

Programmierung in der gymnasialen Oberstufe
für Lehramtsstudierende im Fach Informatik

**Grundlagen und GUIs**

Autor:innen: Maximilian Wadenpohl, Paula Bobach

Hinweise zur Durchführung von Veranstaltungen mit den Materialien:

Das Material besteht aus Aufgabenstellungen zum Java-Editor und der damit verbundenen Arbeit mit GUIs mithilfe von Java Swing. Neben den Aufgaben zur Orientierung und zum Aufdecken von Nachholbedarfen bei den Studierenden stehen im Anhang einige Materialien zur inhaltlichen Auffrischung zur Verfügung, die die Studierenden insbesondere im Selbststudium nutzen können. Auf dem Material aufbauend lassen sich beispielsweise Seminarsitzungen gestalten, die auf die Programmierhinhalte der gymnasialen Oberstufe fokussieren oder didaktischen Diskussionen Raum geben. Bei der angehängten Beispiellösung der Programmieraufgaben handelt es sich nicht um eine Musterlösung. Diese soll weitere Diskussionsanlässe schaffen und enthält daher z.B. keine Kommentare.

Weitere Anmerkung, die sich im Zusammenhang der Durchführung von Seminarsitzungen mit dem Material ergeben haben:

* Insbesondere Studierende ohne Erfahrungen mit Java oder dem Java-Editor benötigen teils viel Zeit zur Vorbereitung des gesamten Materials.
* Die Materialien können Anlass zur Diskussion über viele verschiedene didaktische Aspekte sein, die im Zusammenhang mit der Programmierung im Informatikunterricht stehen. Eine Fokussierung erscheint hier für die Vorbereitung einer Seminarsitzung notwendig.

Mögliche Diskussionsthemen auf Basis dieses Materials:

* Klassenbibliotheken und ihre Bedeutung im Informatikunterricht
* Wie erschließe ich unbekannte Quelltexte?
* Angemessenes Kommentieren von Quelltexten
* „projektartiges Arbeiten“ im Informatikunterricht
* Unterschiede zum Programmieren in Vorlesungsmodulen an der Universität
* Vor- und Nachteile des Java-Editors, wofür kann dieser besonders gut im Unterricht eingesetzt werden?
* Bezug zum Kernlehrplan: Welche Kompetenzen können durch die vorliegenden Aufgaben gefördert werden?
* Wie könnten die vorliegenden Aufgaben im Unterricht eingesetzt werden? Wie müssen diese dazu modifiziert und aufbereitet werden?

**Aufgabe 1**: (Java-Editor und GUI mit Swing)

Hinweis: Java-Editor ist für MacOS leider nicht verfügbar. Ggf. können Sie mit einer virtuellen Maschine arbeiten.

1. Installieren Sie den Java-Editor mithilfe der Instruktionen auf dieser Website: <https://javaeditor.org/doku.php?id=de:installation>

Installieren Sie ggf. vor der Installation des Java Editors eine aktuelle Java jdk.

Abbildung 1: Configuration Fenster zum Eingeben des Dateipfades

Öffnen Sie die Entwicklungsumgebung. Beim erstmaligen Ausführen vom Java-Editor wird der Nutzer dazu aufgefordert in der Configuration den Dateipfad zu der auf dem Gerät installierten Java JDK anzugeben (siehe Abbildung 1). Geben Sie den Dateipfad ihres Geräts an und speichern sie die Konfiguration anschließend.

1. Erstellen Sie zu Testzwecken unter Datei 🡪 Neu 🡪 JFrame ein neues Programmierprojekt.

Im erzeugten Java-File wurden nun automatisch einige Packages, die zur Verwendung von Swing notwendig sind, importiert. Weiterhin wurde ein Klasse mit dem gewählten Namen erzeugt, die von der Klasse JFrame erbt und im Konstruktor der Klasse sind bereits einige Standardeinstellungen für das GUI vorgenommen worden. Letztlich erscheint zudem das in Abbildung 2 gezeigte Fenster, welches den Aufbau des GUI stark erleichtert.

Abbildung 2: Fenster zur Bearbeitung des GUI

1. Fügen Sie Ihrem GUI über die Reiter „Swing 1“ und „Swing 2“ Bestandteile hinzu. Sie können beobachten, dass die neuen Elemente automatisch dem Konstruktor beigefügt werden. Fügen Sie auch mindestens einen Button hinzu. Klicken Sie auf ein Element des GUI werden Ihnen im Java-Editor sämtliche Eigenschaften angezeigt, die das jeweilige Element aufweist (Abb. 3). Ändern Sie einige Eigenschaften Ihrer GUI-Komponenten und beobachten Sie diese ggf. auch durch das Ausführen des Programms (grüner Ausführen-Button oben links im Java-Editor). Sie können diese Änderungen unmittelbar in dem Bereich ändern, der in Abb. 3 rechts gezeigt wird. Der Konstruktor Ihrer Klasse wird automatisch dementsprechend angepasst.

