JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
PROFESSUR BWL – WIRTSCHAFTSINFORMATIK
UNIV.-PROF. DR. AXEL SCHWICKERT

Schwickert, Axel C.; Odermatt, Sven; Bodenbender, Nicole;
Müller, Laura; Patzak, Maximilian; Döring, Mandy-Madeleine

Einführung in die Programmierung mit Java – Reader zur WBT-Serie

ARBEITSPAPIERE WIRTSCHAFTSINFORMATIK

Nr. 01/2015
ISSN 1613-6667

Arbeitspapiere WI Nr. 1 / 2015

Autoren: Schwickert, Axel C.; Odermatt, Sven; Bodenbender, Nicole; Müller, Laura; Patzak, Maximilian; Döring, Mandy-Madeleine

Titel: Einführung in die Programmierung mit Java – Reader zur WBT-Serie

Zitation: Schwickert, Axel C.; Odermatt, Sven; Bodenbender, Nicole; Müller, Laura; Patzak, Maximilian; Döring, Mandy-Madeleine: Einführung in die Programmierung mit Java – Reader zur WBT-Serie, in: Arbeitspapiere WI, Nr. 1/2015, Hrsg.: Professur BWL – Wirtschaftsinformatik, Justus-Liebig-Universität Gießen 2015, 67 Seiten, ISSN 1613-6667

Kurzfassung: Das vorliegende Arbeitspapier dient als Reader zur WBT-Serie „Programmierung mit Java – Teil 1“, die im E-Campus Wirtschaftsinformatik online zur Verfügung steht.

Java als Programmiersprache bietet eine Besonderheit im Vergleich zu klassischen Hochsprachen: Ein in Java geschriebenes Programm ist in seinem fertigen Zustand unabhängig von vielen Plattformen. Das heißt, ein Java-Programm kann auf fast jeder Kombination von Betriebssystem und Prozessor ausgeführt werden. Diese WBT-Serie führt in die Programmiersprache Java ein und erläutert Grundlegendes, wie benötigte Elemente von Programmiersprachen, bis hin zur praktischen Arbeit mit Java. Damit verbunden werden Variablen und Konstanten, Arrays und der modularen Programmierung erläutert.

Schlüsselwörter: Einführung in die Programmierung, Programme erstellen mit Java, Objektorientierte Programmierung

A Die Web-Based-Trainings

Der Lernstoff zum Themenbereich „Einführung in die Programmierung mit Java “ wird durch eine Serie von Web-Based-Trainings (WBT) vermittelt. Die WBT bauen inhaltlich aufeinander auf und sollten daher in der angegeben Reihenfolge und zum vorgesehenen Zeitpunkt absolviert werden. Um einen Themenbereich vollständig durchdringen zu können, muss jedes WBT mehrfach absolviert werden, bis die jeweiligen Tests in den einzelnen WBT sicher bestanden werden.

WBT-Nr.	WBT-Bezeichnung	Dauer	Bis wann bearbeitet?
1	Grundlagen der Programmierung	45 Min.	
2	Einführung in die Programmierung mit Java	45 Min.	
3	Strukturelemente und Befehle	45 Min.	
4	Variablen und Konstanten	45 Min.	
5	Operatoren	45 Min.	
6	Bedingte Anweisungen und Schleifen	45 Min.	
7	Arrays	45 Min.	
8	Methoden	45 Min.	
9	Modulare Programmierung	45 Min.	
10	Objektorientierung	45 Min.	

Übersicht der WBT-Serie

Die Inhalte der einzelnen WBT werden nachfolgend in diesem Dokument gezeigt. Alle WBT stehen Ihnen rund um die Uhr online zur Verfügung. Sie können jedes WBT beliebig oft durcharbeiten. In jedem WBT sind Quellcode-Beispiele enthalten, die Sie unbedingt nachbauen und ausführen sollten.

Inhaltsverzeichnis

Seite

Abbildungsverzeichnis

Seite

1 Einführung in die Programmierung

1.1 Computer und Programme

Computer werden häufig zum Lösen von wiederkehrenden Problemen eingesetzt. Um dem Computer mitzuteilen, wie er ein bestimmtes Problem lösen soll, werden Programme geschrieben. Ein Programm beinhaltet eine Folge von Befehlen. Diese Befehle sind eine genau definierte Abfolge von Schritten, die zur Lösung eines Problems führen sollen. Man spricht dabei auch von einem Algorithmus. Für das Ausführen des Algorithmus ist der Prozessor eines Computers verantwortlich. Der Prozessor liest die einzelnen Befehle und führt diese Schritt für Schritt aus. Damit der Prozessor die Befehle verstehen kann, müssen diese in Maschinencode vorliegen. Unter Windows haben Programme in Maschinencode häufig die Dateiendung „*.exe*“.

Maschinencode ist, einfach gesagt, eine hintereinander Reihung von Nummern (1-en und 0-en), wobei jede Nummer oder Nummernfolge für einen prozessor-spezifischen Befehl steht. Wie Programme in Maschinencode ausgeführt werden, lernen Sie im nächsten Kapitel.

1.2 Programme lesen und ausführen

Die Grundbausteine eines jeden Computers sind der Prozessor, der Arbeitsspeicher und die Festplatte. Diese Bausteine spielen die wesentliche Rolle beim Ausführen und Lesen von Programmen. Programme werden z. B. auf der Festplatte eines Computers gespeichert und jedes Mal, wenn sie ausgeführt werden sollen, vom Betriebssystem in den Arbeitsspeicher des Computers geladen. Nachdem ein Programm im Arbeitsspeicher liegt, kann der Prozessor auf dieses zugreifen und alle Befehle, die in Maschinencode geschrieben wurden, mit Hilfe des Betriebssystems ausführen.

Unglücklicherweise versteht nicht jede Kombination aus Betriebssystem und Prozessor den gleichen Maschinencode. Aus diesem Grund müssen Programme meistens in verschiedenen Versionen, die unterschiedlichen Maschinencode beinhalten, vorliegen, damit sie auf jedem Computer ausführbar sind. Allerdings ist es nahezu unmöglich, komplexe Programme oder Programm-Versionen in Maschinencode zu schreiben. Aus diesem Grund wurden Programmiersprachen wie Java erfunden.

1.3 Programmiersprachen

Programmiersprachen ermöglichen es, Programme in einer Sprache zu schreiben, die der Mensch einfacher erlernen und verstehen kann als Maschinencode. Die Befehle in einer Programmiersprache sind so definiert, dass man durch ihre Kombination wesentlich einfacher mitteilen kann, was der Computer bzw. Prozessor machen soll.

Wenn sich Befehle einer Programmiersprache in Form (Syntax) und Inhalt (Semantik) stark an der menschlichen Sprache orientieren, spricht man auch von einer Hochsprache. Java ist eine solche Hochsprache.

Einen Algorithmus, den man in einer Hochsprache schreibt, muss man in einer Datei speichern. Der Inhalt einer solchen Datei wird Quellcode genannt. Allerdings lässt sich Quellcode nicht mehr ohne Hilfe ausführen, denn der Prozessor versteht ja nur Maschinencode. Aus diesem Grund muss ein *Übersetzungsprogramm* verwendet werden, das aus Quellcode Maschinencode macht. Bei den *Übersetzungsprogrammen* kann zwischen *Compiler* und *Interpreter* unterschieden werden.

1.4 Der Compiler

Die Aufgabe des Compilers ist es, Quellcode in Maschinencode zu übersetzen. Der Compiler liest den Quellcode und erzeugt ein ausführbares Programm in Maschinencode. Dieses Programm wird auf der Festplatte oder einem anderen Datenträger gespeichert und kann beliebig oft durch einen Doppelklick oder einen anderen Befehl ausgeführt werden. Die Funktion des Compilers ist vergleichbar mit der eines Übersetzers, der ein englisches Buch liest, übersetzt und eine neue Ausgabe in Deutsch erzeugt.

1.5 Der Interpreter

Neben der Methode des Übersetzens durch einen Compiler gibt es noch ein Alternative - den Interpreter. Ein Interpreter übersetzt, wie der Compiler, Quellcode. Anstatt ein Programm zu erstellen, führt der Interpreter jede Anweisung, die er übersetzt, direkt aus. Im Vergleich zu einem klassischen Übersetzer ist der Interpreter ein Simultan-Dolmetscher, der alles was er liest direkt übersetzt und ausführt ohne ein neues Dokument zu erzeugen.

1.6 Compiler vs. Interpreter

Sowohl der Compiler als auch der Interpreter erfüllen die gleiche Aufgabe - beide übersetzen Quellcode. Obwohl beide Übersetzerprogramme im Ergebnis das Gleiche tun, hat sowohl der

Compiler als auch der Interpreter seine Vor- und Nachteile. Diese werden im Folgenden kurz aufgeführt:

- **Kompilierte Programme sind schneller:** Programme, die mit einem Compiler erzeugt werden, lassen sich schneller ausführen als interpretierte Programme. Ein kompiliertes Programm liegt in Maschinencode vor und muss lediglich vom Prozessor gelesen werden.
- **Kompilierte Programme sind schwer zu pflegen:** Wird der Quellcode eines kompilierten Programms verändert, geschieht zunächst nichts. Solange der Quellcode nicht zu einem neuen Programm kompiliert wurde, steht die Änderung nicht im Maschinencode und der Prozessor kann sie auch nicht ausführen.
- **Interpretierte Programme sind pflegeleicht:** Der Quellcode eines interpretierten Programms wird jedes Mal zur Laufzeit neu übersetzt. Das bedeutet, jede Änderung am Quellcode wird direkt in Maschinencode umgewandelt und ausgeführt.
- **Interpretierte Programme sind langsamer:** Da ein interpretiertes Programm zur Laufzeit übersetzt und dann ausgeführt wird, sind diese langsamer als kompilierte Programme. Ein kompiliertes Programm liegt bereits in Maschinencode vor und muss lediglich ausgeführt werden.

1.7 Allgemeines über Java

Jetzt wissen Sie bereits grob, was es mit Programmen und Programmiersprachen auf sich hat und dass Java eine Hochsprache ist. Java hat zudem eine Besonderheit im Vergleich zu klassischen Hochsprachen: Ein in Java geschriebenes Programm ist in seinem fertigen Zustand unabhängig von vielen Plattformen. Das heißt, ein Java-Programm kann auf fast jeder Kombination von Betriebssystem und Prozessor ausgeführt werden. Um diesen Zustand zu erreichen, wird der Quellcode eines Java-Programms aus einer Kombination von Compiler und Interpreter (in Java heißt er "Java Virtual Machine") übersetzt. Der Java-Compiler erzeugt aus Java-Quellcode keinen prozessor-spezifischen Maschinencode, sondern sogenannten Bytecode (Dateien, in denen Bytecode steht, haben die Endung „.class“). Bytecode ist ein unabhängiger Maschinencode, der nicht direkt von einem Prozessor ausgeführt werden kann. Stattdessen muss der Bytecode eines Java-Programms von der Java Virtual Machine (JVM) eingelesen und ausgeführt werden. Die JVM ist der Interpreter in Java und Bestandteil jeder Java-Installation. Da Java für die gängigsten Betriebssystem-Prozessor-Kombinationen bereitgestellt wird, kann auch Java-Bytecode auf den meisten Computern ausgeführt werden.

1.8 Die Vorteile von Java-Bytecode

Sie wissen jetzt, dass Java-Bytecode von der Java Virtual Machine ausgeführt wird. Bevor Sie mit den Vorbereitungen für die Erstellung ihres ersten eigenen Programms beginnen, wird hier noch einmal aufgeführt, welche Vorteile aus der Kombination von Java-Compiler und Java Virtual Machine entstehen:

- **Geschwindigkeit:** Der Java-Compiler bereitet den Java-Bytecode so vor, dass er wesentlich schneller von der JVM interpretiert werden kann als normaler Quellcode.
- **Einmal schreiben, überall ausführen:** Ein kompiliertes Java-Programm kann auf jedem Computer ausgeführt werden, wenn die Java Virtual Machine installiert ist. Da für die gängigsten Betriebssystem-Prozessor-Kombinationen die JVM zur Verfügung steht, können kompilierte Java-Programme (also Java-Bytecode) auf fast allen Computern ausgeführt werden. Mehrere Programm-Versionen für unterschiedliche Betriebssysteme und Prozessoren sind nicht nötig (diese Eigenschaft wird *Plattformunabhängigkeit* genannt)

2 Erste Schritte mit Java

2.1 Vorbereitung zum Programmieren

Bevor Sie in die Programmierung Ihres ersten Java-Programms einsteigen, müssen Sie zwei Entwicklungswerkzeuge installieren. Das *Java Development Kit (JDK)* und einen *Texteditor*. Das JDK ist das Herzstück einer Java-Installation und der Texteditor wird zum Schreiben des Quellcodes benötigt.

- **Das Java Development Kit**

  Das Java Development Kit (JDK) beinhaltet den Java-Compiler und die Java Virtual Machine. Laden Sie das JDK unter folgendem Link herunter:

  http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

  Stellen Sie sicher, dass sie das JDK passend für ihr Betriebssystem herunterladen! Installieren Sie das JDK auf der Festplatte *C:/*. Unter folgenden Link finden Sie eine Installationsanleitung:

  http://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_jdk_install.hml

- **Der Texteditor**

  Für die Programmierung ihres ersten Programms genügt ein einfacher Texteditor. Mögliche Optionen sind der "*Editor*" unter Microsoft Windows oder der "*TextEditor*" unter Mac OS.

2.2 Systemvariablen anpassen

Die Installation des JDK haben Sie abgeschlossen. Jetzt müssen Sie die Systemvariablen anpassen. Führen Sie die folgenden Schritte durch:

1. Wählen Sie im Startmenü den Menüpunkt *"Computer"*
2. Wählen Sie die Option *"Systemeigenschaften"*
3. Klicken Sie auf *"Erweiterte Systemeinstellungen"*
4. Klicken Sie auf *"Umgebungsvariablen"*. Wählen Sie unter *"Systemvariablen"* PATH, und klicken Sie auf die Option *"Bearbeiten"*.
5. Fügen Sie den Pfad C:\Program Files\Java\jdk<*Version*>\bin hinzu. Ersetzen Sie <*Version*> durch die auf Ihrem Computer installierte JDK-Version (z. B "C:\Program Files\Java\jdk1.8.0_25\bin"). Sollte der Pfad bereits vorhanden sein, können Sie diese Seite überspringen.

6. Falls die Systemvariable PATH nicht vorhanden ist, können Sie eine neue Variable wählen und PATH als Name und den Pfad C:\Program Files\Java\jdk<*Version*>\bin als Wert hinzufügen.

