Oftmals genügt aber einfach die Anzeige eines Variablen-Werts. Dazu gibt es die Anweisung „`System.out.println(n);`“, wobei die Variable „n“ sowohl für eine Zeichenkette als auch für eine Zahlen-Variable stehen kann. Der Wert der Variablen oder der Zeichenkette wird dann im unteren Bereich des Bildschirms in einem Status-Fenster angezeigt.

Importieren von Java-Quelltexten

Die Verwendung der grafischen Oberfläche zum Erstellen eines Applets („Zusammenklicken der Objekte“) kann man nur dann nutzen, wenn das Projekt von vornherein im Java-Editor erstellt wurde. Bei einem Import eines Java-Quelltextes, der in den folgenden Kapiteln beschrieben wird, ist es anschließend grundsätzlich nicht möglich, Objekte nachträglich automatisch durch Anklicken hinzuzufügen.
Wenn Dir dies zu nachteilig erscheint, musst Du folgendermaßen vorgehen:

- Zuerst erstellst Du eine neues, leeres Projekt.
- Entnimm dem Quelltext, welche Objekte verwendet werden, und setze sie selbst auf die Applet-Fläche. Gib den Objekten die Eigenschaften, die sie auch im Quelltext haben – insbesondere die Namen der Objekte.
- Entnimm dem Quelltext, welche Events den Objekten zugeordnet wurden, und erstelle entsprechende Methoden in Deinem Applet, z. B. die ActionPerformed-Methode eines Buttons.
- Abschließend kannst Du den Inhalt der einzelnen Event-Methoden und die anderen Methoden aus der Quelltext-Vorlage in Deinen Programmtext hineinkopieren.

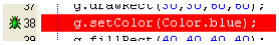
Dieses Vorgehen verlangt ein bisschen „Fingerspitzengefühl“, so dass Du mit einfachen Projekten beginnen solltest, bevor Du Dich an größere Projekte wagst.

Das direkte Ausführen von Java-Quelltexten ist (nach dem Kompilieren) allerdings nur in folgenden Fällen möglich, so dass nur in diesen Fällen ein Import sinnvoll ist:

- Es handelt sich um ein Applet. Die JAVA-Datei enthält dann folgende Zeile:
  `public class NN extends Applet ...,`
  wobei „NN“ für einen beliebigen Klassennamen steht.
- Es handelt sich um eine Anwendung mit einer Main-Methode. Die JAVA-Datei enthält dann folgende Zeile:
  `public static void main(...`

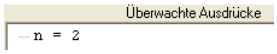
4.1 Hinweise zum Java-Editor

4.1.1 Debuggen mit dem Java-Editor

Für das Debuggen mit dem Java-Editor solltest Du an den betreffenden Stellen im Quelltext sogenannte Haltepunkte setzten.
Ein Haltepunkt bedeutet, dass das Programm während des Ablaufs vor der Ausführung dieser Zeile unterbrochen wird, so dass Du überprüfen kannst, ob das Programm überhaupt bis zu dieser Stelle kommt oder welche Werte die Variablen an dieser Stelle genau haben.
Einen Haltepunkt setzt Du, indem Du mit der Maus auf den grauen 

Bereich links neben der Quelltext-Zeile klickst, an der gestoppt werden soll. (Dort stehen in der Regel die Zeilennummern.) Die betreffende Zeile wird jetzt rot hinterlegt und links ist ein kleiner Käfer zu sehen. Wenn Du den Haltepunkt wieder löschen möchtest, musst Du einfach auf den Käfer klicken.

Häufig ist es sinnvoll, sich den Wert von Variablen etc. an der Stelle des Haltepunkts ausgeben zu lassen. Du findest während des Debuggens alle Variablen im unteren Bereich des Fensters unter „Parameter und lokale Variablen“.
Wenn Du eine übersichtlichere Darstellung für einige ausgewählte Variablen haben möchtest, kannst Du sogenannte Überwachungen einrichten: Wähle „Test | Überwachte Ausdrücke“ und gib anschließend die Variablennamen ein. Bei der Ausführung des Programms erscheint im unteren Bereich des Fensters ein Überwachungsfenster, in dem die aktuellen Werte der überwachten Variablen stehen.



Wenn der Programmablauf an einem Haltepunkt stoppt, wird die aktuelle Quelltext-Zeile blau unterlegt und mit einem roten Pfeil links gekennzeichnet. Du hast dann folgende Möglichkeiten:

- Du setzt den Programmablauf fort: Schaltfläche bzw. „F9“.
- Du setzt den Programmablauf nur eine Zeile fort: Schaltfläche bzw. „F7“.
- Du setzt den Programmablauf vom Haltepunkt bis zur aktuellen Stelle des Cursors fort: Wähle „Test | Bis zum Cursor ausführen“. Den Cursor kannst Du übrigens während des Debuggens an eine andere Stelle setzen.
- Du kannst die Programmausführung beenden: Schaltfläche bzw. „Strg+F2“. Ein Programm, das an einer Quelltext-Zeile angehalten hat, kann übrigens nur über diese Möglichkeit beendet werden.