Abbildung 3: Eigenschaften eines jButtons (Ausschnitt)

1. Machen Sie sich mit den möglichen Komponenten und Ihren Eigenschaften vertraut. Ihnen stehen dazu im Anhang dieses Dokuments Erläuterungen von javaeditor.org zur Verfügung. Weiterhin können Sie die Dokumentation des Packages Java Swing verwenden, die Sie hier finden: <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>
2. Durch das Hinzufügen von mindestens einem Button wurde Ihre Klasse auch um eine Methode ergänzt, die etwa den Namen „JButton1\_ActionPerformed“ aufweist. Lassen Sie diese Methode eine Konsolenausgabe vornehmen. Starten Sie Ihr Programm und klicken Sie den entsprechenden Button. Ihre vorgesehene Konsolenausgabe sollte nun ausgegeben werden.
3. Beobachten Sie im Konstruktor Ihrer Klasse, mit welchen externen Methodenaufrufen Sie die Eigenschaften der Komponenten Ihres GUI jederzeit verändern können. Die dazu zur Verfügung stehenden Methoden können Sie ebenfalls in der oben angegebenen Dokumentation finden.

**Aufgabe 2** (Snackautomat):

Im Folgenden wird ein Snackautomat beschrieben, der in dieser Form mit einem GUI gegen Ende der Einführungsphase im Informatikunterricht umgesetzt werden könnte.

Ein Snackautomat soll die Möglichkeit bieten, aus vier verschiedenen Produkten auszuwählen. Das GUI zu diesem Snackautomat in seinem anfänglichen Zustand ist in Abbildung 4 dargestellt. Nach Auswahl eines Produkts muss das Produkt durch den Einwurf von Münzen bezahlt werden. Sobald der noch ausstehende Betrag auf 0 oder darunter fällt, wird entsprechend Rückgeld gezahlt oder das Produkt ausgegeben. Der Münzbestand wird auf der rechten Seite des GUI dokumentiert. Der Kunde entnimmt letztlich ggf. das Wechselgeld und das Produkt und der Automat geht wieder in seinen Ausgangszustand über. Während des gesamten Vorgangs wird der Kunde über das Label in der Mitte des GUI über den nächsten Schritt informiert.

Abbildung 4: GUI Snackautomat

1. Öffnen Sie den Ordner „Automat“ und die darin enthaltenen Klassen **Automat** und **GUIAutomat** mit dem Java-Editor.
2. Verschaffen Sie sich mithilfe der Klassendokumentationen im Anhang einen Überblick über die Klassen.
3. Starten Sie das Programm und testen Sie einige Male die Funktionen des Snackautomaten. Führen Sie dabei auch den Fall herbei, dass ein auszuzahlendes Rückgeld aufgrund des Münzbestands nicht entrichtet werden kann.

Die Implementation des Snackautomaten enthält viele Grundlagen der Programmierung, die in der EF erlernt werden. Darunter zählen hier vor allem Aufträge, Anfragen, Verzweigungen, Schleifen, Arrays, übergebene Parameter, externe Methodenaufrufe, Vergleichsoperationen und Datentypen (Besonderheiten bei Strings).

1. Betrachten Sie den Quelltext. Versuchen Sie, mithilfe der im Anhang befindlichen Materialien zu den oben genannten Grundlagen den gesamten Quelltext nachzuvollziehen. Notieren Sie mögliche Diskussionsanlässe für die kommende Seminarsitzung.
2. Der Programmcode ist weitestgehend unkommentiert. Fügen Sie sinnvolle Kommentare hinzu und erörtern Sie, welche Kommentare sie von Ihren Schüler:innen erwarten würden.
3. Erläutern Sie die jeweilige Methode Zeile für Zeile. Beziehen Sie sich dabei auf Funktionsweise und Zweck der Methode. Erörtern Sie, welche Ausführungen Sie von einem Ihrer Schüler:innen erwarten würden. Welche Fachbegriffe werden hierbei benötigt? Welche Fachbegriffe haben Sie selbst in ihrer Erörterung verwendet? Highlighten Sie diese nachträglich.
	1. **void** **einzahlung(int betrag)** der Klasse **GUIAutomat**
	2. **boolean wechselGeldAusgeben()** der Klasse **Automat**
4. Modifizieren und ergänzen Sie den Quelltext wie folgt:
	1. Fügen Sie ein neues Produkt hinzu. Ergänzen Sie das GUI und die Funktionsweisen entsprechend.
	2. Die im GUI angezeigten Preise sollen bei Erzeugung des GUI aus dem entsprechenden Attribut des Snackautomat-Objekts ausgelesen und so dynamisch anpassbar werden. Auf diese Weise müssen Preisänderungen nur in diesem Attribut vorgenommen werden.
	3. Ergänzen Sie die Funktionen des Snackautomats um die Einzahlmöglichkeit eines 5€-Scheins.
	4. Fügen sie dem GUI einen Abbrechen-Button hinzu, der den Snackautomaten nach Auswahl eines Produkts und vor der Ausgabe von Wechselgeld oder Produkten in seinen Ausgangszustand zurückversetzen kann. Bisher eingezahltes Geld soll beim Betätigen des Buttons zurückgegeben werden.

Im Folgend werden einige Kompetenzen aus dem Kernlehrplan NRW GOSt 2014 S. 20 – 23 aufgeführt, die im Rahmen der hier vorgestellten Aufgabe geschult werden könnten.

**Kompetenzbereiche:**

Argumentieren (A)

* Die Schülerinnen und Schüler analysieren und erläutern Computerprogramme.

Implementieren (I)

* Die Schülerinnen und Schüler modifizieren und erweitern Computerprogramme.