2.3 Die Ordnerstruktur anlegen

Nachdem das JDK und der Texteditor eingerichtet wurden, müssen Sie eine Ordnerstruktur anlegen, denn Java-Programme sollten nicht einfach willkürlich auf der Festplatte des Computers abgespeichert werden. Legen Sie sich bitte ein Projektverzeichnis *C:/workspace* an. Erstellen Sie in *workspace* für jedes WBT einen neuen Ordner, in dem Sie dann die Programme nachbauen, die Ihnen hier und in den folgenden WBT gezeigt und erklärt werden. Für das aktuelle WBT sollte der Ordner *C:/workspace/a01* angelegt werden. Für das folgende WBT *a02* und so weiter.

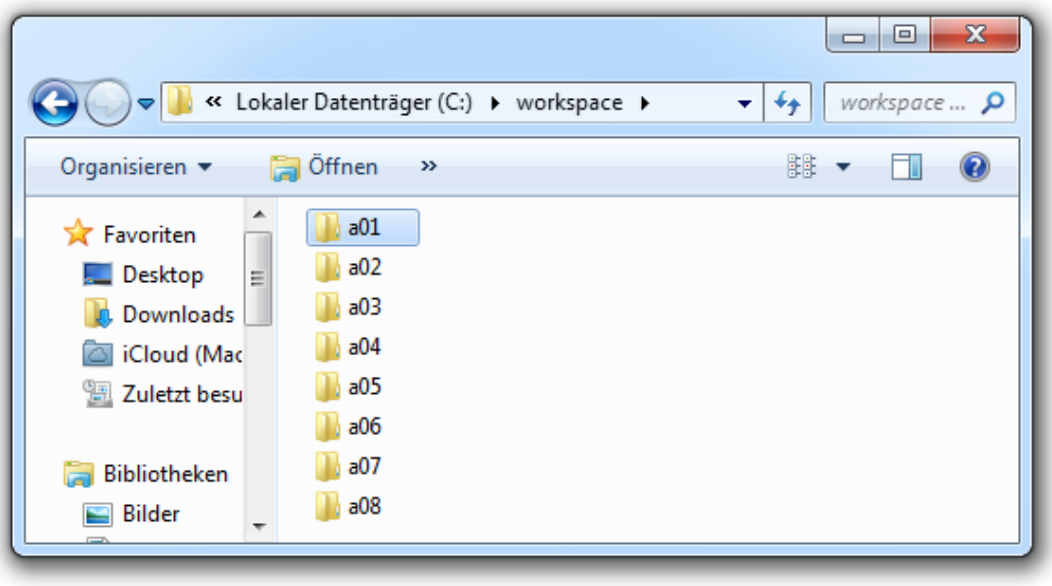

Abb. 1: Ordnerstruktur

2.4 Den ersten Java-Quellcode erstellen

Jetzt kann es mit der Programmierung des ersten Java-Programms losgehen. Wie Sie gelernt haben, müssen Sie mit der Erstellung des Java-Quellcodes beginnen. Das erste Java-Programm soll den Nutzer mit dem Satz "Hallo, Welt!" begrüßen. Dafür muss der folgende Quellcode geschrieben werden:

```
public class HalloWelt{
  public static void main(String[] args){
    System.out.println("Hallo, Welt!");
  }
}
```

Öffnen Sie den Texteditor und geben Sie den Quellcode ein. Speichern Sie den Quellcode in der Datei „HalloWelt.java“. Wählen Sie als Speicherort das Verzeichnis *C:/workspace/a01*.

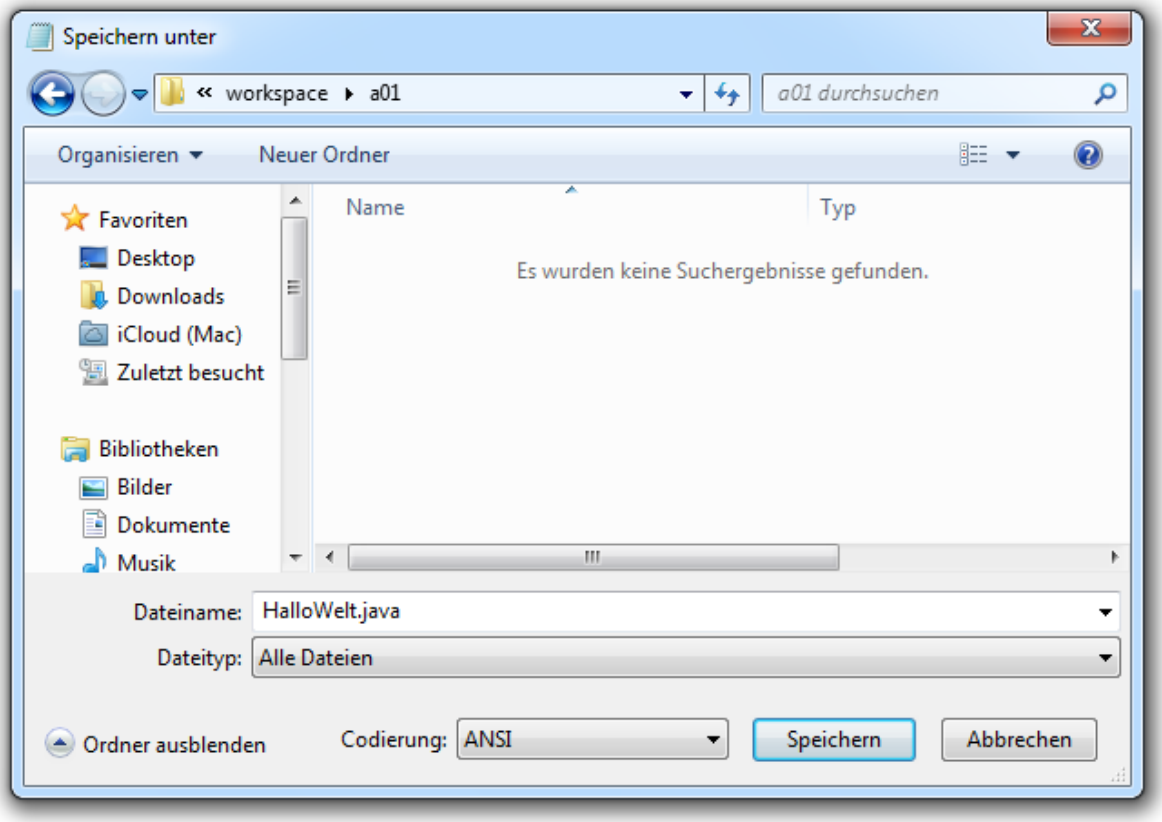

Abb. 2: „HalloWelt.java“ speichern

2.5 Den ersten Java-Bytecode erstellen

Nachdem Sie den Quellcode (in HalloWelt.java) erstellt haben, müssen Sie diesen in Bytecode umwandeln. Dafür ist der Java-Compiler zuständig. Der Java-Compiler wird über die Konsole aufgerufen und bedient. Aus diesem Grund müssen die folgenden Schritte in der Konsole (Windows: „*Eingabeaufforderung*“; Mac OS: „*Terminal*“) durchgeführt werden:

1. Öffnen Sie die Konsole und geben Sie folgenden Befehl ein: *cd C:/workspace/a01*. Mit diesem Befehl wechseln Sie in den Ordner *a01*. In *a01* haben Sie die Datei mit Java-Quellcode gespeichert.
2. Jetzt können Sie den Java-Compiler aufrufen und den Java-Quellcode in Bytecode umwandeln. Mit dem folgenden Befehl erzeugt der Java-Compiler (er heißt "javac") das erstes Java-Programm: *javac HalloWelt.java*
3. Überprüfen Sie, ob der Java-Compiler eine class-Datei erstellt hat. Im Verzeichnis *C:/workspace/wbt02* sollte jetzt „HalloWelt.class“ gespeichert sein. Die class-Datei „HalloWelt.class“ ist ein ausführbares Java-Programm in Bytecode.

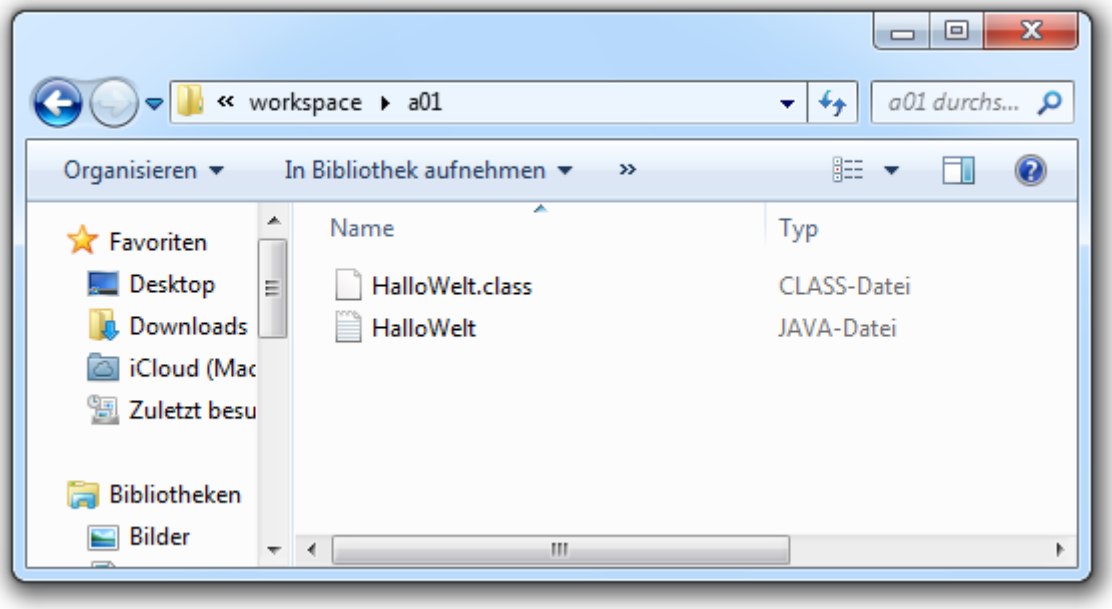

Abb. 3: „HalloWelt.class“ erstellen

2.6 Das erste Java-Programm ausführen

Mit der Datei „HalloWelt.class“ haben Sie das erste Java-Programm in Bytecode erzeugt. Um es auszuführen, müssen Sie die Java Virtual Machine aufrufen. Die Java Virtual Machine muss, genau wie der Java-Compiler, über die Konsole geöffnet und bedient werden:

1. Öffnen Sie die Konsole und wechseln Sie in das Verzeichnis *a01*: *cd C:/workspace/a01*.
2. Geben Sie den folgenden Befehl ein, um mit der Java Virtual Machine den Bytecode des Java-Programms in der Datei „HalloWelt.class“ auszuführen: *java HalloWelt*
3. Nach Bestätigen des Befehls mit der Enter-Taste, sollte auf der Konsole der Satz "Hallo, Welt!" erscheinen. Erscheint der Satz, haben Sie es geschafft. Sie haben ihr erstes Java-Programm geschrieben, kompiliert und ausgeführt.

2.7 Lessons learned

Java-Quellcode muss in einer Datei mit der Endung ".java" stehen. Man nennt diese Datei auch Quellprogramm. Mit *javac HalloWelt.java* wandelt man den Quellcode in Bytecode um. Es entsteht eine class-Datei „HalloWelt.class“. class-Dateien sind Java-Programme in Bytecode. Die class-Dateien können von der Java Virtual Machine ("java") gelesen und ausgeführt werden. „HalloWelt.class“ wird z. B. mit *java HalloWelt* ausgeführt werden.

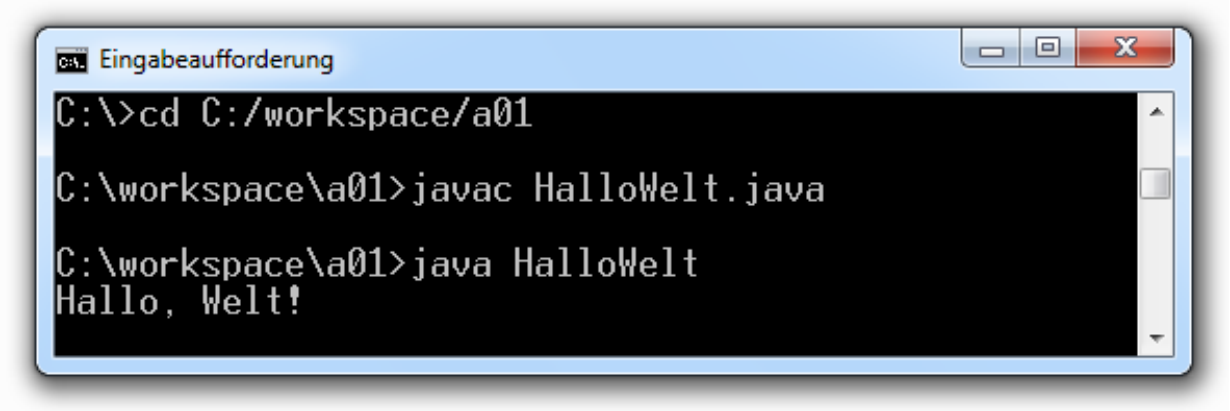

Abb. 4: Konsolenbefehle im Überblick

3 Strukturelemente und Befehle

3.1 Die Rolle der Klasse

Wenn man ein Java-Programm entwickeln möchte, muss man den Quellcode für dieses Programm in eine Java-Datei (HalloWelt.java) schreiben. Diese Datei kann dann kompiliert und ausgeführt werden. Beim Schreiben des Quellcodes muss man immer einer fest vorgegebenen Struktur folgen:

```
public class HalloWelt{
  public static void main(String[] args){
    System.out.println("Hallo, Welt!");
  }
}
```

Ganz am Anfang des Quellcodes muss eine Klassendefinition stehen. In der Klasse können Methoden angelegt werden, die Befehle beinhalten. Damit die Java-Datei als Programm ausgeführt werden kann, muss mindestens die sogenannte main-Methode in der Klasse vorhanden sein. Auf den folgenden Seiten werden die einzelnen Strukturelemente (Klasse, Methode), des hier als Beispiel verwendeten HalloWelt-Programms, analysiert.

3.2 Die HalloWelt-Klasse

Der Java-Quellcode in einer Java-Datei muss immer mit einer Klassendefinition eingeleitet werden. Warum in Java alles innerhalb von Klassen organisiert ist, lernen Sie in späteren Kapiteln. Wichtig ist, dass Sie sich den Aufbau einer Klasse merken:

```
public class HalloWelt{


}
```

Die Klassendefinition des vorliegenden Java-Programms beginnt in der ersten Zeile und wird mit den Wörtern `public` und `class` eingeleitet. Anschließend folgt der Name der Klasse. Hier heißt die Klasse `HalloWelt`. Wichtig ist, dass man einen Klassennamen immer mit einem großen Buchstaben beginnt. Zudem muss eine Klasse immer so benannt sein, wie die Java-Datei, in der die Klasse definiert wird.