4.1.2 Import von Java-Quelltexten mit dem Java-Editor

Wenn eine komplette JAVA-Datei vorliegt, ist der Import dieses Quelltextes mit dem Java-Editor denkbar einfach:

- Du erstellst zunächst ein neues Projektverzeichnis für dieses Projekt und kopierst die JAVA-Datei hier hinein.
- Im Java-Editor schließt Du zunächst alle anderen Dateien („Datei | Alle schließen“).
- Dann öffnest Du die hineinkopierte JAVA-Datei mit dem Java-Editor und führst sie aus.
- Da man einer einzelnen JAVA-Datei, die ein Applet definiert, nicht ansehen kann, für welche Abmessung dieses Applet konstruiert wurde, nimmt der Java-Editor zunächst die „Standard-Maße“ 400×300 Bildpunkte. Wenn diese Abmessung nicht stimmen sollte, kannst Du nach dem ersten Start die vom Java-Editor erstellte HTML-Datei (siehe Kapitel „3.1 Erstellen eines Applets mit dem Java-Editor“) bei den Angaben „`width="400" height="300"`“ so ändern, dass der Appletviewer den richtigen Ausschnitt anzeigt.

Wenn Du eine JAVA-Datei aus Einzelstücken (z. B. aus einer PDF-Datei) zusammensetzen musst, gehst Du folgendermaßen vor:

- Du erstellst zunächst ein neues Projektverzeichnis für dieses Projekt.

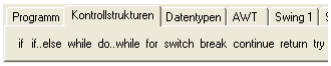
- Im Java-Editor schließt Du zunächst alle anderen Dateien („Datei | Alle schließen“) und öffnest ein leeres JAVA-Dokument („Datei | Neu | Java“ oder kürzer mit „Strg+N“).
- In dieses JAVA-Dokument kannst Du jetzt die einzelnen Programmteile hineinkopieren.
- Das JAVA-Dokument muss in dem neu angelegten Projektverzeichnis unter dem Namen abgespeichert werden, der hinter „`public class`“ steht. Eine Java-Datei mit der Zeile „`public class MeinImport`“... muss also als „MeinImport.java“ gespeichert werden. (Der Java-Editor schlägt den zugehörigen Dateinamen beim ersten Abspeichern vor.)
- Der Java-Editor geht zunächst davon aus, dass es sich bei einer solchen Datei um eine Anwendung handelt. (Im Menü „Start“ heißt es „Starte Applikation F9“.) Wenn Du den Quelltext eines Applets hineinkopiert hast, musst Du vor dem Kompilieren und Starten die Datei einmal schließen und dann wieder öffnen, so dass der Java-Editor erkennt, dass es sich um ein Applet handelt, das im Appletviewer ausgeführt werden soll. (Im Menü „Start“ heißt es jetzt „Starte Applet F9“.)

4.1.3 Weitere Hinweise zum Java-Editor

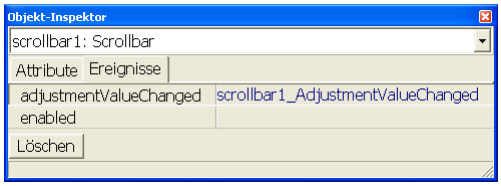
Folgende Hinweise zur Bedienung des Java-Editors könnten für Dich noch hilfreich sein:

- Beim Kopieren und Verschieben von Quelltext stimmt die Formatierung oft nicht, weil zusammengehörige Zeilen nicht untereinander stehen. Wenn Du Zeilen weiter einrücken oder weiter ausrücken möchtest, kannst Du die betreffenden Zeilen markieren und die nebenstehend abgebildeten Schaltflächen verwenden (oder kürzer mit den Tastenkombinationen „Strg+Umschalt+I“ bzw. „Strg+Umschalt+U“).

- Wenn Du bei umfangreichen Projekten in eine bestimmte Programmzeile springen möchtest, kannst Du „Bearbeiten | Gehe zur Zeile“ wählen (oder kürzer mit „Strg+G“) und die gewünschte Zeile eingeben.
- Wenn beim Kompilieren Fehler auftreten, kannst Du im unteren Bereich auf die Zeile doppelklicken. Der Cursor springt dann in die zugehörige Zeile beim Editor.
- Wenn Du eine bestimmte Kontroll- oder Schleifenstruktur (z. B. eine `if-else`-Anweisung) einfügen möchtest, geht dies am einfachsten, wenn Du im Register „Kontrollstrukturen“ auf den entsprechenden Eintrag klickst.



- Der Java-Editor erlaubt Dir, beliebige Shortcuts zu definieren. Dazu solltest Du zuerst die Datei „JEKeyboard.txt“ aus dem Verzeichnis „C:\Programme\JavaEditor\templates“ in ein Verzeichnis Deiner Wahl kopieren. Diese Datei kann jetzt nach Belieben erweitert werden; die Anleitung dazu befindet sich in der Datei selbst. Abschließend muss die Datei noch über „Fenster | Konfiguration | Editor | Tastatur“ bei „Aus Datei“ zugewiesen werden.
- Einige Objekte, z. B. der Scrollbar, reagieren natürlich nicht auf einen Doppelklick. Um die zugehörige Standard-Methode erstellen zu lassen, musst Du im Objekt-Inspektor auf das Register „Ereignisse“ gehen und im Feld rechts neben der gewünschten Methode doppelt klicken oder einmal klicken und den „Neu“-Button anklicken.



Wenn Du Methoden verwenden willst, die der Java-Editor nicht von Haus aus zur Verfügung steht, musst Du die gesamte Implementierung dieser Methode einer anderen Dokumentation entnehmen. Du kannst Dich bezüglich der Struktur aber an der Standard-Methode des Objekts orientieren.