Kommunizieren und Kooperieren (K)

* Die Schülerinnen und Schüler verwenden Fachausdrücke bei der Kommunikation über informatische Sachverhalte.
* Die Schülerinnen und Schüler präsentieren Arbeitsabläufe und -ergebnisse.

**Inhaltsfelder:**

Daten und ihre Strukturierung; Objekte und Klassen

* Die Schülerinnen und Schüler implementieren Klassen in einer Programmiersprache auch unter Nutzung dokumentierter Klassenbibliotheken.

Algorithmen; Analyse, Entwurf und Implementierung einfacher Algorithmen

* Die Schülerinnen und Schüler analysieren und erläutern einfache Algorithmen und Programme.
* Die Schülerinnen und Schüler modifizieren einfache Algorithmen und Programme.
* Die Schülerinnen und Schüler implementieren Algorithmen unter Verwendung von Variablen und Wertzuweisungen, Kontrollstrukturen sowie Methodenaufrufen.

Formale Sprachen und Automaten; Syntax und Semantik einer Programmiersprache

* Die Schülerinnen und Schüler implementieren einfache Algorithmen unter Beachtung der Syntax und Semantik einer Programmiersprache.

**Anhang**

**Dokumentationen der verwendeten Klassen**

**Die Klasse GUIAutomat**

Objekte der Klasse GUIAutomat verwalten ein GUI für einen Snackautomaten.

**Auszug aus der Dokumentation der Klasse GUIAutomat**

**GUIAutomat()**

Ein Objekt der Klasse GUIAutomat wird initialisiert.

**void pButton1\_ActionPerformed(ActionEvent evt)**

Es werden die Aufträge ausgeführt, die beim Klicken auf den ersten Produktbutton angesprochen werden sollen. Analog für alle weiteren ActionPerformed-Methoden der anderen Buttons. Diese werden hier nicht explizit aufgeführt.

**String betragAnzeige(int betrag)**

Wandelt den übergebenen Geldbetrag (cent) in einen String der Form „x,xx€“ um und gibt diesen zurück.

**void sperreProdukte()**

Sperrt alle Buttons, mit denen Produkte ausgewählt werden können.

**void entsperreProdukte()**

Entsperrt alle Buttons, mit denen Produkte ausgewählt werden können.

**void sperreEinzahlung()**

Sperrt alle Buttons, mit denen Münzgeld eingezahlt werden kann.

**void entsperreEinzahlung()**

Entsperrt alle Buttons, mit denen Münzgeld eingezahlt werden kann.

**void produktAuswahl(String produkt)**

Wählt das übergebene Produkt aus, ermittelt seinen Preis und zeigt diesen im Anweisungslabel des GUI an. Bereitet das GUI auf das Einzahlen von Münzen vor.

**void einzahlung(int betrag)**

Zahlt eine Münze entsprechend dem übergebenen Betrag ein und prüft, ob weiterhin eingezahlt werden muss oder die Produktausgabe bzw. Auszahlung von Rückgeld übergegangen werden muss, die dann entsprechend eingeleitet werden. Falls der Münzbestand für das auszuzahlende Rückgeld nicht ausreicht, wird das GUI gesperrt. Der angezeigte Münzbestand wird aktualisiert.

**void produktAusgabe()**

Die Produktausgabe wird eingeleitet und das GUI entsprechend vorbereitet.

**void ausgangsZustand()**

Der Ausgangszustand des GUI wird wiederhergestellt.

**void updateMuenzbestand()**

Die Anzeige des Münzbestands im GUI wird aktualisiert.

**Die Klasse Automat**

Objekte der Klasse Automat verwalten einen Snackautomaten.

**Auszug aus der Dokumentation der Klasse Automat**

**Automat()**

Ein Objekt der Klasse Automat wird initialisiert.

**int getMuenzbestand(int betrag)**

Gibt die Anzahl der Münzen zurück, die dem übergebenen Geldbetrag entsprechen.

**int getZahlbetrag()**

Gibt den zu zahlenden/zurück zu gebenden Geldbetrag zurück. Ein Rückgabewert <0 bedeutet dabei, dass Geld zurückgegeben werden muss, ein Rückgabewert >0 gibt einen noch durch den Kunden zu zahlenden Betrag an.

**String produktAusgabe()**

Händigt dem Kunden ein Produkt aus und gibt den Namen des Produkts zurück.

**int waehleProdukt(String auswahl)**

Das übergebene gewählte Produkt und der durch den Kunden zu zahlende Betrag wird gespeichert und der Preis des Produktes zurückgegeben.

**void einzahlen(int betrag)**

Zahlt eine Münze entsprechend dem übergebenen Betrag in den Bestand ein und verringert den durch den Kunden zu leistenden Zahlbetrag entsprechend.

**boolean wechselGeldAusgeben()**

Prüft, ob der durch den Automat zu leistenden Betrag an Rückgeld mit dem vorhandenen Münzbestand auszahlbar ist und gibt entsprechend true oder false zurück. Während des Prüfvorgangs wird das Rückgeld bereits parallel ausgezahlt. Dabei werden Münzen mit höherem Wert stets bevorzugt.

### Nachfolgend finden sich einige Informationen der Java-Editor-Homepage zu verschiedenen GUI-Komponenten. Diese wurden unter der CreativeCommons-Lizenz CC BY-SA 4.0 veröffentlicht. Zu finden sind diese auch unter <https://javaeditor.org/doku.php?id=de:gui-verwendung#jtable>

### JLabel

 Mit einer JLabel-Komponenten beschriftet man Komponenten eines GUI-Formulars. Eine JLabel Komponente kann außer Text auch ein Bild oder Beides darstellen. Um ein Bild darzustellen wählt man im Object-Inspector beim Attribut Icon die gewünschte Bilddatei aus.