Der Beginn und das Ende einer Klassendefinition wird mit einer geschweiften Klammer { } gekennzeichnet. Innerhalb der geschweiften Klammern können Methoden formuliert werden.

Eine Klasse beinhaltet meistens eine oder mehrere Methoden. Wie Methoden aufgebaut sind und was sie bewirken, lernen Sie im nächsten Kapitel.

3.3 Aufbau einer Methode

Methoden sind Strukturierungselemente in Java, die als Behälter für Befehle eingesetzt werden. Alle Befehle, die eine Klasse bereitstellen soll, müssen in Methoden definiert sein. Eine Methode besteht immer aus einem Methodenkopf und einem Methodenkörper und muss in einer Klasse angelegt werden. Das heißt, sie muss zwischen den geschweiften Klammern der Klassendefinition stehen:

```
public class HalloWelt{
  public static void main(String[] args){

  }
}
```

Ein Methodenkopf enthält die Worte `public` und `static`, den Namen der Methode und Parameter. Hier heißt die Methode `main` und hat `String[] args` als Parameter. Ein Methodenname wird immer mit einem kleinen Buchstaben eingeleitet. Der Methodenkörper kann Befehle enthalten, die von der Methode ausgeführt werden sollen. Beginn und Ende des Methodenkörpers werden durch geschweifte Klammern { } gekennzeichnet.

Die Klasse `HalloWelt` beinhaltet die Methode mit dem Namen `main`. Diese Methode erfüllt eine besondere Rolle in Java-Programmen und wird im Folgenden noch einmal detailliert erläutert.

3.4 Die Rolle der main-Methode

Eine Java-Datei und die darin enthaltene Klasse kann nur als eigenständiges Programm ausgeführt werden, wenn sie eine main-Methode beinhaltet.

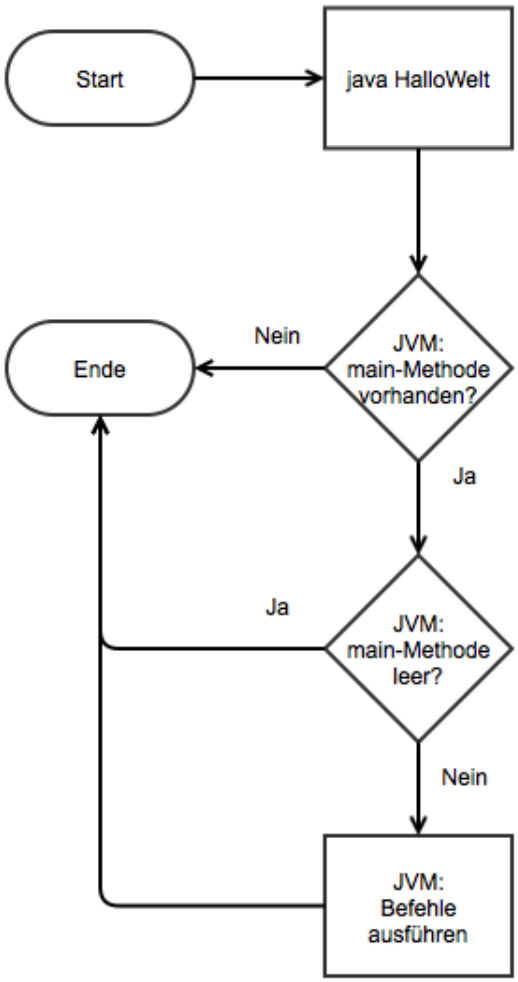

Abb. 5: Die Aufgabe der main-Methode

Eine main-Methode wird immer mit `public static void main (String[] args)` deklariert. Anschließend folgt der Methodenkörper. Im Methodenkörper der main-Methode muss nacheinander beschrieben werden, was das Programm machen soll. Um das zu tun, verwendet man Befehle. Die main-Methode der vorliegenden HalloWelt-Klasse beinhaltet einen Befehl:

```
System.out.println("Hallo, Welt!");
```

An dieser Stelle ist es wichtig, dass Sie sich den Aufbau der main-Methode genau merken. Was die einzelnen Sprachelemente (`public, static, void`) bedeuten, lernen Sie in späteren Kapiteln.

3.5 Der Ausgabebefehl

Mit dem Befehl `System.out.println();` kann man Sätze auf dem Bildschirm - genauer gesagt in der Konsole - anzeigen. In der Klammer von `println()` muss in Anführungszeichen stehen, was angezeigt werden soll.

Der Nutzer des vorliegenden Programms sollte mit "Hallo, Welt!" begrüßt werden, deshalb wurde `System.out.println("Hallo, Welt!");` in den Methodenkörper der main-Methode geschrieben. Diesen Befehl kann man auch beliebig oft wiederholen und nacheinander unterschiedliche Sätze oder Worte auf dem Bildschirm anzeigen:

```
public class HalloWelt{
  public static void main(String[] args){
    System.out.println("Hallo, Welt!");
    System.out.println("Mein Name ist");
    System.out.println("Max Muster");
  }
}
```

Abb. 6: Erweitertes HalloWelt-Programm

3.6 Kommentieren und Einrücken

Nachdem man Neues gelernt hat, ist es sinnvoll sich Notizen zu machen. Das geht nicht nur auf Papier, sondern auch im Quellcode. Notizen im Quellcode werden Kommentare genannt. Kommentare können in zwei verschiedenen Arten in Java-Quellcode eingebaut werden. Man kann einzeilige Kommentare oder Kommentarblöcke anlegen:

```
/* Das hier ist ein Kommentarblock für die Klasse HalloWelt.
HalloWelt liegt in der Datei HalloWelt.java und beinhaltet eine main-
Methode und Befehle. */
public class HalloWelt{
  //main-Methode
  public static void main(String[] args){
    //Befehle
    System.out.println("Hallo, Welt!");
    System.out.println("Mein Name ist");
    System.out.println("Max Muster");
  }
}
```

Ein einzeiliger Kommentar wird mit // eingeleitet. Alles was dahinter steht, wird vom Compiler ignoriert. In der nächsten Zeile geht es aber ganz normal weiter. Möchte man länger Kommentare schreiben, kann man einen Kommentarblock erstellen. Ein Kommentarblock wird mit /* geöffnet und mit */ geschlossen.

Eine weitere Maßnahme zur übersichtlichen Programmierung ist das Einrücken von Quellcode. Die Entwickler von Java schreiben in sogenannten Quellcode-Konventionen vor, dass man alles, was innerhalb von geschweiften Klammern { } passiert, einrücken soll. Gerade am Anfang kommt es häufig vor, dass Klammern vergessen werden. Durch konsequentes Einrücken sind solche Fehler gut sichtbar und außerdem erleichtern die verschiedenen Einrücktiefen das Lesen des Quellcodes wesentlich.

3.7 Lessons learned

Ein Java-Programm besteht aus einer Java-Datei, in der eine Klasse definiert werden muss, die wie die Java-Datei benannt ist. Damit die Datei als Programm ausgeführt werden kann, muss die Klasse eine Methode `main` beinhalten. Die main-Methode wird mit `public static void main(String[] args)` innerhalb der Klasse definiert. Die Ausführung des Programms (java HalloWelt) beginnt immer mit dem ersten Befehl innerhalb der main-Methode und endet mit dem letzten Befehl.

4 Variablen und Konstanten

4.1 Die Rolle der Variablen

Damit man mit einer Klasse nicht nur Texte ausgeben kann, sondern auch komplexere Problemstellungen bearbeiten kann, muss man Variablen anlegen. Eine Variable bezeichnet einen Speicherplatz, in dem Zahlen oder Buchstaben abgespeichert werden. In Java werden Variablen z. B. in Methoden einer Klasse angelegt:

```
//Datei Wetter.java
public class Wetter{
  public static void main(String[] args){
    double temperatur = 30.9;  //legt die Variable temperatur an
  }
}
```

Auf den folgenden Seiten werden Sie lernen, wie Variablen in der main-Methode deklariert, initialisiert und verwendet werden.

4.2 Variablen anlegen

Variablen werden mit einer Variablendefinition angelegt und mit Werten bestückt. Eine Variablendefinition beginnt mit einem Datentyp, gefolgt vom Namen der Variablen. Wenn man den Datentyp und Namen in den Quellcode geschrieben hat, spricht man von einer Deklaration. Ab jetzt weiß Java, dass es die Variable gibt:

```
//Datei Stundent.java
public class Student{
  public static void main(String[] args){
    int matrikelnummer = 1234567; //Deklaration & Initialisierung
    int alter; //Deklaration
    //Quellcode
    alter = 25; //Initialisierung
  }
}
```

In der vorliegenden Klasse `Student` wurden die Variablen `matrikelnummer` und `alter` mit dem Datentyp `int` deklariert. Der Name einer Variablen muss immer mit einem kleinen Buchstaben beginnen.

Damit die Variablen eine sinnvolle Aufgabe im Programm erfüllen können, wird ihnen ein Wert zugewiesen. Die erstmalige Zuweisung eines Wertes nennt man Initialisierung. Die

Initialisierung erfolgt durch ein Gleichzeichen und kann auf zwei Arten erfolgen: Entweder direkt in der Zeile, in der die Variable bekanntgegeben wurde oder in einem späteren Abschnitt des Quellcodes.

Welche Rolle der Datentyp `int` bei der Definition spielt, werden Sie im nächsten Kapitel lernen.

4.3 Einfache Datentypen

Jedes mal, wenn Sie eine Variable anlegen, müssen Sie mit einem Datentyp bestimmen, welcher Wert in der Variablen abgelegt werden soll. Dafür wurde in der Klasse zuvor `int` benutzt. Neben dem Datentyp `int` gibt es noch andere Datentypen, die für Variablen genutzt werden können.

Datentyp	Wertebereich	Wertetyp
boolean	True , False	Wahrheitswert
char	90.000 Zeichen (Unicode)	Buchstaben und Zeichen
byte	-128 bis +127	Ganze Zahl
short	-32.768 bis +32.767	Ganze Zahl
int	-2.147.483.648 bis +2.147.483.647	Ganze Zahl
long	-9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807	Ganze Zahl
float	$+/- 10^{38}$	Fließkomma-zahl
double	$+/- 10^{308}$	Fließkomma-zahl

Abb. 7: Einfache Datentypen

Mit diesen sogenannten *einfachen Datentypen* kann man ganze Zahlen, Fließkommazahlen oder einzelne Zeichen in einer Variablen speichern. Wenn man eine Zahl speichern will, kann man zwischen `byte, short, int, long, float` oder `double` wählen:

```
//Quellcode
    boolean wahr = true;
    char zeichen = "a";
    byte alter = 100;
    short jahr = 2015;
    int nummer = 061182312;
    long bevoelkerung = 7200000000;
    float temperatur = 30.90f;
    double PI = 3.14159265359;
```

Variablen vom Typ `int` können z. B. ganze Zahlen, die größer als -2.147.483.648 und kleiner als +2.147.483.648 sind, speichern. `double` und `float` erlauben dagegen das Abspeichern wesentlich größerer Zahlen, inklusive Komma. Wenn Sie Kommazahlen in einer Variable vom Typ `float` speichern wollen, müssen Sie immer ein `f` hinter den Wert schreiben (z. B. `float temperatur = 30.9f;`).

4.4 Der Datentyp String

Ein besonderer Datentyp ist der Datentyp `String`. Variablen, die mit dem Datentyp `String` angelegt werden, können Zeichenketten speichern. Eine Zeichenkette kann ein einfaches Zeichen sein, mehrere Zeichen, Wörter oder ganze Sätze. Um eine Variable mit dem Datentyp `String` zu deklarieren, schreiben Sie erst den Datentyp `String` hin und dann folgt der Name:

```
String zeichenkette = "Hallo, Welt!";
```

Ein `String` wird, wie einfachen Datentypen, mit einem Gleichzeichen initialisiert. Zeichenketten, die in einem String gespeichert werden sollen, müssen immer in Anführungszeichen stehen.

4.5 Variablen ändern

Der Wert einer Variablen kann beliebig oft geändert werden. In den meisten Programmen werden Variablen nicht angelegt, um im Ablauf des Programms den selben Wert zu behalten. Wie der Name schon sagt, ist der Speicherplatz einer Variablen „variabel“. Um den Wert zu ändern, muss mit einem Gleichzeichen ein neuer Wert zugewiesen werden:

```
//Datei Zuweisung.java
public class Zuweisung{
  public static void main(String[] args){
    byte ersteZahl = 25;
    byte zweiteZahl = 2;
    zweiteZahl = ersteZahl;
    ersteZahl = 18;
  }
}
```

Mit einem Gleichzeichen wird hier der Variablen `ersteZahl` die 18 neu zugewiesen. Alternativ besteht die Möglichkeit, den Wert einer bereits vorhanden Variablen zuzuweisen. Durch den Befehl `zweiteZahl = ersteZahl;` wird in der Variablen `zweiteZahl` der Wert von `ersteZahl`, also 18, gespeichert.

4.6 Konstanten

Wie Sie gelernt haben, lässt sich der Wert einer Variablen durch Zuweisungen verändern. Bei manchen Variablen ist es jedoch notwendig, die Zuweisung eines Wertes nach der Initialisierung zu verbieten:

```
//Datei Konstanten.java
public class Konstanten{
  public static void main(String[] args){
    final double PI = 3.141592654;
    final byte MWST = 19;
    PI = 23; //Verboten
  }
}
```

Um zu verhindern, dass der Wert einer Variablen verändert wird, verwendet man das Wort `final`. Wird eine Variable mit `final` deklariert, spricht man nicht mehr von einer Variablen, sondern von einer Konstanten. Im Gegensatz zu Variablen sollte eine Konstante immer in der Zeile in der sie deklariert wird auch initialisiert werden. Der Name einer Konstanten wird ausschließlich in großen Buchstaben geschrieben.

4.7 Variablen und der Ausgabebefehl

Variablen können auf dem Bildschirm dargestellt werden. Dafür muss der Name einer Variablen in die Klammer des Befehls `System.out.println();` geschrieben werden:

```
//Datei Ausgabe.java
public class Ausgabe{
  public static void main(String[] args){
    final double PI = 3.141592654;
    System.out.println("PI hat den Wert: " +PI);
    System.out.println(PI+ " ist der Wert von PI");
    System.out.println("Der Wert " +PI+ " ist PI");
  }
}
```

Mit einem Pluszeichen (+) kann die Variable mit beliebigem Text kombiniert werden. Steht die Variable zwischen zwei Text-Elementen, muss das Plus (+) auf beiden Seiten des Variablennamens stehen. Wichtig ist, dass man nie vergisst, den Text in Anführungszeichen zu stellen. Wenn man die Anführungszeichen vergisst, denkt Java, dass es sich um eine Variable handelt.