- Wenn Du eine Standard-Methode löschen möchtest, musst Du auf den entsprechenden Eintrag im Objekt-Inspektor klicken und den „Löschen“-Button anklicken. Wenn die Methode nicht komplett leer ist oder nicht nur den Standard-Kommentar enthält, wird sie sicherheitshalber nicht gelöscht. Das müsstest Du dann noch von Hand durchführen oder alternativ vorher den Inhalt der Methode löschen.
- Bei einigen Quelltexten (insbesondere der VLIN) wird davon ausgegangen, dass auf dem Applet kein Panel liegt und dort ein Null-Layout verwendet wird. Wenn Du Dir ein solches Applet im Designer erstellen möchtest, musst Du folgendermaßen vorgehen:
  Zuerst musst Du die drei Zeilen am Anfang der init-Methode löschen oder auskommentieren, in denen das Panel definiert, mit Eigenschaften versehen und auf das Applet gesetzt wird.
  Mithilfe von „Enter“ erzeugst Du ggf. noch eine leere Zeile im oberen Bereich der init-Methode. Klicke auf das Register „Layout“ (oben rechts) und klicke dort auf die Schaltfläche  (AbsoluteLayout). Es wird automatisch die Programm-Zeile „`setLayout(null);`“ ergänzt. Dies bedeutet, dass wir alle Objekte auf dem Applet frei positionieren können.
  Wenn Du jetzt Objekte auf dem Applet platzierst, geht der Java-Editor allerdings immer noch davon aus, dass ein Panel vorhanden ist. Deswegen beginnen alle automatisch generierten Anweisungen für das Platzieren der Objekte in der init-Methode mit „`cp.add(`“. Den Teil „`cp.`“ musst Du in jedem dieser Fälle löschen, damit das Programm korrekt kompiliert werden kann.
- Wenn Du bei umfangreichen Programmen den Überblick über verschiedene Versionen behalten möchtest, musst Du das Tool „Subversion“ installieren. Der Link für den Download steht ganz unten auf http://javaeditor.org/index.php?title=Configuration/de.

Wenn Du ein Projekt weitergeben möchtest, solltest Du das gesamte Projektverzeichnis weitergeben (evtl. in ein ZIP-Archiv packen), damit der Empfänger auch mit dem Designer arbeiten kann und nicht nur auf die JAVA-Datei angewiesen ist.
Beim Java-Editor geht das Packen übrigens am einfachsten durch Aufruf des Befehls „Start | Jar-Datei | Packen“. Es werden dann alle JAVA- und JFM-Dateien des aktuellen Verzeichnisses in eine JAR-Datei gepackt. Damit auch die HTML-Dateien mit eingepackt werden, muss in der Konfiguration („Fenster | Konfiguration“ oder Schaltfläche ) unter „Java | Jar | Packen“ ein „*.html“ ergänzt werden.

4.2 Hinweise zu Java

Die folgenden Anmerkungen sind eine Sammlung von Hinweisen zu Java als Programmiersprache, auf die Du vermutlich häufig zurückgreifen musst. Sie sind hier nur der Vollständigkeit halber aufgelistet und deshalb kürzer gefasst.

Allgemeine Hilfe zu Java kannst Du auch gut im deutschsprachigen Java-Forum unter der URL http://www.java-forum.org/ erhalten.

4.2.1 Datentypen

Folgende Datentypen sind für den Anfang vor allem von Bedeutung:

Datenbereich	*Datentyp*	*Bedeutung / Verwendung*
Wahrheitswerte	`boolean`	mögliche Werte: „true“ und „false“
Zahlen	`long`	Ganzzahl im Bereich von -2^{63} bis $+2^{63}-1$
	`int`	Ganzzahl im Bereich von -2^{31} bis $+2^{31}-1$
	`double`	Dezimalzahl (mit doppelter Genauigkeit); Angabe kann auch in der Form „`3e-2`“ für 0,03 erfolgen
Zeichen	`char`	Ein einzelnes Zeichen; Zuweisung durch einfache Hochkommas: „`char c = 'a';`“. Ein „leeres Zeichen“, also „`''`“, ist nicht zulässig. Es gibt u. a. folgende „Sonderzeichen“, die auch „Escape-Sequenzen“ genannt werden.: `'\n'`: Zeilenschaltung (newline) `'\r'`: Wagenrücklauf (carriage return) `'\\'`: Backslash `'\''`: einfaches Hochkomma `'\"'`: doppeltes Hochkomma
	`String`	Eine Zeichenkette aus mehreren Zeichen; Zuweisung durch doppelte Hochkommas: „`String s="Wort"`“ Eine leere Zeichenkette, also „`""`“, ist zulässig. Die unter „char“ stehenden Escape-Sequenzen sind zulässig und werden dann als ein Zeichen interpretiert. Ein Backslash muss also entsprechend „escaped“ werden: Die Angabe des Pfades „D:\Computer“ müsste also entsprechend als „`"D:\\Computer"`“ erfolgen.