Beispiel: 

Mit der Methode setLocation(int x, int y) der JLabel-Komponente kann man ein Bild animieren. 

### JTextField

 Mit einer JTextField-Komponente kann man Text eingeben. Den eingegebenen Text erhält man über die getText()-Methode der JTextField-Komponenten.

Beispiel: 

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) [Name](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+name) = tfName.getText();

### JNumberField

 Mit einer JNumberField-Komponenten kann man ganz einfach Zahlen anzeigen und eingeben.

Die Methoden getDouble(), getFloat(), getInt() undgetLong() liefern Zahlen des gewünschten Typs:

Beispiel: 

int Alter = nfAlter.getInt();

Dezimalzahlen können mit maximaler oder einer angegeben Anzahl von Nachkommastellen ausgegeben werden:

Beispiel:

nfSumme.setDouble(Summe); *// maximale Anzahl von Nachkommastellen: 3657.42323426347834*

nfSumme.setDouble(Summe, 2); *// zwei Nachkommastellen: 3657.42*

### JTextArea

 Eine JTextArea-Komponente stellt mehrzeiligen Text dar, eine JTextField-Komponente einen einzeiligen. Im Object-Inspector kann man den Text eingeben. Zur Laufzeit fügt man mit der append()-Methode Text hinzu bzw. setzt den Text mit setText() oder liest ihn mit getText(). Die Zeilen werden durch das Steuerzeichen „\n“ (new line) voneinander getrennt.

Beispiel: 

 taAusgabe.setText("Ausgabe:**\n\n**");

 taAusgabe.append("Zahl " + Zahl + " gefunden! **\n**");

### JButton

 Jede JButton-Komponente erhält automatisch eine Ereignismethode für das actionPerformed-Ereignis, das beim Anklicken eines Buttons erzeugt wird. Beim Doppelklicken eines Buttons im GUI-Formular wird der Cursor an den Anfang der zugehörigen Ereignismethode platziert.

Beispiel:

 public void jButton1\_ActionPerformed([ActionEvent](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+actionevent) evt) {

 *// TODO hier Quelltext einfügen*

 } *// end of jButton1\_ActionPerformed*

### JCheckBox

 Eine JCheckBox-Komponente kann ausgewählt sein oder nicht. Den aktuellen Zustand liefert die Methode isSelected().

Beispiel: 

 if (MeineCheckBox.isSelected()) ...

### Buttongroup

 Eine Buttongroup gruppiert Radiobuttons oder Checkboxen. Setzen Sie das Attribut Checkboxes auf true, wenn Sie eine Gruppe von Checkboxen wünschen. Geben Sie Ihre Optionen mit dem Items-Attribut ein.

Für Radiobuttons fügt der Java-Editor diese Methode in den Quelltext ein:

 public [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) buttonGroup1BG\_getSelectedButtonGroupLabel() { ... }

Damit lässt sich dann ganz einfach die ausgewählte Option einer Buttongroup ermitteln. Beispiel:

 if (bgFarbeBG\_getSelectedButtonGroupLabel().equals("grün"))

### JList



Eine JList-Komponente zeigt eine Liste von Objekten an. Der Anwender kann ein oder mehrere Objekte auswählen. Im Objekt-Inspektor kann man über das Attribut Items Strings in die Liste eingeben.

Die Daten der Liste werden im ListModel verwaltet. Zur Laufzeit kann man mit Methoden des ListModels die Daten bearbeiten.

Beispiele:

Zugriff über das ListModel

 meinListeModel.addElement("Vera");

 meinListeModel.remove(0);

 [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = ([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)) meinListeModel.elementAt(3);

Das ListModel kennt zwar alle Daten, aber ein Element wird in der Liste ausgewählt, nicht im ListModel.

Zugriff über die Liste

 int i = meineListe.getSelectedIndex();

 [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = ([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)) meineListe.getSelectedValue();

Enthält die Liste Zahlen, so muss der ausgewählte String noch in eine Zahl umgewandelt werden:

 [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = ([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)) meineListe.getSelectedValue();

 int Zahl = [Integer](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+integer).parseInt(s);

### JComboBox



Eine JComboBox-Komponente ist eine Kombination aus einer Eingabezeile und einer ausklappbaren Auswahlliste. Der Anwender kann zur Laufzeit ein Listenelement auswählen oder in der Eingabezeile Text eingeben.

Im Objekt-Inspektor kann man über das Attribut Items Strings in die Auswahlliste eingeben. Um in der Eingabezeile einer ComboBox etwas eingeben zu können, setzt man die Eigenschaft editable auf true.

Die Daten der ComboBox werden im ComboBoxModel verwaltet. Zur Laufzeit kann man die Daten mit Methoden des ComboBoxModels bearbeiten.

Beispiele:

Zugriff über das ComboBoxModel

 meinComboBoxModel.addElement("Vera");

 meinComboBoxModel.removeElementAt(0);

 [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = ([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)) meinComboBoxModel.getElementAt(3);

Das CombobBoxModel kennt zwar alle Daten, aber ein Element wird vom Benutzer in der ComboBox ausgewählt, nicht im ComboBoxModel.