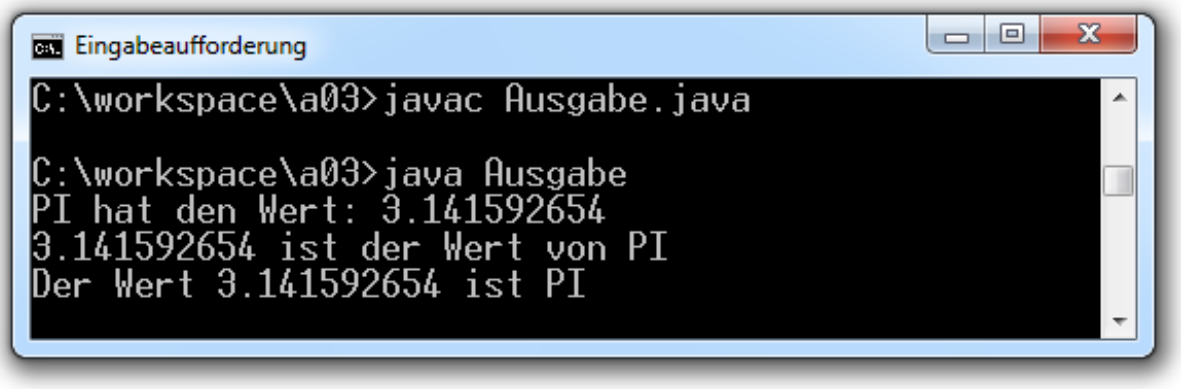

Abb. 8: Variablen im Ausgabebefehl

4.8 Lessons learned

Eine Variable muss immer mit einem Datentyp, gefolgt von einem Namen, deklariert werden:

```
public class BeispielKlasse{
  public static void main(String[] args){
    //Variable
    int zahle = 13;
    //Konstante
    final double PI = 3.141592654;
  }
}
```

Die Zuweisung eines Wertes erfolgt durch ein Gleichzeichen (=) und heißt Initialisierung. Solange eine Variable nicht mit dem Wort `final` als Konstante definiert wurde, kann der Wert durch Zuweisungen verändert werden. Nachdem Sie nun wissen, was Variablen und Konstanten sind, werden Sie im nächsten Kapitel lernen, wie mit Variablen mathematische Berechnungen und Vergleiche durchgeführt werden können.

5 Operatoren

5.1 Die Rolle der Operatoren

Bis jetzt wurden Programme geschrieben, die in der main-Methode Variablen deklariert, initialisiert und dann auf dem Bildschirm ausgegeben haben. Damit ein Programm bestimmte Aufgaben oder Probleme mit diesen Variablen lösen kann, werden Operatoren eingesetzt.

Operatoren werden für Berechnungen und Vergleiche verwendet. Auf den folgenden Seiten werden Sie lernen, welche Operatoren es gibt und wie diese zum Berechnen und Vergleichen von Variablen in der main-Methode einer Klasse eingesetzt werden:

```
public class Operatoren{
  public static void main(String[] args){
    byte i = 2;
    byte j = 3;
    System.out.println((i + j) * 3); //Berechnung
    System.out.println(j < i);       //Vergleich
  }
}
```

5.2 Arithmetische Operatoren

Arithmetische Operatoren können Variablen addieren, subtrahieren, multiplizieren und dividieren:

Operator	Beschreibung	i	j	Ergebnis
+	Addiert i und j.	10	6	16
-	Subtrahiert j von i.	10	4	6
*	Multipliziert i mit j.	10	9	90
/	Dividiert i mit j.	10	2	5
%	Dividiert i mit j und liefert den Restwert.	10	4	2

Abb. 9: Arithmetische Operatoren

Für arithmetische Operationen müssen Variablen mit dem Datentypen `byte`, `short`, `int`, `long`, `float` oder `double` verwendet werden. Das Ergebnis einer arithmetischen

Operation kann in einer Variablen gespeichert werden. Alternativ ist es möglich, eine arithmetische Operation direkt in einen Befehl zu schreiben:

```
public class Arithmetik{
  public static void main(String[] args){
    int ergebnis_plus = 10 + 6;   //16
    int ergebnis_minus = 10 - 4;  //6
    int ergebnis_mal = 10 * 9;    //90
    int ergebnis_durch = 10 / 2;  //5
    int modulo = 10 % 4;          //2
    System.out.println("Ergebnis 1: " +ergebnis_plus);
    //usw.
    System.out.println("Ergebnis 6: " + ((10 * 5) /2));
  }
}
```

Mit Klammern kann man Prioritäten setzen. Mit `((10 * 5) /2)` wird z. B. `10` mit `5` multipliziert und das Ergebnis der Multiplikation dann durch `2` geteilt. In der vorliegenden Klasse wurden alle arithmetischen Operationen direkt mit Zahlen durchgeführt, es ist aber auch möglich Variablen zu verwenden:

```
int i = 13; int j = 5;
int ergebnis = i * j;
```

5.3 Vergleichsoperatoren

Variablen oder Zahlen können gleich, größer, größer gleich, kleiner oder kleiner gleich als andere Variablen oder Zahlen sein. Vergleichsoperatoren überprüfen diese Fälle.

Operator	Beschreibung	i	j	Ergebnis
==	Testet, ob der Wert von i gleich j ist.	10	10	True
		10	9	False
!=	Testet, ob der Wert von i ungleich j ist.	10	10	False
		10	9	True
>	Testet, ob der Wert von i größer j ist.	10	9	True
		9	9	False
>=	Testet, ob der Wert von i größer gleich j ist.	10	10	True
		10	11	False
<	Testet, ob der Wert von i kleiner j ist.	10	9	False
		9	10	True
<=	Testet, ob der Wert von i kleiner gleich j ist.	10	11	False
		9	10	True

Abb. 10: Vergleichsoperatoren

Vergleichsoperatoren können mit einfachen Datentypen arbeiten. Das Ergebnis einer Vergleichsoperation ist ein Wahrheitswert. Ein Wahrheitswert kann nur den Wert `true` (wahr) oder `false` (falsch) annehmen. Wenn ein Vergleich positiv ist, ist das Ergebnis `true`. Fällt ein Vergleich negativ aus, ist das Ergebnis `false`:

```
public class Vergleiche{
  public static void main(String[] args){
    byte i = 10; byte j = 11;
    System.out.println(i == j);      //false
    System.out.println(i != j);      //true
    System.out.println(i > j);       //false
    System.out.println(i >= j);      //false
    System.out.println(i < j);       //true
    System.out.println(i <= j);      //true
  }
}
```

Wenn man einen Vergleich nicht direkt ausgeben will, sondern in einer Variablen speichern möchte, muss man den Datentyp `boolean` verwenden. Der Datentyp `boolean` kann nur den Wert `true` oder `false` speichern und sonst keinen:

```
boolean vergleich = i != j; //true;
```

5.4 Logische Operatoren

Vergleichsoperatoren produzieren Wahrheitswerte (`true` oder `false`). Logische Operatoren werden zum Weiterverarbeiten dieser Wahrheitswerte verwendet.

Operator	Beschreibung	i	j	Ergebnis
&&	Testet, ob i UND j wahr sind.	True	True	True
		True	False	False
		False	False	False
		False	True	False
||	Testet, ob i ODER j wahr ist.	True	False	True
		False	False	False
		False	False	False
		False	True	True
^	Testet, ob i ODER j NICHT wahr ist.	True	True	False
		True	False	True
		False	False	True
		False	True	True
!	Testet ob i NICHT wahr ist.	True	-	False
		False	-	True

Abb. 11: Logische Operatoren

Das Ergebnis logischer Operationen ist wieder ein Wahrheitswert. In der vorliegenden Klasse werden die Vergleiche `10 == 10` und `9 != 9` mit logischen Operatoren weiterverarbeitet:

```
public class Logik{
  public static void main(String[] args){
    boolean i = 10 == 10;
    boolean j = 9 != 9;
    System.out.println(i&&j); //false
    System.out.println(i||j); //true
    System.out.println(!i);   //false
    System.out.println(i^j);  //true
  }
}
```

Der logische Operator „`&&`“ testet, ob der Wert von `i` UND von `j true` ist. Sind beide Werte in den Variablen `true,` ist auch das Ergebnis des logischen Vergleichs `true`, andernfalls ist es `false`. Mit „`||`“ wird getestet, ob die Variable `i` ODER `j` den Wert `true` hat. Auch hier liefert ein positiver Vergleich den Wert `true` und ein negativer den Wert `false`. Das

Ausrufezeichen „`!`“ überprüft, ob der Wert von `i` NICHT `true` ist und das Hütchen „^“ testet, ob `i` ODER `j` NICHT `true` sind.

5.5 Lessons learned

Damit Variablen für Berechnungen und Vergleiche eingesetzt werden können, stellt Java Operatoren zur Verfügung. Arithmetische Operatoren werden für mathematische Berechnungen verwendet und liefern eine Zahl als Ergebnis:

```
int a = ((13+2) / 5);  //speichert 3 in a
```

Das Ergebnis von Vergleichsoperationen ist immer ein Wert vom Typ `boolean`. Entweder `true` oder `false`:

```
int a = ((13+2) / 5);
int b = 4;
boolean vergleich = a != b;  //liefert true
```

Logische Operatoren verarbeiten Vergleiche weiter und liefern erneut einen Wert vom Typ `boolean`:

```
boolean logik = (a == 3) && (b != 5 ); //liefert true
```

6 Bedingte Anweisungen und Schleifen

6.1 Bedingte Anweisungen

Bis jetzt haben Sie einfache Datentypen kennengelernt und Berechnungen, Vergleiche und logische Operationen durchgeführt. Als nächstes werden Sie bedingte Anweisungen kennenlernen. Bedingte Anweisungen stehen in geschweiften Klammern { }, sogenannten Blöcken.

Einen Block können Sie sich dabei wie einen gesicherten Bereich vorstellen, der nur betreten werden kann, wenn eine Bedingung erfüllt ist. Eine Bedingung wird mit dem Wort `if` eingeleitet. Sie steht in Klammern vor dem Block und ist üblicherweise ein Vergleich (z. B. `alter >=18`):

```
//Bedingte Anweisung
if(alter >= 18){
  System.out.println("Ich darf Auto fahren");
}
```

In diesem Fall können Sie die bedingte Anweisung wie folgt interpretieren: Wenn (`if`) das Alter (`alter`) größer oder gleich (>=) 18 ist, wird der Satz "Ich darf Auto fahren" angezeigt, sonst nicht.

6.2 Die if-Anweisung

Eine bedingte Anweisung wird auch if-Anweisung genannt. In der vorliegenden Klasse besteht die if-Anweisung aus nur einem Befehl. Der Block, in dem der Befehl steht, wird durch den Vergleich `i == 12` abgesichert:

```
public class IfAnweisung{
  public static void main(String[] args){
    int i = 12;
    if (i == 12){
      System.out.println("i ist gleich 12");
    }
    System.out.println("ich stehe außerhalb");
  }
}
```

Die if-Anweisung führt den Befehl nur dann aus, wenn das Ergebnis des Vergleichs wahr ist. In diesem Fall ist es wahr (`true`) und deshalb wird bei der Ausführung des Programms der

Satz "i ist gleich 12" angezeigt. Der Befehl, der nicht im Block der if-Anweisung steht, wird nicht durch den Vergleich geschützt und unabhängig davon ausgeführt.

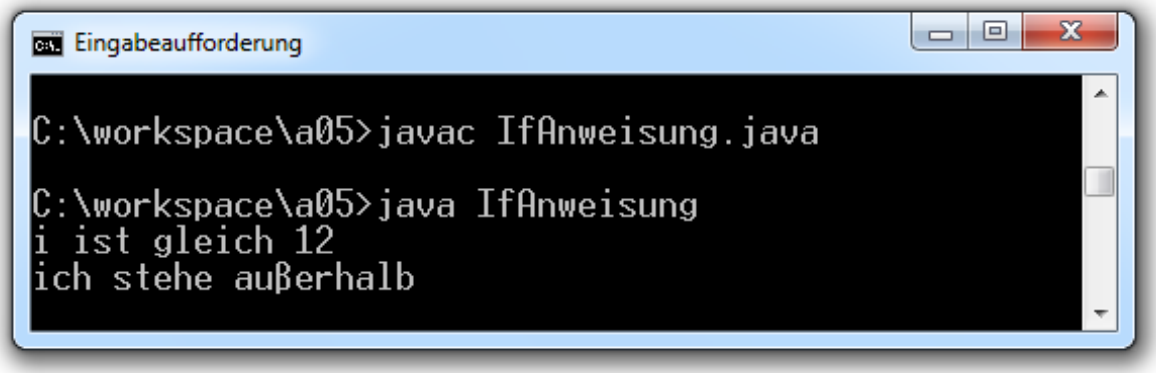

Abb. 12: Die if-Anweisung

6.3 Die if-else-Anweisung

Eine if-Anweisung kann mit einer else-Anweisung kombiniert werden. Die else-Anweisung besteht aus dem Schlüsselwort `else` und einem eigenen Block:

```
public class IfElseAnweisung{
  public static void main(String[] args){
    int i = 10;
    if (i == 12){
      System.out.println("i ist gleich 12");
    }
    else{
      System.out.println("i ist ungleich 12");
    }
    System.out.println("ich stehe außerhalb");
  }
}
```

Wenn das Ergebnis des Vergleichs der if-Anweisung falsch ist, werden die Befehle im Block der else-Anweisung ausgeführt. In der Klasse `IfElseAnweisung` liefert der Vergleich `i == 12` das Ergebnis falsch (`false`). Der Befehl im Block der else-Anweisung wird ausgeführt und die Befehle im Block der if-Anweisung ignoriert. Der Befehl außerhalb der if-else-Anweisung wird unabhängig vom Vergleich `i == 12` ausgeführt, denn er steht weder im Block der if-, noch im Block der else-Anweisung.