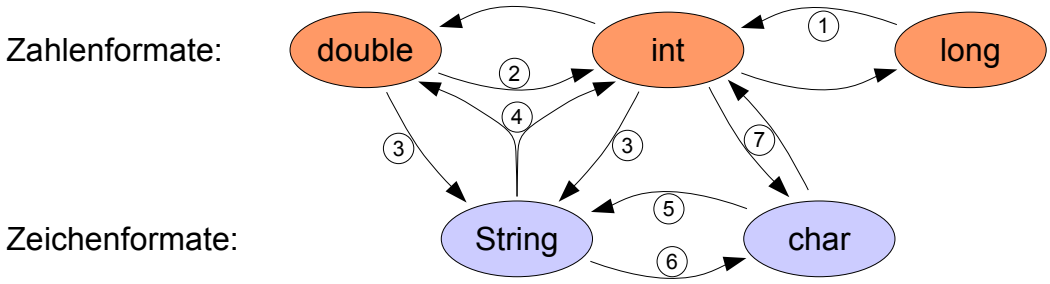
Spezielle Fälle der Anwendung:

- Bei der Auswertung von Ausdrücken vom Typ „`boolean`“ gibt es folgende Verknüpfungen:
  - „`a && b`“ ist wahr, wenn `a` und `b` wahr sind (UND-Verknüpfung)[25]
  - „`a || b`“ ist wahr, wenn `a` oder `b` wahr ist (ODER-Verknüpfung)[26]
  - „`!a`“ ist wahr, wenn `a` falsch ist (Nicht-Operator)

  Wenn `a` oder `b` keine Variablen, sondern Vergleiche o. ä. sind, sollten diese Ausdrücke eingeklammert werden.
- Die mathematische Division wird mithilfe von „`/`“ vorgenommen. Wenn auf beiden Seiten ein Wert vom Typ „`int`“ steht, ist auch das Ergebnis wieder vom Typ „`int`“ und zwar der ganzzahlige Anteil der Division. Der Term „`19/7`“ ergibt also 2. Wenn Du eine Division mit Rest durchführen möchtest, musst Du den Operator „`%`“ verwenden. Der Term „`19%7`“ ergibt also 5.
- Die Zahl π erhältst Du über „`Math.PI`“. Die Zahl e ist über „`Math.exp(1)`“ zu erhalten. Weitere Funktionen der Math-Klasse zur Berechnung von trigonometrischen Funktionen, Wurzeln, Beträgen, Minima und Maxima kannst Du z. B. dem in Kapitel „2.3 Installation des Java-Editors“ erwähnten Java-Buch in Kapitel 17.2 entnehmen.

4.2.2 Type-Casting

Unter Type-Casting versteht man die Umwandlung von verschiedenen Datentypen ineinander, z. B. die Umwandlung einer Zahl in einer Zeichenkette. Die wesentlichen Type-Casting-Fälle habe ich in folgender Übersicht zusammengefasst:



Im Folgenden verwende ich folgende Variablennamen:

```
double d = 4.5;
int    i = 69;
long   l = 123456789;
String s = "709";
char   c = 'j';
```

Die beiden Pfeile ohne Nummer sind Umwandlungen, die automatisch ablaufen. Die folgenden Zeilen werden also vom Compiler akzeptiert:

```
d = i;  // Umwandlung int → double
l = i;  // Umwandlung int → long
```

Die anderen Umwandlungen müssen explizit ausgeführt werden:

1. Eine Umwandlung `long` → `int` ist nur dann ohne Einbußen möglich, wenn die Zahl zwar vom Typ her `long` ist, vom Wert her aber `int` sein könnte. Anderenfalls wird nur der hintere Teil der Dualzahl konvertiert. Die Anweisung lautet:
   ```
   i = (int) l;
   ```
2. Das Umwandeln einer Dezimal- in eine Ganzzahl läuft auf einen Rundungsvorgang hinaus. Dafür gibt es gleich mehrere Varianten:
   a) Abschneiden der Nachkommastellen (aus –4.5 wird die Zahl –4):
      ```
      i = (int) d;
      ```
   b) Mathematisches Runden der Zahl (aus –4.5 wird die Zahl –4):[27]
      ```
      i = (int) Math.round(d);
      ```
   c) Ab- bzw. Aufrunden der Zahl (aus –4.5 wird die Zahl –5 bzw. –4):[28]
      ```
      i = (int) Math.floor(d);  // Abrunden der Zahl
      i = (int) Math.ceil(d);   // Aufrunden der Zahl
      ```
3. Wenn Du Zahlen mithilfe der Methoden `setText` oder `drawString` ausgeben möchtest, wird als Parameter der Datentyp `String` verlangt. Dies wird automatisch mithilfe von „+“ durchgeführt – vorausgesetzt, die Operation beginnt mit einer Zeichenkette, die auch leer sein darf:
   ```
   s = "" + i;  // Umwandlung int → String
   s = "" + d;  // Umwandlung double → String
   ```
   Eine weitere Möglichkeit der Umwandlung von Dezimalzahlen ist die der formatierten Ausgabe. Mit folgender Anweisung wird die Dezimalzahl als Zeichenkette mit zwei Nachkommastellen dargestellt, wobei aufgrund der Ländereinstellung tatsächlich ein Komma verwendet wird (aus –4.5 wird –4,50):[29]
   ```
   java.text.DecimalFormat df = new java.text.DecimalFormat("0.00");
   s = df.format(d);
   ```

Weitere Informationen zu `DecimalFormat` und den möglichen Parametern kannst Du z. B. dem in Kapitel „2.3 Installation des Java-Editors“ erwähnten Java-Buch entnehmen.