Zugriff über die ComboBox

 int i = meineComboBox.getSelectedIndex();

 [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) s = ([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)) meineComboBox.getSelectedItem();

Um automatisch auf eine Eingabe oder Auswahl in einer ComboBox reagieren zu können, erstellt man im Objekt-Inspector eine Ereignismethode für actionPerformed.

### JSpinner



Mit einer JSpinner-Komponente kann man Zahlen aus einem Bereich zwischen Minimum und Maximum auswählen. Im Object-Inspektor legt man Minimum, Maximum, Schrittweite (StepSize) und aktueller Wert (Value) fest.

Die Daten einer Spinner-Komponente werden im SpinnerModel verwaltet. Zur Laufzeit kann man mit Methoden des SpinnerModels die Daten bearbeiten.

Beispiele:

Zugriff über das SpinnerModel

 meinSpinnerModel.setStepSize(2);

 int Wert = meinSpinnerModel.getNumber().intValue();

Zugriff über den Spinner

 int Wert = (int) meinSpinner.getValue();

### JScrollBar



Mit einer JScrollBar-Komponente kann man einen Bildlauf durchführen, bei dem der sichtbare Ausschnitt eines dargestellten Textes oder einer Grafik verschoben wird.

Dazu erstellt man mit dem Objekt-Inspektor eine adjustmentValueChanged-Ereignismethode, in der man mit

 int Wert = meineScrollBar.getValue();

den ScrollBar-Wert abruft und dementsprechend den Bildlauf durchführt.

### JScrollPane



Eine JScrollPane-Komponente stellt eine horizontale und eine vertikale JScrollBar-Komponente für einen zweidimensinonalen Bildlauf zur Verfügung.

Der Java-Editor versieht JTextArea, JList, JTable, JTree, JEditorPane und JTextPane-Komponenten automatisch mit einer JScrollPane-Komponenten.

### JPanel



JPanel-Komponenten setzt man zur Strukturierung von GUI-Oberflächen ein. Jede JPanel-Komponente ist ein Container, der andere GUI-Komponenten aufnehmen kann. Bei JPanel-Komponenten kann man im Objekt-Inspektor einen Rahmen setzen.

### Canvas



Eine Canvas-Komponente stellt eine Zeichenfläche zur Verfügung (canvas, engl. Leinwand).

Zum Zeichen auf der Zeichenfläche stellt eine Canvas-Komponente einen sogenannten Grafikkontext zur Verfügung. Man kann sich das vereinfacht als Deckfarbkasten aus dem Kunstunterricht vorstellen, während die Zeichenfläche dem Malblock entspricht.

Nennt man die Canvas-Komponente Zeichenflaeche, so erhält man den zugehörigen Grafikkontext mit

 [Graphics](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+graphics) g = Zeichenflaeche.getGraphics();

Mit Hilfe des Deckfarbkastens – sprich Grafikkontextes g – kann man dann beispielsweise ein Rechteck zeichnen:

 g.fillRect(100, 50, 200, 250);

Minimiert man das Anwendungsfenster mit einer Canvas-Komponente und stellt dann das Fenster wieder dar, so ist der Inhalt der Zeichenfläche verschwunden. Will man dies vermeiden, so muss man eine Unterklasse von Canvas erstellen und in ihr mit der Methode paint(Grahics g) die Zeichnung erstellen. Eine solche Klasse kann über die rechte Maustaste auf das Canvas-Symbol in das GUI-Formular aufgenommen werden.

Die zum Java-Editor gehörende Turtle ist ein Beispiel für eine Canvas-Unterklasse.

### Turtle



Ab Version 14.04 haben wir eine animierte Turtle für Programme mit JavaFX-Oberfläche. Sie unterstützte wie dir frühere Turtle ein kartesisches Koordinatensystem, das man auf der Zeichenfläche mit setOriginX/setOriginY platzieren kann.



Seit Version 13.00 des Java-Editors hat sich die Turtle-Komponente geändert. Wir haben nun eine animierte Turtle-Komponente, die auf der Turtle von [Ägidius Plüss](http://www.aplu.ch/home/apluhomex.jsp) basiert. Die Playground- und Turtle-Komponete befinden sich auf der Utilities-Leiste. Man kann ein oder mehrere Turtles auf einem Playground platzieren.

Zwecks didaktischer Reduktion und wegen technischer Gegenheiten reagiert die GUI nicht, solange die Turtle zeichnet.



Vor Version 13.00 gab es eine viel einfachere Turtle. Wenn Sie ein existierendes Programm mit dieser alten Turtle verwenden wollen, so müssen Sie diese so importieren:

import je.util.Turtle;

Als didaktische Reduktion gibt es die setOrigin(double x, double y)-Methode, mit welcher in der Zeichenfläche ein aus dem Mathematikunterricht bekanntes Koordinatensystem eingerichtet werden kann.

Beispiel: 

Bei der neuen Turtle gehört die setOrigin()-Methode zur Playground-Klasse.

### JMenuBar



Mit einer JMenuBar-Komponente kann man eine Menüleiste erstellen. Die Konfiguration erfolgt über die Menüs (JMenu-Komponenten), die in der Menüleiste angezeigt werden sollen.