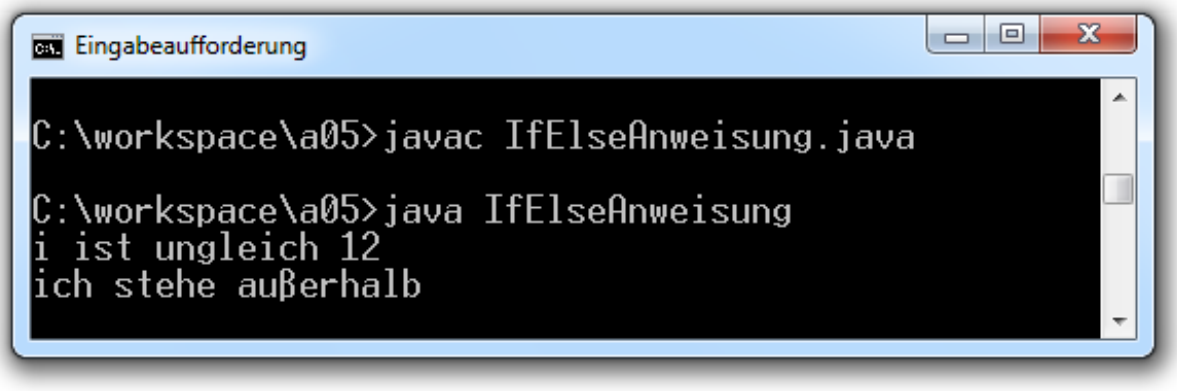

Abb. 13: Die if-else-Anweisung

6.4 Die switch-Anweisung

Mit dem Wort `switch` wird die Anweisung eingeleitet. Anschließend muss eine Variable, in der Klammer nach dem `switch`, an die Anweisung übergeben werden. Hier ist es `alter`. Die switch-Anweisung überprüft in der vorliegenden Klasse, ob der Wert der Variablen `alter` zu einem Vergleichswert (`case`) im Block der switch-Anweisung passt:

```
public class SwitchAnweisung{
  public static void main(String[] args){
    int alter = 12;
    switch ( alter ){
      case 10:
        System.out.println("Ich bin 10");
        break;
      case 11:
        System.out.println("Ich bin 11");
        break;
      case 12:
        System.out.println("Ich bin 12");  //wird ausgegeben
        break;
      default:
        System.out.println("Ich bin nicht 10, 11 oder 12");
    }
  }
}
```

Ein Vergleichswert ist eine Zahl, die mit `case` angelegt wird. Am Ende des Vergleichswerts muss ein Doppelpunkt stehen. Wenn die Variable `alter` zu einem Vergleichswert innerhalb der switch-Anweisung passt, wird der Befehl zwischen dem Vergleichswert und `break;`

ausgeführt. Passt die Variable `alter` zu keinem Vergleichswert, wird der Befehl unterhalb des Wortes `default:` ausgeführt.

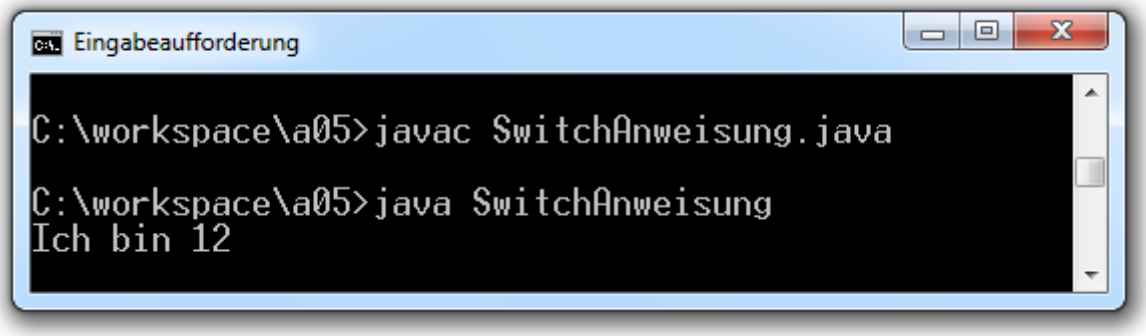

Abb. 14: Die switch-Anweisung

6.5 Schleifen

Bis jetzt werden alle Befehle, die Sie in Ihren Java-Programmen schreiben, einmal und nacheinander ausgeführt. Häufig soll ein Programm aber Befehle beliebig oft wiederholen. Hier kommen Schleifen ins Spiel:

```
//Schleife
while( i < 10 ){
  System.out.println("i: " + i);
    i = i + 1;
  }
```

Schleifen wiederholen einen Block von Befehlen beliebig oft. Wie oft genau ein Block wiederholt werden soll, müssen Sie mit einer Schleifenbedingung angeben. Eine Schleifenbedingung ist, genau wie bei bedingten Anweisungen, üblicherweise ein Vergleich (hier `i < 10`).

In diesem Fall können Sie die Schleife wie folgt interpretieren: Solange (`while`) `i` kleiner 10 ist, soll der Wert von `i` auf dem Bildschirm angezeigt werden. In jedem Durchlauf soll `i` um 1 erhöht werden (`i = i + 1`), damit die Schleife nicht ewig läuft.

6.6 Die while-Schleife

Die while-Schleife wird mit Schlüsselwort `while` eingeleitet, gefolgt von einem Vergleich. In diesem Fall `i < 10`. Der Vergleich `i < 10` legt die Laufzeit der while-Schleife fest:

```
public class WhileSchleife{
  public static void main(String[] args){
    int i = 0;
    //while-Schleife
    while( i < 10 ){
      System.out.println("i = " +i);
      i = i + 1;  //Schrittweite
    }
  }
}
```

Solange der Vergleich `i < 10` wahr ist, werden die Befehle im Schleifen-Block der while-Schleife wiederholt. Damit die Schleife nicht endlos läuft, muss man im Schleifen-Block sicherstellen, dass die Laufzeit der Schleife irgendwann zu Ende ist. Dafür muss man die sogenannte Schrittweite bestimmen. Der Befehl `i = i + 1;` sorgt dafür. Der Wert der Variablen `i` wird bei jedem Schleifendurchlauf um 1 erhöht.

Nach dem 10. Durchlauf hat `i` den Wert 10. Der Vergleich `i < 10` liefert dann das Ergebnis falsch (`false`). Das Laufzeitende ist also erreicht und die while-Schleife wird beendet.

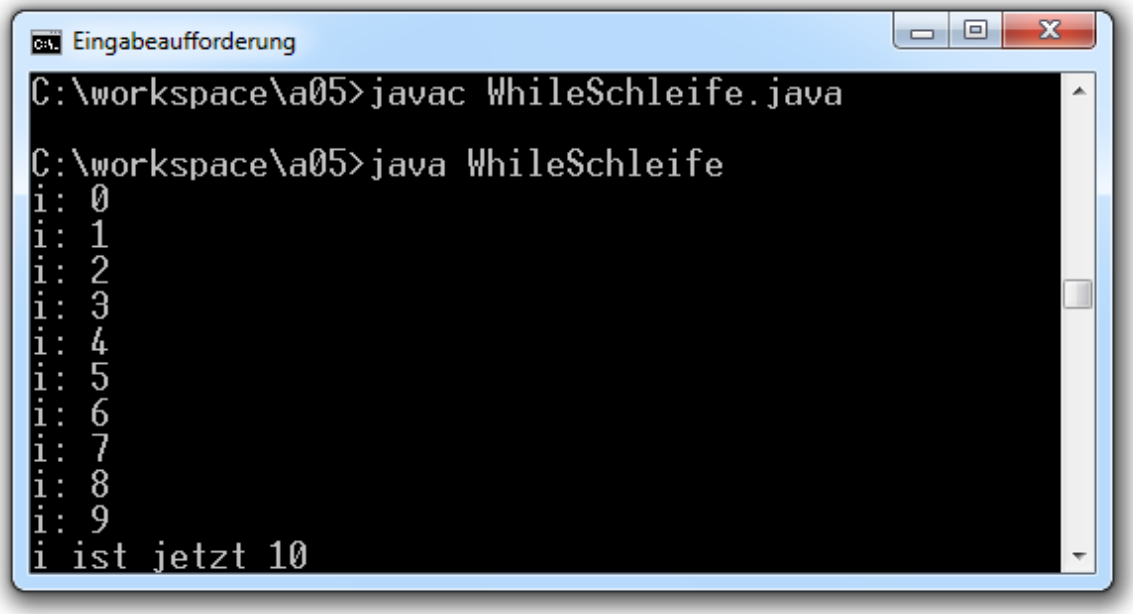

Abb. 15: Die while-Schleife

6.7 Die for-Schleife

Die for-Schleife wird mit dem Schlüsselwort `for` eingeleitet. Anschließend müssen 3 Ausdrücke in Klammern folgen:

```
public class ForSchleife{
  public static void main(String[] args){
    //for-Schleife
    for(int i = 0; i < 10; i = i + 1){
      System.out.println("i = " + i);
      System.out.println("i ist kleiner 10");
    }
    System.out.println("i ist jetzt 10");
  }
}
```

- `int i = 0;` legt in diesem Fall eine Variable mit dem Wert 0 an. Den Startwert.
- `i < 10;` bestimmt die Laufzeit der for-Schleife. Die Befehle in der for-Schleife werden solange wiederholt, wie i kleiner 10 ist.
- `i = i + 1;` legt die Schrittweite der for-Schleife fest. Jedes mal, nachdem die Befehle im Schleifen-Block ausgeführt wurden, wird der Wert von `i` um 1 erhöht.

Nach dem 10. Durchlauf hat `i` den Wert 10. Der Ausdruck i < 10 liefert dann das Ergebnis falsch und die for-Schleife wird beendet.

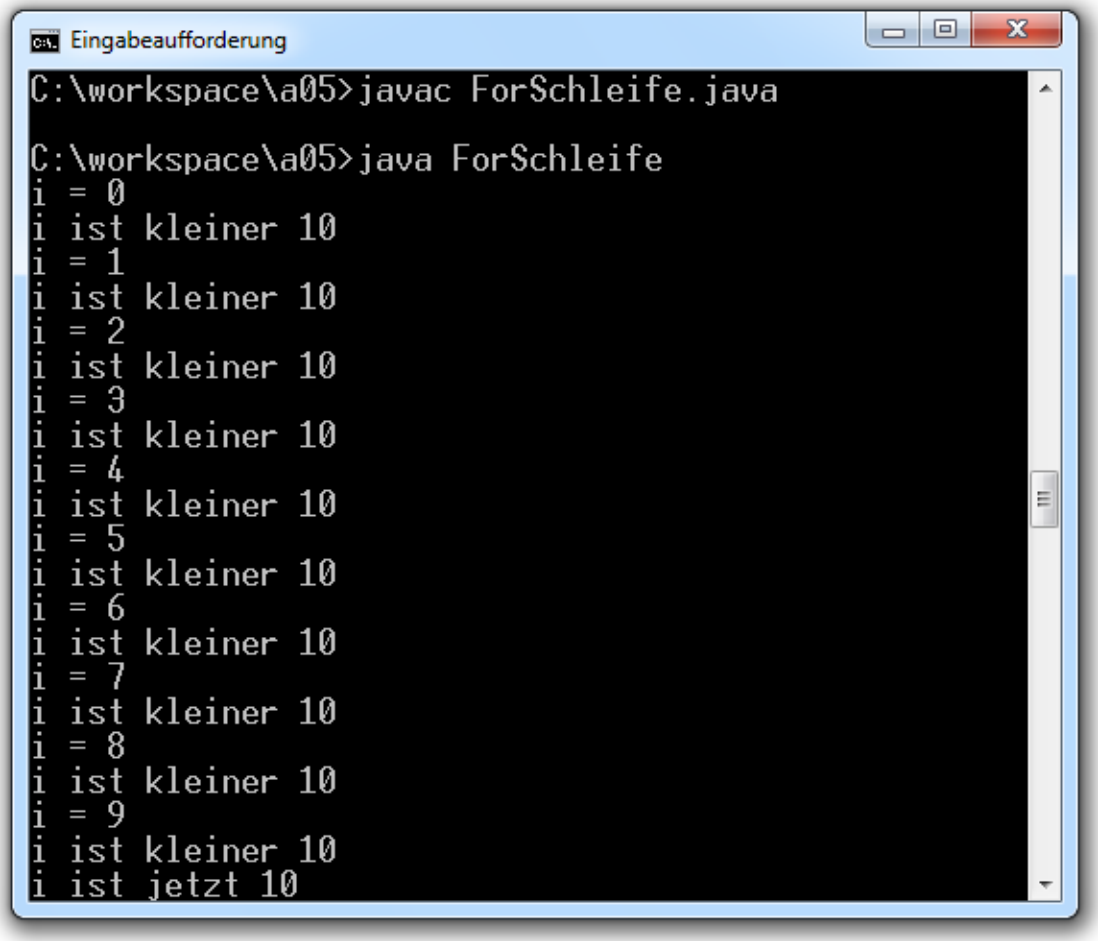

Abb. 16: Die for-Schleife

6.8 Lessons learned

Eine bedingte Anweisung heißt if-Anweisung. Die if-Anweisung besteht aus einem Block, in dem ein oder mehrere Befehle stehen:

```
if(alter >=18){
  System.out.println("Ich darf Auto fahren");
}
```

Die Befehle des Blocks werden nur ausgeführt, wenn eine Bedingung erfüllt ist. Die Bedingung wird direkt vor dem Block mit dem Wort `if` eingeleitet. Anschließend folgt eine Klammer in der Sie die eigentliche Bedingung formulieren. (z. B. `alter >= 18`).

Schleifen sind Blöcke von Befehlen, die solange wiederholt werden, wie eine Schleifenbedingung (`i < 10`) wahr (`true`) ist. Eine Schleife braucht, neben einer Bedingung, einen Startwert (`int i = 0`), und eine Schrittweite (`i = i + 1`):

```
//Quellcode
for(int i = 0; i < 10; i = i + 1){
  System.out.println("i ist " +i);
}
```

7 Arrays

7.1 Die Rolle eines Arrays

Wenn Sie sich eine Variable wie einen Speicherplatz vorstellen, in dem ein Wert gespeichert werden kann, dann ist ein Array so was wie ein Lager, bestehend aus mehreren Fächern, in denen mehrere Werte des gleichen Typs gespeichert werden können. Das Array hat eine feste Anzahl an Fächern und jedes Fach hat eine Nummer. In ein Fach können Sachen gelegt werden. Da ein Array ein Konstrukt einer Programmiersprache ist, sind Sachen üblicherweise Zahlen, Zeichen oder andere Daten.