4. Die Umwandlung einer Zahl, die in einer Zeichenkette steht, funktioniert folgendermaßen:
```
i = Integer.parseInt(s);     // Umwandlung String → int
d = Double.parseDouble(s);   // Umwandlung String → double
```
Für den Fall, dass die Zeichenkette nicht korrekt umgewandelt werden kann, erfolgt ein Laufzeitfehler.

5. Die Umwandlung eines Zeichens in eine Zeichenkette funktioniert analog zu dem in 3. beschrieben Verfahren:
```
s = "" + c;  // Umwandlung char → String
```

6. Die Umwandlung einer Zeichenkette in ein einzelnes Zeichen ist streng genommen nur für Zeichenketten möglich, die aus einem Zeichen bestehen. Der folgende Befehl schneidet aus einer Zeichenkette ein einzelnes Zeichen an der angegebenen Position heraus, wobei zu beachten ist, dass das erste Zeichen einer Zeichenkette die Position „0“ hat:
```
c = s.charAt(3);  // Umwandlung String → char
```

7. Die folgende Umwandlung ist streng genommen kein Type-Casting. Jedem Zeichen ist über den ASCII-Code eine Zahl zugeordnet und umgekehrt für die Zahlen von 0 bis 255. Diese Umwandlung ist folgendermaßen möglich:
```
c = (char) i;  // Umwandlung ASCII-Code → Zeichen
i = (int) c;   // Umwandlung Zeichen → ASCII-Code
```
Aus „j“ wird also „106“, und aus „69“ wird „E“.

4.2.3 Zeichenketten-Operationen

Im Folgenden eine kurze Übersicht über die am häufigsten verwendeten Methoden von Zeichenketten. Die verwendeten Variablen `s`, `t` und `u` sind vom Typ `String`. Wenn eine Position verwendet wird, musst Du beachten, dass das erste Zeichen einer Zeichenkette die Position 0 hat; dementsprechend hat das letzte Zeichen eine Position, die die Länge der Zeichenkette vermindert um 1 ist.

Methode	**Ergebnis-Typ**	**Bedeutung**
Informationen über die Zeichenkette		
`s.length();`	`int`	ergibt die Länge der Zeichenkette

`s.indexOf(t);`
`s.indexOf(t,anfang);`
`s.lastIndexOf(t);`
	`int`	ergibt die Position der Zeichenkette `t` innerhalb der Zeichenkette `s` und zwar das erste Auftreten bzw. das erste Auftreten ab der Position „anfang“ bzw. das letzte Auftreten. Wenn die Zeichenkette nicht auftritt, lautet das Ergebnis –1.

`s.startsWith(t);` `s.endsWith(t);`	`boolean`	überprüft, ob die Zeichenkette `t` am Anfang bzw. Ende von Zeichenkette `s` steht.
Herauskopieren von Zeichen		
`s.charAt(stelle);`	`char`	ergibt das Zeichen innerhalb der Zeichenkette, das an der Position „stelle“ steht
`s.substring(anfang, ende);` `s.substring(anfang);`	`String`	ergibt die Teil-Zeichenkette ab der Position „anfang“ ggf. bis zur Position, die eine Stelle vor der Position „ende“ liegt

Methode	Ergebnis-Typ	Bedeutung
Veränderung der Zeichenkette Die Zeichenkette `s` selbst wird nicht verändert, sondern nur eine veränderte Kopie als Ergebnis zurückgegeben!		
`s.toLowerCase();` `s.toUpperCase();`	`String`	Die Zeichenkette `s` wird so verändert, dass alle Buchstaben Klein- bzw. Großbuchstaben sind.
`s.trim();`	`String`	Wenn die Zeichenkette `s` mit Leerzeichen beginnt oder endet, werden dies abgeschnitten.
`s.replaceFirst(t,u);` `s.replaceAll(t,u);`	`String`	In der Zeichenkette `s` werden das erste Auftreten der bzw. alle vorkommenden Zeichenketten `t` durch die Zeichenkette `u` ersetzt.
Vergleich von Zeichenketten		
`s.equals(t);`	`boolean`	überprüft, ob die Zeichenketten `s` und `t` den gleichen Inhalt haben
`s.compareTo(t);`	`int`	vergleicht zwei Zeichenketten lexikographisch miteinander; das Ergebnis ist < 0 bzw. $= 0$ bzw. > 0.

Darüber hinaus können zwei Zeichenketten mithilfe von „`+`“ zusammengefügt werden, z. B. „`t+u`“.

Wenn man testen möchte, ob ein Zeichen `c` (vom Typ „`char`“) ein Vokal ist oder nicht, kann man den Ausdruck „`("aeiouäöüAEIOUÄÖÜ".indexOf(c) > -1)`“ verwenden, der vom Typ „`boolean`“ ist.

4.2.4 Label – TextField – TextArea

Die Objekte „Label“, „TextField“ und „TextArea“ eignen sich alle zum Ausgeben von Text. Es gibt aber wesentliche Unterschiede:

	Label	TextField	TextArea
Verwendung	nur Ausgabe	Ein- und Ausgabe	
Ausgabe	einzeilig; links-/rechtsbündig oder zentriert	einzeilig; linksbündig	mehrzeilig
Methode zum Auslesen	`getText()`		
Methode zum Schreiben	`setText(s)`		
			`append(s)`

Die Variable „`s`“ ist vom Typ „`String`“.