### JMenu



Mit einer JMenu-Komponente erstellt man ein Menü für eine Menüleiste. Im Objekt-Inspektor gibt man beim Attribut MenuBar die gewünschte Menüleiste an, beim Attribut MenuItems trägt man die Menübefehle (z. B. New, Open, Save, Print, exit) ein und beim Attribut Text die Bezeichnung des Menüs (z. B. File) in der Menüleiste.

Für jeden Menübefehl erzeugt der Java-Editor eine zugehörige Ereignismethode.

### JPopupMenu



Mit einer JPopupMenu-Komponente erstellt man ein Popup-Menü. Im Beispiel reagiert die Canvas-Komponente mit gelbem Hintergrund auf den Rechtsklick und zeigt das Popup-Menü an.



Im Objekt-Inspektor gibt man beim Attribut Listener die GUI-Komponente an, die beim Rechtsklick das Popup-Menü anzeigen soll. Bei der Vorgabe this reagiert der Hintergrund des Anwendungsfensters auf einen Rechtsklick. Beim Attribut MenuItems trägt man die Menübefehle (z. B. Clear, Draw, Print) ein. Normalerweise wird das Attribut Text als Label eines Popup-Menüs nicht angezeigt.

Für jeden Menübefehl erzeugt der Java-Editor eine zugehörige Ereignismethode.

### Timer



Mit einer Timer-Komponente kann man Abläufe zeitabhängig steuern. Vorgänge können nach einer vorgegebenen Zeitspanne einmalig oder periodisch gestartet und gestoppt werden.

Die Timer-Komponente verwendet die Klasse javax.swing.Timer, nicht die Klasse java.util.Timer.

Beispiel:

 timer1.start();

 ... tue etwas

 timer1.stop();

### JTable

 Mit einer JTable-Komponente kann man Tabellen darstellen.

Im Object-Inspector stellt man die gewünschte Anzahl von Spalten (ColCount) und Zeilen (RowCount) ein. Im Attribut Columns gibt man die Namen der Spalten ein. Setzt man AutoCreateRowSorter auf true, so kann man nach den Spaltennamen sortieren.

Mit der Methode setValueAt(Object aValue, int row, int column) kann man einen Wert in die Zelle an der Position (row, column) setzen und mit getValueAt(int row, int column) lesen. getSelectedRow() liefert den Index der ersten ausgewählten Zeile.

Beispiele: „Anna“ wird in die 1. Spalte der ausgewählten Zeile geschrieben.

 int i = jTabelle.getSelectedRow();

 jTabelle.setValueAt("Anna", i, 0);

### Nachfolgend finden sich einige Informationen von zum Unterrichten zu bedingten Anweisungen und Operatoren. Diese wurden unter der CreativeCommons-Lizenz CC BY-SA 4.0 veröffentlicht. Zu finden sind diese auch unter <https://unterrichten.zum.de/wiki/Java/Algorithmik#Vergleichsoperatoren>

## Bedingte Anweisung



[Struktogramm](https://unterrichten.zum.de/wiki/Algorithmus#Grundstrukturen) einer if-Anweisung

Eine **bedingte Anweisung** ist eine der wichtigsten Bestandteile der Program­mierung, da durch sie ein Programm auf unterschiedliche Zustände und Eingaben reagieren kann.

**if** (Bedingung) {

 *//Anweisungsblock*

 …

}

**else** {

 *//Anweisungsblock*

 …

}

Dies ist folgendermaßen zu lesen:

* **Wenn** die Bedingung eintritt,
dann wird der erste Anweisungsblock ausgeführt;
* **Sonst** wird der zweite Anweisungsblock ausgeführt.

Bedingte Anweisungen können auch **verschachtelt** werden:
Im folgenden Beispiel ist im else-Zweig der ersten Verzweigung eine weitere Verzweigung untergebracht.

**public** void prüfeZahl(int pZahl){

 **if** (pZahl >= 10) {

 System.out.println(pZahl + " ist zwei oder mehrstellig");

 }

 **else** {

 **if** ( pZahl < 0) {

 System.out.println(pZahl + "ist negativ.");

 }

 **else** {

 System.out.println(pZahl + " liegt zwischen 0 und 9.999...");

 }

 }

}

## Operatoren

### Vergleichsoperatoren

Um solche Bedingungen zu formulieren, brauchen Sie Vergleichsoperatoren

 < (kleiner)

 > (größer)

 <= (kleiner gleich)

 >= (größer gleich)

 == (gleich mit zwei == Zeichen)

 != (ungleich)

### Logische Operatoren

Wichtige logische Operatoren sind UND, ODER und NICHT.
Wir verwenden sie täglich, wenn wir Aussagen formulieren, häufig in Sätzen mit „Wenn...“:

* **Wenn** ich eine gute schriftliche **UND** mündliche Note habe, bekomme ich eine gute Zeugnisnote (d.h. nur wenn beides der Fall ist):

**if** ((klausurNote <= 2) && (soMiNote <= 2)) ...

* **Wenn** es regnet **ODER** die Sonne stark scheint, sitze ich unter einem Schirm (d.h. wenn zumindest eines erfüllt ist. Wenn es regnet und zugleich die Sonne stark scheint, sitze ich natürlich erst recht unter einem Schirm.)

**if** ((sonneScheint == **true**) ||(esRegnet == **true**)) ...

* **Wenn** es **NICHT** hell ist, dann geht die Straßenlaterne an.