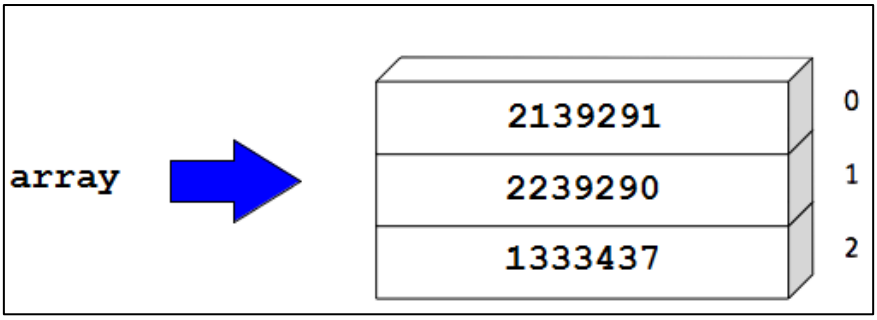

Abb. 17: Ein Array als Lager

7.2 Eindimensionale Arrays

Bleiben wir bei dem Beispiel mit dem Lager. Wenn Sie in einem Lager Artikel speichern wollen, können Sie das mit einem Array tun. Da Sie in einem Programm keine echten Artikel speichern können, speichert man Artikelnummern:

```
public class ArtikelLager{
  public static void main(String[] args){
    int[] lager = new int[3];
    //Quellcode
```

In diesem Fall wird das Array `lager` mit dem Datentyp `int` angelegt. Ein Array, das Daten vom Typ `int` speichert, wird mit `int[]` bekanntgegeben. Die eckigen Klammern legen fest, dass es ein Array ist. Danach folgt der Name. Mit `new int[3]` wird das Array mit Fächern ausgestattet. In den Klammern steht die Anzahl der Fächer. `lager` hat in diesem Fall 3 Fächer. Bis jetzt sind alle Fächer leer. Um Artikelnummern in `lager` zu speichern, müssen sie die Nummer des Fachs benutzen.

7.3 Eindimensionale Arrays verwenden

Auf die einzelnen Fächer eines Arrays greift man über ihre Nummer zu. In Java hat das erste Fach eines Arrays die Nummer 0, **nicht** 1. Das Array `lager` wird in diesem Fall mit einigen

fiktiven Artikelnummern gefüllt. Dafür wird der Name des Arrays angegeben und in einer eckigen Klammer die Nummer des Fachs, in dem man die Artikelnummer speichern will:

```
    int[] lager = new int[3];
    lager[0] = 2139291;
    lager[1] = 2239290;
    lager[2] = 1333437;
```

Die Zuweisung der Artikelnummern erfolgt durch ein Gleichzeichen. Nachdem Artikelnummern im Array gespeichert sind, kann man auf diese über ihre Nummer zugreifen. Ein Fach eines Arrays kann, wie eine Variable, mit Operatoren kombiniert oder in einen Befehl geschrieben werden:

```
    System.out.println("Artikel 1: " +lager[0]);
    System.out.println("Artikel 2: " +lager[1]);
    System.out.println("Artikel 3: " +lager[2]);
  }
}
```

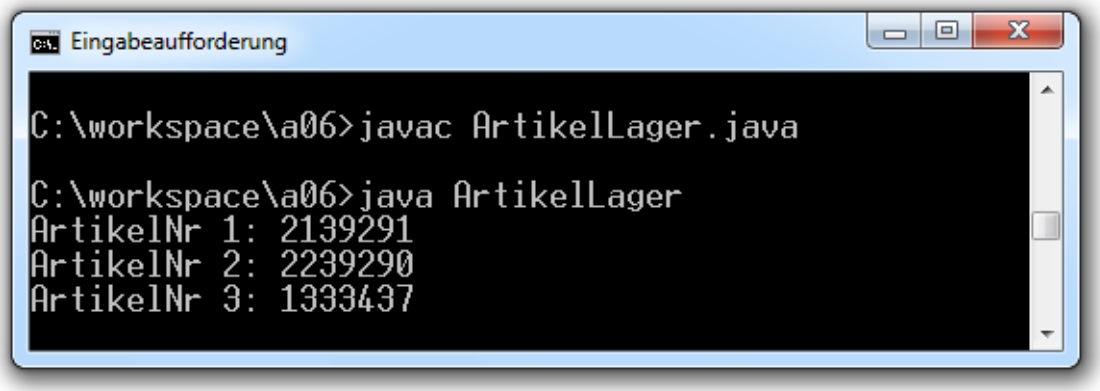

Abb. 18: Ein Array mit Artikelnummern ausgeben

7.4 Arrays direkt initialisieren

Wenn ein Array nicht zu viele Fächer hat, kann man diese auch beim Erzeugen des Arrays direkt angeben. Dafür verwendet man eine geschweifte Klammer { }. Der Vorteil dabei ist, dass man nicht das Wort `new` benutzen und auch nicht die Anzahl der Fächer angeben muss:

```
public class Arrays{
  public static void main(String[] args){
    int[] ganzezahlen = {1, 2, 3, 4};
    double[] kommazahlen = {12.2, 8.2, 13.37, 6.0};
    System.out.println(ganzezahlen[3] * kommazahlen[0]);
```

```
    System.out.println(ganzezahlen[1] / kommazahlen[1]);
  }
}
```

In der Klasse `Arrays` werden zwei Arrays (`ganzezahlen, kommazahlen`) für unterschiedliche Datentypen definiert. Jeder Array besteht aus einem anderen Datentyp und wird direkt in geschweiften Klammern mit Werten initialisiert.

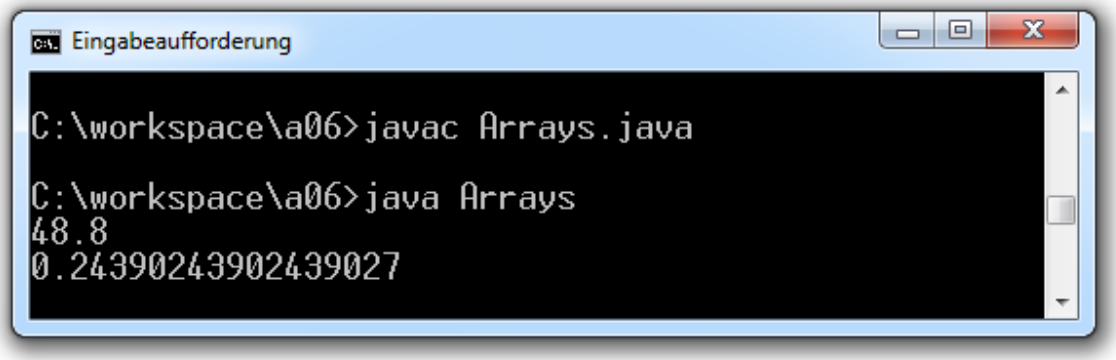

Abb. 19: Ein Array direkt initialisieren und ausgeben

7.5 Arrays und Schleifen

Um auf die Fächer eines Arrays zuzugreifen, spricht man sie mit ihrer Nummer an. Sie müssen aber nicht jedes Fach einzeln im Quellcode angeben, sondern können diese auch bequem mit einer Schleife auslesen:

```
public class ArraySchleife{
  public static void main(String[] args){
    int[] feld = {4,3,1,2,3,4,5};
    int i = 0;
    while(i < 7){
      System.out.println(feld[i]);
      i = i + 1;
    }
    System.out.println("Array wurde ausgelesen");
  }
}
```

In der vorliegenden Klasse liest eine while-Schleife die Fächer des Arrays `feld` aus. Die Variable `i` ist der Startwert der while-Schleife. Die Schleife selbst wird solange ausgeführt, wie `i < 7` ist - also 6 mal, denn die Schrittweite der Schleife ist `i = i + 1;`. Jedes Mal wenn die Schleife durchlaufen wird, wird ein Array-Fach mit `feld[i]` angezeigt. Im ersten

Durchlauf `feld[0]`. Am Ende des ersten Durchlaufes wird `i` mit `i = i + 1;` auf 1 erhöht. Der zweite Durchlauf zeigt `a[1]` an, denn `i` ist dann 1. Danach wird `i` wieder erhöht und der nächste Durchlauf beginnt. Die Schleife wiederholt die Befehle solange, bis `i` auf 7 erhöht wird. Bei `i = 7` bricht sie ab.

Abb. 20: Ein Array mit einer Schleife auslesen

7.6 Zweidimensionale Arrays

Auch mehrere Dimensionen sind mit einem Array möglich. Wenn man eine Tabelle mit 3 Spalten und 3 Reihen benötigt, kann man diese, wie hier mit dem Array `tabelle`, anlegen:

```
 public class Tabelle{
  public static void main(String[] args){
    int[] tabelle = new int[3][3]
    tabelle[0][0] = 1;
    tabelle[0][1] = 2;
    tabelle[0][2] = 3;
    tabelle[1][0] = 4;
    tabelle[1][1] = 5;
    tabelle[1][2] = 6;
    tabelle[2][0] = 7;
    tabelle[2][1] = 8;
    tabelle[2][2] = 9;
  }
}
```

Das Array `tabelle` hat zwei Dimensionen. Die erste Dimension stellt 3 Reihen dar. Die zweite Dimension 3 Spalten. Mit `tabelle[i][j]` können Sie auf Elemente der Tabelle zugreifen. `i` steht dabei für eine Spalte und `j` für eine Reihe.

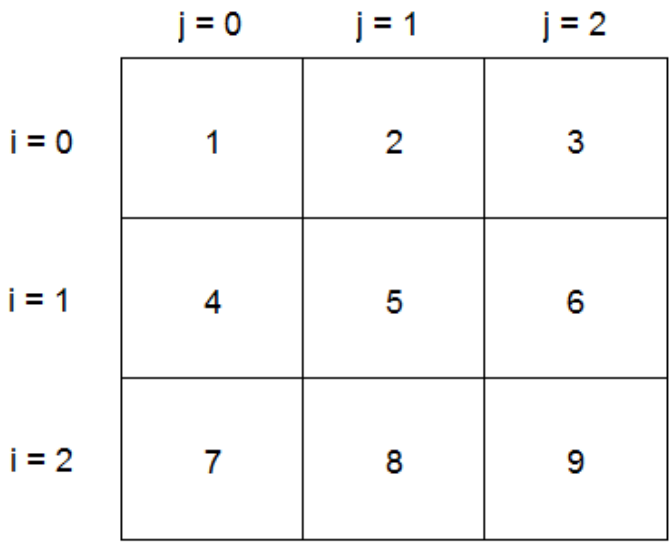

Abb. 21: Zweidimensionales Array

7.7 Lessons learned

Ein Array ist eine Variable, in der mehrere Zahlen, Zeichen oder Zeichenketten abgespeichert werden. Ein Array hat Fächer und muss mit dem Schlüsselwort `new` oder geschweifte Klammern initialisiert werden:

```
//Quellcode
int[] lager1 = new int[3];
lager1[0] = 2139291;
lager1[1] = 2239290;
int[] lager2 = {21931233,2331233,...};
lager1[0] * lager2[1]
```

Jedes Fach eines Arrays hat eine Nummer. Das erste Fach hat die Nummer 0. Das zweite Fach die Nummer 1. Fächer eines Arrays werden über ihre Nummer aufgerufen und können, wie normale Variablen, verwendet werden.

8 Methoden

8.1 Die Rolle von Methoden

Der Startpunkt des Programmablaufs ist immer die main-Methode. In der main-Methode muss man nacheinander schreiben, was das Programm machen soll. Damit nicht alle Befehle in der main-Methode stehen müssen, kann man weitere Methoden außerhalb von `main` schreiben. Eine Methode außerhalb von `main` ruft man mit einem Methodenaufruf auf:

```
public static int quadriere(int x){
   return x * x;
}
public static void main(String[] args){
   //Quellcode
   int zahl = quadriere(5); //Methodenaufruf
   System.out.println(zahl) //gibt 25 aus
}
```

Ein Methodenaufruf übergibt Parameter. Die Methode verarbeitet die Parameter und gibt sie an den Aufruf zurück. Zur Einführung in die Funktionsweise und den Aufbau von Methoden wird auf den folgenden Seiten ein Programm geschrieben, das mit Methoden mathematische Berechnungen durchführt.

8.2 Eine Methode deklarieren und verwenden

Die Klasse `MatheProgramm` ist hier das Beispiel. Am Anfang soll eine Methode zur Berechnung eines Quadrats bereitgestellt werden:

```
public class MatheProgramm{
  public static int quadriere(int x){  //Methode quadriere
     return x * x;
  }
```

Die Methode wird über der main-Methode deklariert. Ihr Name ist `quadriere`. Direkt nach dem Namen folgt eine Klammer. In der Klammer steht ein Parameter (hier `int x`). Damit Sie `quadriere` nutzen können, müssen Sie die Methode in der main-Methode mit *MethodenName(Parameter);* aufrufen. Ein Methodenaufruf besteht aus dem Namen der Methode und einem Parameter:

```
  public static void main(String[] args){
     //Quellcode
     int ergebnis = quadriere(5);       //Methodenaufruf
     System.out.println(ergebnis);
  }
}
```

Ein Parameter ist eine Variable die als Eingabewert verwendet wird. Wenn man die Methode mit `quadriere(5)` aufruft, wird die Zahl 5 als Parameter `int x` übergeben. Im Block der Methode wird der Parameter 5 mit `x * x` quadriert. `return` sorgt dafür, dass das Ergebnis der Multiplikation (25) zurückgegeben wird.

Der Rückgabewert (25) wird an den Methodenaufruf zurückgegeben. Um Ihn zu verwenden, muss der Methodenaufruf in einer Variablen gespeichert werden. Hier wird `quadriere(5)` in der Variablen `int ergebnis` gespeichert.

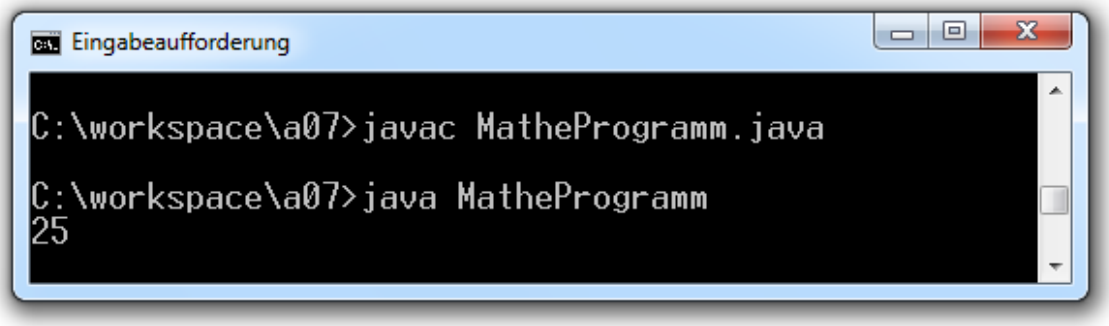

Abb. 22: Das Ergebnis der Methode „quadriere“

8.3 Rückgabewert und Rückgabetyp einer Methode

Direkt vor den Methodennamen muss ein Rückgabetyp geschrieben werden. Ein Rückgabetyp legt fest, von welchem Datentyp der Rückgabewert ist:

```
public static int quadriere(int x){
     return x * x;
  }
```

Die Methode `quadriere` hat den Rückgabetyp `int`. Das bedeutet, die Methode gibt eine Zahl zurück, die mit dem Datentyp `int` abgebildet wird. Für die aufrufende Methode (hier `main`) bedeutet das, dass sie den Methodenaufruf in einer int-Variablen speichern muss, um den Rückgabewert weiterzuverwenden:

```
public static void main(String[] args){
     //Erlaubt
     int ergebnis1 = quadriere(5);
     int ergebnis2 = quadriere(4);
     //Verboten
     char ergebnis3 = quadriere(2);
     byte ergebnis4 = quadriere(3);

     System.out.println("5^2 = " +ergebnis1);
     System.out.println("4^2 = " +ergebnis2);
  }
}
```

8.4 Die Eigenschaften static und public

Vor dem Rückgabetyp stehen die Worte `static` und `public`:

```
  //Falsch
  public int quadriere(int y){
     return x * x;
  }

  //Richtig
  public static int quadriere(int x){
     return x * x;
  }
```

Das Wort `static` bedeutet, dass die Methode mit einem Methodenaufruf ausgeführt werden kann. Wenn Sie `static` vergessen, können Sie eine Methode nicht mehr wie gewohnt ausführen. `public` legt fest, dass eine Methode öffentlich ist. In Java gibt es öffentliche oder private Methoden. Da Sie private Methoden noch nicht kennen, müssen Sie jede Methode mit `public` einleiten.