Während „`setText`“ den gesamten Inhalt des Objekts ersetzt, fügt „`append`“ weitere Inhalte (unten) hinzu.

Wenn man bei einem TextArea einen Zeilenumbruch hinzufügen möchte, muss man die Escape-Sequenz „`\n`“ verwenden, z. B. in „`setText("Zeile1\nZeile2")`“.

Wenn man den Inhalt eines TextAreas mithilfe von „`getText`“ ausliest, bekommt man einen String, in dem die einzelnen Zeilen durch die Escape-Sequenz „`\n`“ getrennt sind. Man muss also für das Extrahieren einzelner Zeilen den String nach dem Auftreten von „`\n`“ absuchen und dann eine neue Zeile beginnen.

4.2.5 Zufallszahlen

Die Funktion `Math.random()` produziert Zufallszahlen, die im Intervall [0;1[liegen. In den meisten Fällen werden jedoch Zufallszahlen benötigt, die Ganzzahlen aus einem bestimmten Bereich sind. Bei einem simulierten Würfel-Wurf werden Zahlen von 1 bis 6 benötigt, bei der zufälligen Darstellung von Grafik-Elementen vielleicht aus dem Bereich 100 bis 200.
Hier helfen folgende Methoden, die gleichverteilte, ganzzahlige Zufallszahlen liefern:[30]

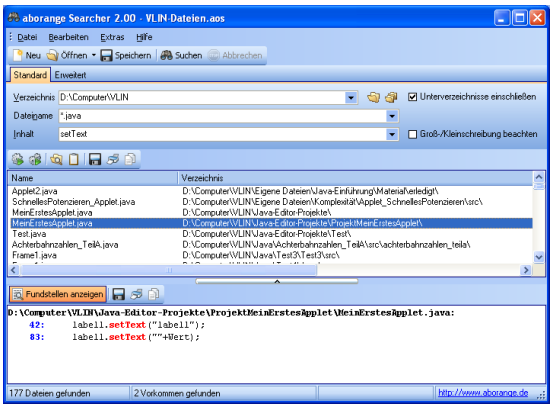
```
public int zufallszahl(int anfang, int ende){
  return (int) (Math.random()*(ende-anfang+1) + anfang);
}

public int zufallszahl(int ende){
  return zufallszahl(1,ende);
}
```

Bei der ersten Methode wird ein Bereich angegeben, z. B. `zufallszahl(100,200)`, bei der zweiten Methode nur die End-Zahl für den Bereich von 1 bis zu dieser Zahl, z. B. `zufallszahl(6)`. Die angegebenen Grenzen treten mit als Zufallszahl auf.

Eine etwas flexiblere, dafür aber aufwendigere Erzeugung von Zufallszahlen ist mithilfe der Klasse „Random“ möglich. Weitere Informationen dazu kannst Du z. B. dem in Kapitel „2.2 Installation des Java-Editors“ erwähnten Java-Buch in Kapitel 16.1.2 entnehmen.

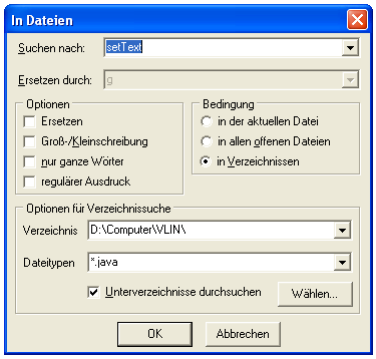
4.2.6 Suchen nach Begriffen in Quelltexten

Die Frage „In welcher JAVA-Datei habe ich die Methode `setText` verwendet?“ lässt sich mit Bordmitteln von Windows XP gar nicht und unter Windows Vista bzw. Windows 7 nur sehr unkomfortabel beantworten. Hier hilft das kleine Freeware-Programm „aborange Searcher“, das unter der URL http://aborange.de/products/freeware/searcher.php heruntergeladen werden kann.



Der „aborange Searcher“ hat den Vorteil, dass ausgewählte Unterverzeichnisse durchsucht werden können und dass diese Sucheinstellungen gespeichert werden können. Außerdem werden bei den gefundenen Dateien die Zeilen angezeigt, in denen der Suchbegriff vorkommt, der dann gleich hervorgehoben wird.

Beim Java-Editor kann auch ohne den „aborange Searcher“ auskommen: Unter „Bearbeiten | In Dateien...“ kann man sich analog die gewünschten Dateien anzeigen lassen. Eine Hervorhebung des Suchbegriffs findet jedoch nicht statt, und die Suche ist nicht ganz so komfortabel. Dafür läuft alles innerhalb eines Programms ab.



5 Wenn Du es genau wissen willst...

Im Text dieses Dokuments sind einige Stellen mit Endnoten gekennzeichnet, zu denen weitergehende Informationen zur Verfügung stehen, die Du aber nur dann lesen solltest, wenn Du es wirklich genau wissen willst:

---

1 Am einfachsten geht das Lesen der Endnoten übrigens, wenn Du dieses Dokument als PDF-Dokument liest. Dann kannst Du einfach auf die jeweilige Endnote klicken und sofort die zugehörige Endnote nachlesen. Mit „Alt+(Nach-links-Taste)“ kommst Du wieder an die richtige Stelle im fortlaufenden Text zurück.