**if** (!(helligkeitsSensor > 30)) *// ... das ist das gleiche wie: if (helligkeitsSensor <= 30)*

**Zusammenfassung**

Für zwei boolesche Werte a und b gilt:

* **!a** bedeutet **NICHT a** und erzeugt das **Gegenteil von a**. Wenn a true ist, ist !a false und andersherum.
* **a && b** bedeutet **a UND b** und ist true, wenn **sowohl a als auch b true** sind. Sonst nicht.
* **a || b** bedeutet **a ODER b** und ist true, wenn **entweder a oder b oder beide true** sind.
D.h. nur wenn sowohl a als auch b false sind, ist die Aussage a

### Nachfolgend finden sich einige Informationen von zum Unterrichten zu Schleifen. Diese wurden unter der CreativeCommons-Lizenz CC BY-SA 4.0 veröffentlicht. Zu finden sind diese auch unter <https://unterrichten.zum.de/wiki/Java/Schleife>

## for-Schleife

Häufig benötigt man beim Programmieren eine mehrfache Ausführung eines Teilbereichs. Denkbar ist, dass ein Programmierer dazu einfach Zeilen wiederholt. So lässt sich z.B. das kleine 1 x 1 von 3 wie folgt ausgeben:

System.out.println("3 x 1 = 3");

System.out.println("3 x 2 = 6");

System.out.println("3 x 3 = 9");

System.out.println("3 x 4 = 12");

System.out.println("3 x 5 = 15");

System.out.println("3 x 6 = 18");

System.out.println("3 x 7 = 21");

System.out.println("3 x 8 = 24");

System.out.println("3 x 9 = 27");

Für genau diesen Fall gibt es aber auch eine Schleife: die for-Schleife.
Sie ist immer dann sinnvoll einsetzbar,
wenn eine **bestimmte** Anzahl von Wiederholungen durchgeführt werden soll.

**for** ( int faktor = 1; faktor <= 9; faktor ++ )

{

 System.out.println("3 x " + faktor + " = " + 3\*faktor );

}

### Allgemeiner Aufbau

*// allgemeiner Aufbau*

**for** ( int Zählvariable mit Initialisierung; Abbruchbedingung; Inkrementierung) {

 *// Anweisung, die wiederholt werden soll*

}

### Anwendungsbeispiele

**for** ( int i = 1; i <= 10; i++ ) {} *// Hochzählen von 1 bis 10*

**for** ( int i = 1; i <= 10; i+=5 ) {} *// Hochzählen in 5er-Schritten*

**for** ( int i = 10; i >= 1; i-- ) {} *// Countdown von 10 bis 1*

## while-Schleife

In den meisten Programmiersprachen gibt es die **While-Schleife** als **Kontrollstruktur**.

* Sie dient dazu, eine Abfolge von Anweisungen mehrfach auszuführen, **solange** eine Bedingung erfüllt ist.
* Diese Bedingung wird geprüft, **bevor** die Anweisungsfolge abgearbeitet wird.
	+ Es kann also auch sein, dass die Abfolge gar nicht ausgeführt wird.
	+ Wenn die Bedingung ständig erfüllt ist, dann wird die Schleife zur **Endlosschleife**.
* Die While-Schleife verwendet man bei **dynamischen** Abbruchbedingungen, die [for-Schleife](https://unterrichten.zum.de/wiki/Java/Schleife%22%20%5Cl%20%22for-Schleife) hingegen bei einer festen Anzahl von Durchläufen.

### Beispiel: Fakultät

Das folgende Beispiel zeigt, wie sich die Berechnung der Fakultät n! durch eine while-Schleife realisieren lässt:

 **public** int fakultaet(int zaehler)

 {

 int fakultaet = 1;

 **while** (zaehler > 1) {

 fakultaet \*= zaehler;

 zaehler--;

 }

 **return** fakultaet;

 }

### Übungen mit while und Zufallszahlen

Übungen zu Zufallszahlen eignen sich, um die Interaktion zwischen verschiedenen Klassen zu demonstrieren. Die recht einfache Klasse Zufallszahl sieht wie folgt aus:

**public** **class** **Zufallszahl**

{

 **private** int zufallszahl;

 *// gibt eine Zufallszahl zwischen 1 und maximum zurück*

 **public** int getZufallszahl(int maximum)

 {

 **return** (int) (Math.random()\*maximum+1);

 }

}

Diese Klasse kann man nun auf einfache Weise in anderen Klassen verwenden. Eine Eigenschaft z.B. wuerfel vom Typ Zufallszahl wird vorbereitet (vgl. erste Zeilen).

Nun lassen sich mit wuerfel.getZufallszahl(6) Zufallszahlen zwischen 1 und 6 bestimmen, die man z.B. ausgeben kann (erste Methode) oder einer Variable zuweisen kann (zweite Methode).