8.5 Methoden mit mehreren Parametern

Als nächstes wird die Klasse `MatheProgramm` mit einer neuen Methode erweitert. Die Methode `multipliziere` führt eine Multiplikation durch:

```
public class MatheProgramm{
  public static int multipliziere(int x, int y){
    return x * y;
  }
```

Neu ist, dass die Methode zwei Parameter hat. Wenn man einer Methode mehrere Parameter gibt, muss man diese mit einem Komma trennen.

Die Parameter `int x` und `int y` werden im Block der Methode miteinander multipliziert. `return` markiert die Multiplikation als Rückgabewert. Der Rückgabetyp ist wieder `int`. Aus diesem Grund muss ein Aufruf der Methode auch in einer int-Variablen gespeichert werden:

```
    int ergebnis1 = multipliziere(5,5);  //x=5, y=5
    int ergebnis2 = multipliziere(2,5);  //x=2, y=5
    System.out.println(" 2 * 5 = " +ergebnis1);
    System.out.println(" 5 * 5 = " +ergebnis2);
```

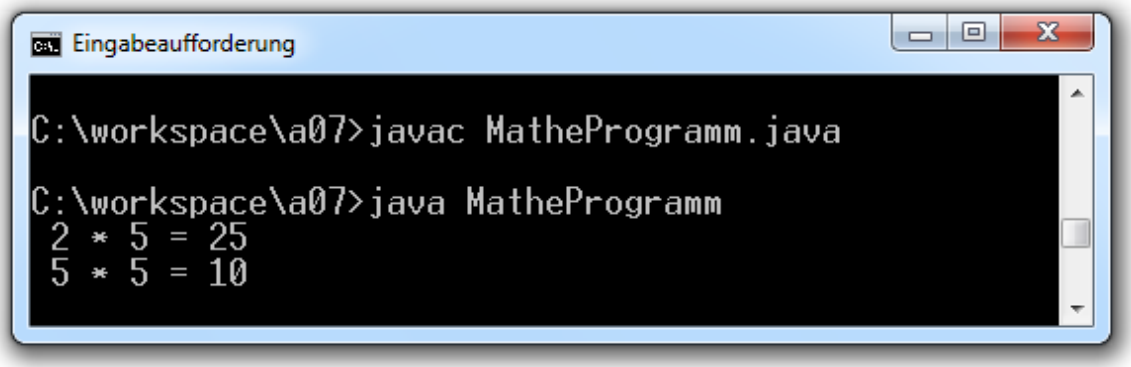

Abb. 23: Das Ergebnis der Methode „multipliziere“

8.6 Methoden mit void als Rückgabetyp

In einigen Situationen kann es vorkommen, dass man eine Methode schreiben muss, die keinen Rückgabewert hat. Die Methode in unserem Programm heißt `anzeigen`:

```
public class MatheProgramm{
  public static void anzeigen(int zahl){
    System.out.println("Das Ergebnis ist " +zahl);
  }
```

Die Aufgabe von `anzeigen` ist es, einen Parameter, der ihr übergeben wird, auf dem Bildschirm anzuzeigen. Sie soll also einfach nur einen Befehl ausführen und keinen Rückgabewert haben. Bei Methoden ohne Rückgabewert muss man das Wort `void` als Rückgabetyp verwenden.

Methoden die `void` sind, geben nichts an ihren Aufruf zurück. Sie führen alle Befehle in ihrem Block aus und sind dann zu Ende:

```
  public static int multipliziere(int x, int y){
    return x * y;
  }
  public static int quadriere(int x){
     return x * x;
  }
  public static void main(String[] args){
     int ergebnis1 = quadriere(4);
     int ergebnis2 = multipliziere(10,5);
     anzeigen(ergebnis1); //gibt 16 aus
     anzeigen(ergebnis2); //gibt 25 aus
  }
}
```

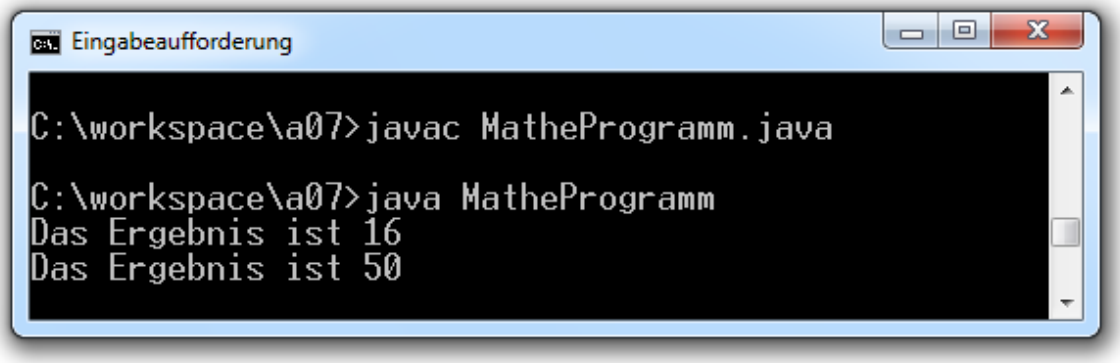

Abb. 24: Die Klasse „MatheProgramm“ kompilieren und ausführen

8.7 Methoden verschachteln

Eine Methode kann beliebig oft mit passenden Parametern aufgerufen werden. Sie können eine Methode aber nicht nur aufrufen, sondern auch mit anderen Methoden verschachteln:

```
public class MatheProgramm{
  //Methoden
  public static void main(String[] args){
     //Quellcode
     anzeigen(multipliziere(quadriere(2), quadriere(5)));
  }
}
```

In der main-Methode von `MatheProgramm` zeigt `anzeigen` den Rückgabewert von `multipliziere` an. `multipliziere` führt davor eine Multiplikation mit den Rückgabewerten von `quadriere(2)` und `quadriere(5)` durch.

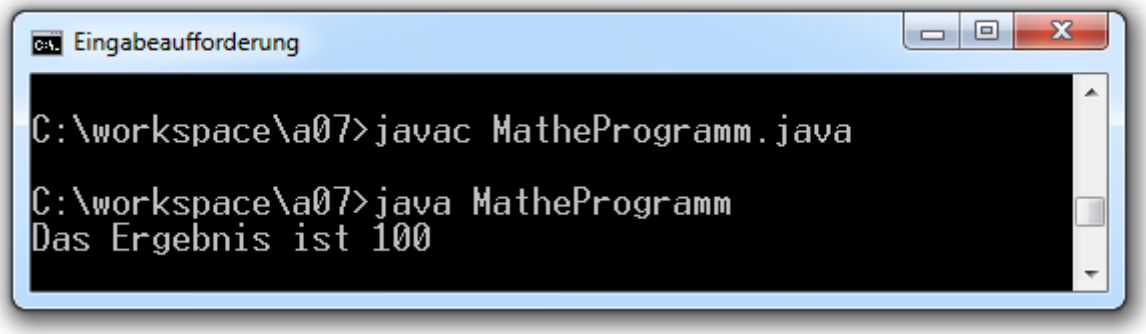

Abb. 25: Verschachtelte Methode ausführen

Wenn Sie Methoden verschachteln, wird immer von innen nach außen ausgewertet. Das bedeutet in diesem Fall, dass zuerst quadriert, dann multipliziert und dann angezeigt wird.

8.8 Der Programmaufbau

Wenn Sie ein Programm schreiben, sollten Sie die main-Methode von den restlichen Methoden klar trennen. Das bedeutet, Sie legen alle Methoden entweder unterhalb oder oberhalb der main-Methode an. In der main-Methode führen Sie dann die Programmlogik ein. Sie formulieren mit logisch angeordneten Befehlen die Programmschritte, um zur Lösung eines Problems oder einer Aufgabe zu gelangen. Damit der Quellcode gut lesbar bleibt, sollten alle Befehle, die sich wiederholen, in Methoden außerhalb von `main` formuliert werden.

8.9 Lessons learned

Eine Methode hat einen Rückgabetyp, einen Namen und Parameter. Zudem muss eine Methode mit den Worten `public` und `static` eingeleitet werden. Im Methoden-Block steht, was die Methode machen soll:

```
  public static int quadriere(int x){
    return x * x;
  }
```

Wenn die Methode einen Rückgabetyp hat, muss mit `return` ein Rückgabewert markiert werden. Um eine Methode zu verwenden, schreibt man einen Methodenaufruf in die main-Methode. Ein Methodenaufruf besteht aus dem Namen der Methode und einem Parameter:

```
int a = quadriere(4);
```

Wenn eine Methode kein Ergebnis hat, ist der Rückgabetyp `void`:

```
public static void anzeigen(int zahl){
  System.out.println(“Das Ergebnis ist “ +zahl)
}
```

9 Modulare Programmierung

9.1 Modularisierung von Programmen

Bisher wurden Programme geschrieben, die aus einer Java-Datei bestehen. In der Java-Datei wurde eine Klasse mit einer main-Methode definiert. Die main-Methode steuert die Logik des Programms. Alles beginnt mit dem ersten Befehl in der main-Methode und endet mit dem Letzten. Damit nicht alle Befehle in `main` stehen müssen, kann man Methoden anlegen, die aus `main` aufgerufen werden:

```
public class RechenProgramm{
  public static int multipliziere(int x, int y){
     return x * x;
  }
  public static void main(String[] args){
    multipliziere(2,3)
  }
}
```

Diese Methoden müssen aber nicht alle in einer Klasse definiert sein, sondern können auch auf verschiedene Klassen verteilt werden. Da pro Java-Datei nur eine Klasse existieren darf, muss man für jede Klasse eine neue Datei anlegen. Wenn man Klassen auf mehrere Dateien verteilt, spricht man auch von modularer Programmierung:

Bei der modularen Programmierung erzeugt man Klassen als vorgefertigte Funktionseinheiten. Diese Klassen beinhalten Methoden für bestimmte Aufgabenbereiche und können aus der main-Methode eines *Hauptprogramms* aufgerufen werden.

9.2 Eine Klasse als Modul schreiben

Wenn man modular programmiert, sollten Methoden nicht nach Belieben auf verschiedene Klassen verteilt werden, sondern nur dann, wenn sich daraus eine logische Einheit ergibt. Klassen mit ihren Methoden bilden dann eine logische Einheit, wenn sie einzeln geplant, programmiert und getestet werden können. Sie müssen ein abgeschlossener Funktionsteil sein, der unabhängig von der restlichen Programmlogik entwickelt werden kann. Eine so entwickelte Klasse wird Modul genannt:

```
//Datei Rechenarten.java (Modul)
public class Rechenarten{
  public static int addiere(int x, int y){
    return x + y
  }
  public static int subtrahiere(int x, int y){
    return x - y
  }
  public static int multipliziere(int x, int y){
    return x * y;
  }
  public static double dividiere(int x, int y){
    return x / y;
  }
}
```

Als Beispiel werden hier die vier Grundrechenarten verwendet. Egal in welchem Programm man Grundrechenarten einsetzt, sie funktionieren immer gleich. Grundrechenarten sind also unabhängig von der speziellen Logik eines Programms und sollten in einer eigenen Klasse (hier ist es `Rechenarten`) implementiert sein.

9.3 Eine Klasse als Modul verwenden

Alleinstehend kann `Rechenarten` nicht ausgeführt werden, denn Module sind Unterprogramme, die keine main-Methode haben. Sie müssen von einem Hauptprogramm (Klasse mit main-Methode) ausgeführt werden. Hier wird die Klasse `RechenProgramm` als Hauptprogramm geschrieben:

```
//Datei Rechenprogramm.java (Hauptprogramm)
public class RechenProgramm{
  public static void main(String[] args){
    int ergebnis = Rechenarten.mulitpliziere(12,2);
    System.out.println("Das Ergebnis ist " +ergebnis); //24
  }
}
```

In der main-Methode von `RechenProgramm` wird gesteuert, was das Programm tun soll. Wenn man die Grundrechenarten aus dem Modul bzw. der Klasse `Rechenarten` nutzen möchte, kann man die Punktnotation verwenden. Mit der Punktnotation

Klassenname.Methodenname greift man auf Methoden externer Klassen bzw. Module zu. Die Punktnotation funktioniert hier, wenn beide Klassen im selben Ordner gespeichert sind.