2 Bei Apple-Rechnern ist Java bereits inklusive (siehe http://developer.apple.com/java/). Das JDK gibt es auch für Linux. Der Java-Editor ist nur unter einer Windows-Emulation lauffähig.

3 Der folgende Abschnitt bezieht sich im Wesentlichen auf die Anleitung zur Konfiguration auf der WWW-Seite http://javaeditor.org/index.php?title=Configuration/de. Sie ist für dieses Dokument auf ein notwendiges Maß beschränkt worden.

4 Wenn die Java-Dokumentation fehlt, kann im Java-Editor die automatische Code-Vervollständigung nicht aufgerufen werden. Diese Erweiterung ist also ein absolutes Muss, auf die anderen Erweiterungen kann ggf. verzichtet werden.
Empfehlenswert könnte evtl. noch das Jalopy-Plugin sein, das einem hilft, den Programmtext in einem einheitlichen Stil umzuwandeln. Die Beschreibung dazu und den Link für den Download findest Du unter der URL http://javaeditor.org/index.php?title=Configuration/de.

5 Es handelt sich hierbei um das als HTML-Version (in der 5. Auflage) kostenlos erhältliche Buch „Handbuch der Java-Programmierung“ von Guido Krüger und Thomas Stark. Informationen dazu auf der WWW-Seite http://www.javabuch.de.
Im Kapitel 3.2.3 dieses Java-Buches wird erläutert, wie man auf die Java-Dokumentation bzw. die Java-API-Dokumentation zugreifen kann. Dies ist bei der von mir gewählten Installationsvariante der Dokumentation nicht möglich, weil die Dokumentation nur als CHM-Datei vorliegt, was aber wesentlich platzsparender ist. Der Zugriff aus dem Java-Buch heraus ist deshalb nicht möglich, Du kannst auf die Dokumentation aber immer vom Java-Editor aus zugreifen.

6 Der vom Java-Editor für den Download verwendete Server „download.bildung.hessen.de“ bricht einen Download nach gut 5 Minuten ab, so dass die vollständige Datei im Umfang von 54 MB nicht heruntergeladen werden kann. Der von mir angegebene Link ist die Original-Quelle der verwendeten Dokumentation.
Bei einer extrem langsamen Internet-Verbindung könnte ein vergleichbares Vorgehen sogar beim Tutorial notwendig sein. Der zugehörige Link lautet: http://javadoc.allimant.org/dist/java-tutorial-2008-03-14.zip.

7 In der Standard-Konfiguration verursacht der Java-Editor beim Ausführen eines Applets eine CPU-Auslastung von 50 % (siehe Endnote 22). Dies kann man verhindern, wenn man bei „Fenster | Konfiguration | Optionen“ einen Haken bei „Programme unabhängig vom Java-Editor ausführen“ setzt. In diesem Fall sind allerdings keine Ausgaben mehr auf der Konsole innerhalb des Java-Editors mithilfe von „`System.out.println`“ möglich. Außerdem kann ein „hängendes“ Programm nicht mehr vom Java-Editor aus beendet werden.

8 Der Java-Editor speichert zusammengehörige Dateien immer mit gleichem Dateinamen, aber unterschiedlicher Endung ab. Ein Projekt besteht dann üblicherweise mindestens aus einer JAVA- und einer JFM-Datei. Beim Öffnen einer Datei werden dann beide identisch angezeigt. Die gezielte Auswahl ist jedoch auch bei ausgeblendeten Dateierweiterungen dadurch möglich, dass bei „Dateityp“ anstelle von „Java-Editor“ „Java (*.java)“ bzw. „Formular (*.jfm)“ gewählt wird.

9 Einen Überblick über die weiteren Funktionen des Java-Editors findest Du übersichtlich dargestellt unter der URL http://javaeditor.org/index.php?title=Java-Editor/de.

10 Auf Leerzeichen und Sonderzeichen solltest Du in Verzeichnis- und Dateinamen sicherheitshalber verzichten.

11 Die Veränderung der Größe des Applets wirkt sich dann automatisch auf den Java-Quelltext (Abmessungen des Panels) sowie auf den HTML-Quelltext (Abmessungen des Applets) aus. Die Größenänderung ist alternativ über den Objekt-Inspektor möglich.

12 Der Anfang eines Kommentars in der Form „`/**`“ kennzeichnet einen Javadoc-Kommentar. Weitere Informationen dazu unter http://de.wikipedia.org/wiki/Javadoc.

13 Wenn Du diese Kommentarzeilen nicht bei jedem neuen Projekt löschen möchtest, kannst Du die zugrunde liegende Template-Datei „JEApplet.java“ entsprechend ändern. Sie befindet sich im Verzeichnis „C:\Programme\JavaEditor\templates“. Anschließend muss die Vorlagendatei unter „Fenster | Konfiguration | Editor | Vorlagen | Applet“ zugeordnet werden. Die Änderung des Inhalts der Kommentarzeilen ist darüber hinaus unter „Fenster | Konfiguration | Editor | Kommentar“ möglich. Wenn dort als „Art“ der Wert „frei“ gewählt wird, fällt der Kommentar vollständig weg.