**public** **class** **Wuerfelexperiment**

{

 **private** Zufallszahl wuerfel;

 **public** Wuerfelexperiment (){

 wuerfel=**new** Zufallszahl();

 }

 **public** void gibWuerfelAus(int anzahl)

 {

 int i=1;

 **while** (i<=anzahl){

 System.out.println("Wurf "+i+ " ist "+wuerfel.getZufallszahl(6));

 i++;

 }

 }

 **public** void wuerfeleBisZurSechs()

 {

 int i=1;

 *// Erster Wurf muss auf jeden Fall durchgeführt werden*

 int zwischenspeicher=wuerfel.getZufallszahl(6);

 System.out.println("Wurf "+i+ " ist "+zwischenspeicher);

 *// weitere Würfe*

 **while** (zwischenspeicher!=6){

 i++;

 zwischenspeicher=wuerfel.getZufallszahl(6);

 System.out.println("Wurf "+i+ " ist "+zwischenspeicher);

 }

 }

}

## do-while-Schleife

Eine **do-while-Schleife** ist in der Programmierung eine „*fußgesteuerte* oder *nachprüfende* Schleife, bei der nach dem Durchlauf des Schleifenrumpfes die Abbruchbedingung überprüft wird“[[1]](https://unterrichten.zum.de/wiki/Java/Schleife#cite_note-1) (meist als DO-WHILE, das entspricht *ausführen-während*).

### "Mensch ärgere dich nicht"

int wuerfel;

**do** {

 wuerfel=(int) (Math.random()\*6+1);

} **while** (wuerfel!=6);

dh. würfele solange, bis eine 6 gefallen ist, bzw. andersherum: wiederhole das Würfeln solange, solange das Würfelergebnis nicht 6 ist.

Dabei muss mindestens einmal gewürfelt werden.

### Nachfolgend finden sich einige Informationen von Serlo zu Arrays. Diese wurden unter der CreativeCommons-Lizenz CC BY-SA 4.0 veröffentlicht. Zu finden sind diese auch unter <https://de.serlo.org/informatik/143284/array>

# **Array**

Ein Array in Java ist

* ein Objekt, das der Speicherung von Werten eines bestimmten Datentypen dient.
* eine ein- oder mehrdimensionale Datenstruktur, bei der der Zugriff auf ein bestimmtes Element über Indizes erfolgt.

## Eindimensionaler Array

### Eigenschaften

Eindimensionale Arrays funktionieren prinzipiell wie eine einfache Liste mit Werten einer bestimmten Art. Sie haben jedoch eine unveränderbare Größe, die in dem Attribut *length*gespeichert ist. Es gibt keine Möglichkeit, Werte über die Maximalanzahl hinaus im Array zu speichern.

Der Index des Array gibt die Möglichkeit, auf bestimmte Werte des Array zuzugreifen. Er startet immer bei 0, sodass der letzte Indize der Länge *length*- 1 entspricht.

Hier eine beispielhafte Visualisierung eines Arrays für den Datentyp Integer mit der Länge 6:



### Deklaration

Es gibt mehrere Möglichkeiten, eine Variable in Java als Array zu deklarieren. Typisch sind jedoch die eckigen Klammern, die bei der Deklaration immer vorkommen. Vor der Variable, in der ein Array gespeichert werden soll, steht der Datentyp.

Deklaration zweier Integer-Arrays mit den Variablenamen array1 und array2. Es macht keinen Unterschied, ob die Klammern hinter dem Datentyp oder dem Variablenamen stehen:



Bei der Deklaration einer Variable als Array ist die Länge noch nicht festgesetzt. Die geschieht erst im Schritt der Initialisierung.

### Initialisierung

Bei der Initialisierung wird die Größe des Felds angegeben. Durch den Befehl "new" wird ein neues Objekt erzeugt. Im Folgenden wird das Array "liste" initialisiert.

liste = new int [10];

Wichtig: Das Array "liste" besitzt 10 Plätze. Dabei ist zu beachten, dass der erste Index 0 ist.

Den Indizes können nun Variablen zugewiesen werden. Hierbei muss bedacht werden, dass die Variablen dem Datentyp des Arrays entsprechen müssen. Im Folgenden werden die ersten 5 Primzahlen dem Array "liste" zugewiesen:

liste [0] = 2;

liste [1] = 3;

liste [2] = 5;

liste [3] = 7;

liste [4] = 11;

**Quellenverzeichnis**

Java-Editor: gui-verwendung, <https://javaeditor.org/doku.php?id=de:gui-verwendung#jtable>, zuletzt aberufen am 06.03.2023 14:15 Uhr. Veröffentlicht unter der Lizenz CC BY-SA 4.0: <https://creativecommons.org/licenses/by-sa/4.0/deed.de>

Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen: Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen, Informatik, 1. Auflage 2014. <https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/if/KLP_GOSt_Informatik.pdf>, zuletzt abgerufen am 06.03.2023 14:13 Uhr.

Serlo: Array, <https://de.serlo.org/informatik/143284/array>, zuletzt abgerufen am 06.03.2023 14:18 Uhr. Veröffentlicht unter der Lizenz CC BY-SA 4.0: <https://creativecommons.org/licenses/by-sa/4.0/deed.de>

zum Unterrichten: Algorithmik; bedingte Anweisungen, Operatoren, <https://unterrichten.zum.de/wiki/Java/Algorithmik#Vergleichsoperatoren>, zuletzt abgerufen am 06.03.2023 14:20 Uhr. Veröffentlicht unter der Lizenz CC BY-SA 4.0: <https://creativecommons.org/licenses/by-sa/4.0/deed.de>

zum Unterrichten: Schleife; for-Schleife, while-Schleife, do-while-Schleife, <https://unterrichten.zum.de/wiki/Java/Schleife>, zuletzt abgerufen am 06.03.2012 14:25 Uhr. Veröffentlicht unter der Lizenz CC BY-SA 4.0: <https://creativecommons.org/licenses/by-sa/4.0/deed.de>