14 Der Java-Editor verwendet beim Erstellen von Programmen mithilfe des Applet-Fensters („Zusammenklicken der Objekte“) immer ein Panel, damit das Vorgehen bei allen Fenster-Objekten (Frames etc.) identisch ist. Durch die Anweisung „`new Panel(null);`“ wird gleichzeitig dafür gesorgt, dass auf dem Panel de facto ein Null-Layout zur Verfügung steht. Wenn Du Programme übernehmen willst, die kein Panel und stattdessen ein

Null-Layout verwenden, solltest Du Dir den Tipp in Kapitel „4.1.3 Weitere Hinweise zum Java-Editor“ ansehen.

15 Da wir weiter unten innerhalb einer anderen Methode den Hintergrund des Panels verändern wollen, muss das Panel selbst global definiert sein, damit wir darauf zurückgreifen können.

16 Wenn Du diese Verschiebung nicht bei jedem neuen Projekt durchführen möchtest, kannst Du die in Endnote 13 genannte Template-Datei entsprechend ändern. Allerdings ist diese Verschiebung nur bei wenigen Projekten wirklich notwendig, so dass Du Dir diese Arbeit sparen kannst.

17 Das Verwenden von Swing-Objekten ist auch möglich, es muss aber zu Beginn des Dokuments von Hand die Zeile „`import javax.swing.*;`“ eingefügt werden.
Die Swing-Formulare sind zwar etwas moderner als die AWT-Formulare (Details dazu findest Du z. B. in Wikipedia unter http://de.wikipedia.org/wiki/Swing_(Java).), das Handling der AWT-Formulare ist allerdings etwas einfacher, so dass Du zumindest am Anfang unbedingt AWT-Formulare einsetzen solltest. Auf dieses lassen sich dann ggf. auch Swing-Komponenten platzieren, wenn es dann unbedingt „schicker“ aussehen soll. Vom „Mischen“ von AWT- und Swing-Komponenten wird in der Regel aber abgeraten. Die Swing-Komponenten erkennt man im Java-Quelltext übrigens daran, dass ihnen ein „J“ vorangestellt ist: „JButton“ bei Swing anstelle von „Button“ bei AWT.

18 Es hat sich eingebürgert, Java-Klassen etc. mit einem Großbuchstaben beginnen zu lassen, während Bezeichner (von Objekten oder Variablen) kleingeschrieben werden.

19 Wenn das Applet-Formular oder das gewünschte Objekt nicht sichtbar ist, kannst Du auch im Objekt-Inspektor das gewünschte Objekt mithilfe des DropDown-Menüs direkt auswählen.

20 Es ist zwar auch möglich, die Beschriftung direkt im Quelltext zu verändern, dies hat jedoch keine Auswirkung auf das Applet-Fenster während des Programmierens, da diese Informationen in der separaten Formular-Datei (Dateierweiterung „.jfm“) gespeichert werden. Wenn sich versehentlich ein Unterschied zwischen beiden ergeben haben sollte, kannst Du einfach im Objekt-Inspektor eine neue Bezeichnung eingeben, die dann automatisch in den Quelltext übernommen wird.

21 Diese Einstellung kann ggf. unter „Fenster | Konfiguration | Java | Applets“ geändert werden.

22 Natürlich können auch alle Dateien in einem Verzeichnis stehen; dann muss jedoch im HTML-Code die codebase-Anweisung gestrichen werden. Analog kann das Applet auch in jede beliebige HTML-Datei eingebunden werden, indem die applet-Anweisung in die gewünschte HTML-Datei übernommen und ggf. bezüglich des Unter-Verzeichnisses angepasst wird.

23 Weitere Dokumente der VLIN befassen sich ebenfalls ausführlich mit den Grafik-Befehlen.

24 Vgl. aber auch Endnote 9.

25 Es ist auch die Verknüpfung „`&`“ möglich. Bei der Verwendung von „`&&`“ wird jedoch `b` gar nicht ausgewertet, wenn `a` falsch ist, weil dann das Ergebnis falsch sein muss.

26 Es ist auch die Verknüpfung „`|`“ möglich. Bei der Verwendung von „`||`“ wird jedoch `b` gar nicht ausgewertet, wenn `a` wahr ist, weil dann das Ergebnis wahr sein muss.

27 Der `round`-Befehl existiert nur in der Math-Bibliothek. Er liefert einen Wert vom Typ `long` zurück, so dass noch ein zusätzliches Type-Casting mithilfe von `(int)` durchgeführt werden muss.

28 Auch die Befehle `floor` und `ceil` existieren nur in der Math-Bibliothek. Obwohl auf eine Ganzzahl gerundet wird, haben sie den Ergebnistyp `double`, so dass mithilfe von `(int)` noch ein zusätzliches Type-Casting durchgeführt werden muss.

29 Bei mehrfacher Verwendung von `DecimalFormat` kann mithilfe von „`import java.text.*;`“ am Anfang der JAVA-Datei der Quelltext einfacher gestaltet werden.

30 Mathematischer Hintergrund: Aufgrund des Abschneidens mit „`(int)`“ muss die Dezimal-Zufallszahl mit der Anzahl der im Bereich liegenden Zahlen, also mit „`ende-anfang+1`“, multipliziert werden. Anschließend muss mithilfe von „`+anfang`“ noch eine Verschiebung in den gewünschten Bereich stattfinden. Die Verwendung von „`round`“ würde dazu führen, dass die beiden Randzahlen des Bereichs nur halb so häufig auftreten wie die anderen Zahlen. Aus diesem Grund wurde das Type-Casting mithilfe von „`(int)`“ gewählt.