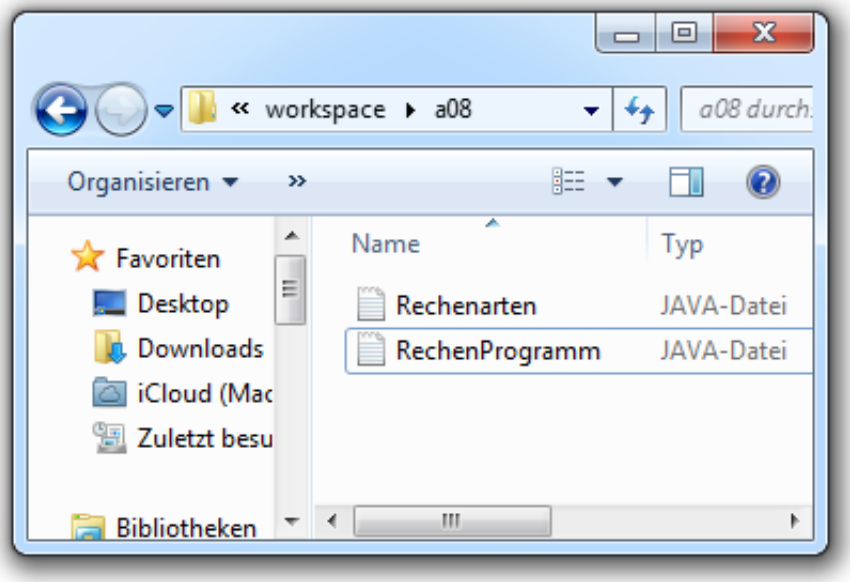

Abb. 26: „Rechenarten.java“ und „RechenProgramm.java“

9.4 Das Baukastenprinzip

Genau wie Lego-Bausteine für mehrere Baukästen verwendet werden, können auch Klassen, die als Module geschrieben werden, in mehreren Programmen wiederverwendet werden. Damit mehrere Programme die Klasse `Rechenarten` nutzen können, sollte man sie an zentraler Stelle vorhalten. Hier wird die Klasse in *C:/workspace/module* verschoben:

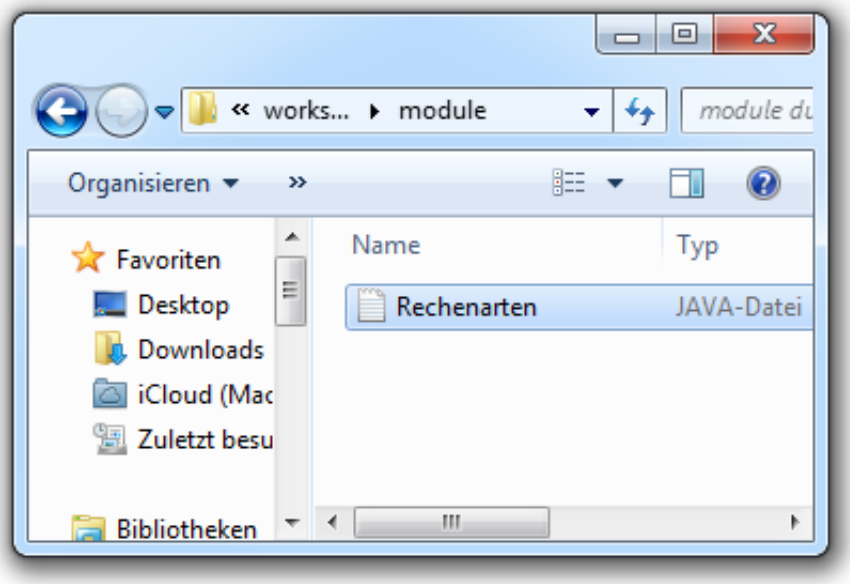

Abb. 27: Ein Ordner als Baukasten für Module

Der Quellcode muss mit `package module;` erweitert werden, um Java mitzuteilen, dass die Klasse `Rechenarten` Teil des Ordners bzw. Baukastens *module* sein soll:

```
//Datei Rechenarten.java (Modul)
package module;

public class Rechenarten{
  public static int multipliziere(int x, int y){
    return x * y;
  }
  //Quellcode
}
```

Wenn man jetzt ein Programm schreibt, kann man sich an den `Rechenarten` im Baukasten *module* mit `import module.Rechenarten;` bedienen. Der Import-Befehl muss immer vor der Klassendefinition stehen. Hier wird die Klasse `RechenProgramm` als Beispiel angepasst:

```
//Datei RechenProgramm.java
import module.Rechenarten;


public class RechenProgramm{
```

Weil `Rechenarten` nicht mehr im selben Ordner wie `RechenProgramm` liegt, muss man beim Kompilieren von `RechenProgramm` mitteilen, wo der Baukasten `module` liegt, auf den mit `import module.Rechenarten;` zugegriffen wird. Dafür nutzt man ***-classpath*** beim Kompilieren und Ausführen.

`RechenProgramm` wird hier in C:/workspace/*a08* gespeichert und muss deshalb aus *a08* mit:

javac -classpath "C:/workspace" RechenProgramm.java

kompiliert und mit

java -classpath "C:/workspace";. RechenProgramm

ausgeführt werden.

Abb. 28: „RechenProgramm“ mit –classpath kompilieren und ausführen

9.5 Die Programmierschnittstelle

Bis jetzt haben Sie gelernt, wie man Klassen als Modul schreibt, an zentraler Stelle speichert und verwendet. Für jede Aufgabe, die ein Programm lösen soll, eine eigene Klasse mit Methoden zu schreiben, ist aber nicht nötig.

Java bietet eine Programmierschnittstelle (engl. application programming interface), die eine Menge an Klassen beinhaltet, die jeder Programmierer frei nutzen darf. Diese Klassen werden automatisch mit Java installiert und können, wie selbstgeschriebene Klassen, mit dem Wort `import` importiert werden. Eine Übersicht und die Dokumentation dieser Klassen findet man unter dem folgenden Link:

http://docs.oracle.com/javase/7/docs/api/

Zur Einführung in die Verwendung der Java-Programmierschnittstelle (Java-API) wird die Klasse `Math` in eine Klasse importiert und verwendet. Was `Math` so kann, steht hier:

http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

9.6 Die Klasse Math aus der API importieren

Für das vorliegende Programm wurde nur die Klasse `Zufall` selbst geschrieben. Die Klasse `Math` kommt aus der API:

```
//Datei Zufall.java
import java.lang.Math.*;
public class Zufall{
  public static void main(String[] args){
    for(int i = 1; i <= 5; i = i + 1){
      double zufall = Math.random();
      zufall = zufall * 100;
      System.out.println(zufall);
    }
  }
}
```

Um die Klasse `Math` aus der Programmierschnittstelle zu importieren, müssen Sie ganz am Anfang des Quellcodes den Befehl `import java.lang.Math.*;` schreiben. Anschließend kann man alle Methoden von `Math` nutzen. In der vorliegenden Klasse wird die Methode `random()` verwendet. In der Online-Dokumentation der Programmierschnittstelle können Sie die Funktionsweise von `random()` nachlesen.

Hier wird `random()` in einer for-Schleife 5 Mal ausgeführt, um Zufallswerte auf dem Bildschirm anzuzeigen. Da `random()` `double`-Werte zwischen 0.0 und 1.0 ausgibt, wird jeder Wert mit 100 multipliziert, um eine größere Zahl zu erhalten.

Wichtig: Bei Klassen die als Module aus der Programmierschnittstelle importiert werden, muss *-classpath* **nicht** verwendet werden, um mitzuteilen, wo die Klasse liegt. Java findet Klassen aus der Java-API automatisch, wenn der import-Befehl vorhanden ist.

9.7 Lessons learned

Bei der modularen Programmierung werden Programme und ihre Methoden auf mehrere Klassen verteilt, wenn sich logische Einheiten ergeben. Eine logische Einheit ist eine Klasse die einzeln geplant, programmiert und getestet werden kann. Man spricht dabei auch von Modulen. Module können nicht eigenständig ausgeführt werden, sondern müssen in eine Klasse mit main-Methode (Hauptprogramm) importiert werden. Dafür schreibt man das Wort `import` an den Anfang des Quellcodes einer Klasse. Mit `import` kann man aber nicht nur selbstgeschriebene Klassen als Modul importieren, sondern auch Klassen aus der Java-Programmierschnittstelle (API), wie z. B. mit `import java.lang.Math.*;`

10 Objektorientierung

10.1 Objekte und Objektorientierung

Jeder Gegenstand den Sie sich vorstellen können ist ein Objekt. Objekte können Dinge (z. B. Auto, Tisch), Lebewesen (z. B. Mensch, Tier) oder Begriffe (z. B. Adressen, Titel) sein. Ein Objekt wird durch seine Eigenschaften beschrieben. Ein Auto wird z. B. durch die Eigenschaften Marke, Modell und Motorleistung beschrieben. In der Objektorientierung bedient man sich genau dieser Sichtweise. Alle Aufgaben, die ein Programm erledigen soll, werden in Objekte umgewandelt. Dieses Konzept ist abstrakter als die Modularisierung von Programmen, ermöglicht auf Dauer aber intuitivere und übersichtlichere Programme.

10.2 Klassen als Datentyp

Als Beispiel wird ein Programm geschrieben, das Konten für eine Bank erfasst und verwaltet. Ein Konto hat mehrere Eigenschaften: einen Inhaber, ein Guthaben, die IBAN und den BIC. Bisher haben Sie eine Reihe von Datentypen wie `int`, `double` oder `String` kennengelernt. Es wäre möglich für jedes Konto Variablen mit diesen Datentypen anzulegen und die Eigenschaften darin zu speichern. Das funktioniert zwar, kann aber zu Problemen führen:

```
//Datei BankProgramm.java
public class BankProgramm{
  public static void main(String[] args){
    String inhaber_01 = "Max Muster";
    double betrag_01 = 50000;
    String iban_01 = "DE230511203027371";
    String bic_01 = "BA28AN";
    //Quellcode
    String konto_inhaber_100 = "Moritz";
    int konto_betrag_100 = 1000;
    String konto_iban_100 = "DE23320129232";
    String bic_100 = "BA23AN";
    boolean aktiv_100 = false;
  }
}
```

Gerade wenn Programme komplexer werden, kann es vorkommen, dass unterschiedliche Datentypen oder missverständliche Namen gewählt werden, um ein Konto abzubilden. Es wäre

besser, wenn man einen Datentyp hätte, der alle Eigenschaften eines Kontos beinhaltet und diese dem Programmierer vorgibt.

Um einen solchen Datentyp zu realisieren, deklariert man eine Klasse, die alle Eigenschaften eines Kontos enthält. Dazu muss man eine neue Datei anlegen, mit dem Quellcode dieser Klasse:

```
//Datei Konto.java
public class Konto{
  String inhaber;
  double guthaben;
  String iban;
  String blz;
}
```

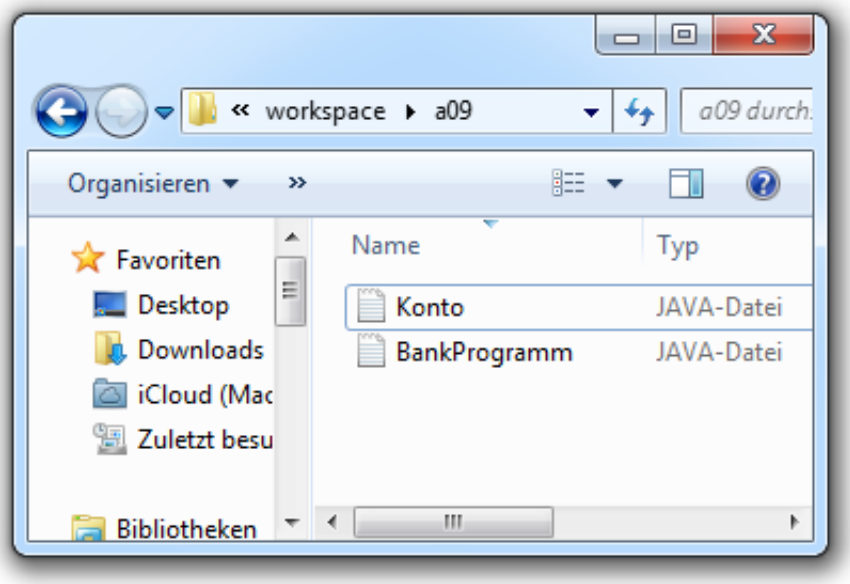

Abb. 29: Die Klassen „Konto“ und „BankProgramm“

10.3 Objekte erzeugen

Die Eigenschaften eines Kontos wurden mit den Variablen `String inhaber`, `double guthaben`, `String iban` und `String bic` in der Klasse `Konto` implementiert. Um diese Eigenschaften als Datentyp zu verwenden, muss man in der main-Methode des Bank-Programms `Konto giro = new Konto();` schreiben:

```
//Datei BankProgramm.java
public class BankProgramm{
  public static void main(String[] args){
   Konto giro = new Konto();
```

Mit Konto `giro` gibt man Java bekannt, dass es eine Variable vom Datentyp `Konto` geben soll, die `giro` heißt. Mit `new Konto();` wird dann ein Objekt erzeugt. Das Objekt besteht aus einer Kopie aller Eigenschaften, die in der Klasse `Konto` definiert wurden. Das Gleichzeichen (=) speichert das Objekt in der Variablen `giro`.

Mit der Punktnotation können Werte in den Eigenschaften des Objekts abgelegt werden. Dazu schreibt man *Variablenname.Eigenschaft:*

```
   giro.inhaber = "Max Muster";
   giro.guthaben = 50000;
   giro.iban = "DE131232123"
   giro.bic = "A4GS2"
  }
}
```

10.4 Objekte als Instanzen einer Klasse

Aus der Klasse `Konto` kann man aber nicht nur ein Objekt, sondern beliebig viele Objekte erzeugen. In Java nennt man dieses Vorgehen "Instanzen einer Klasse bilden".

Ein Objekt ist die Instanz einer Klasse. Das bedeutet, jedes Objekt der Klasse `Konto` besteht aus einer exakten Kopie aller Eigenschaften, die in der Klasse angelegt wurden. Die Eigenschaften können für jedes Objekt individuell mit Werten belegt werden:

```
//Datei BankProgramm.java
public class BankProgramm{
  public static void main(String[] args){
    Konto giro = new Konto();
    giro.inhaber = "Max Muster";
    giro.guthaben = 50000;
    giro.iban = DE131232123;
    giro.bic = A4GS2;

    Konto tagesGeld = new Konto();
    tagesGeld.inhaber = "Moritz";
    tagesGeld.guthaben = 10000;
    tagesGeld.iban = "DE23654321";
    tagesGeld.bic = "A4GS2";
  }
}
```

10.5 Objektmethoden

Damit ein Konto nicht nur Werte speichern kann, sondern auch ein Verhalten hat, verwendet man Methoden. Bisher waren alle Methoden `static`. Das muss sich ändern. Damit ein Objekt Methoden aus einer Klasse übernehmen kann, müssen diese nicht `static` sein. Die Methode `ueberweisung` ist hier das Beispiel:

```
//Datei Konto.java
public class Konto{
  String inhaber;
  double guthaben;
  String iban;
  String bic;

  public void ueberweisung(double x){
    guthaben = guthaben - x;
  }
}
```

Nachdem die Methode in der Klasse `Konto` implementiert wurde, können alle Objekte dieser Klasse Überweisungen durchführen:

Jedes Objekt hat eine eigene Methode. Das Objekt in der Variablen `giro` kann mit `giro.ueberweisung(1000);` das `guthaben` um 1000 mindern. Mit `tagesGeld.ueberweisung(10);` wird das Guthaben auf dem Tagesgeld-Konto um 10 minimiert:

```
//Datei BankProgramm.java
  //Quellcode
    Konto giro = new Konto();
    //Quellcode
    giro.ueberweisung(1000);

    Konto tagesGeld = new Konto();
    //Quellcode
    tagesGeld.ueberweisung(10);

    System.out.println(giro.betrag);       //gibt 49000 aus
    System.out.println(tagesGeld.betrag);  //gibt 9990 aus
  //Quellcode
```

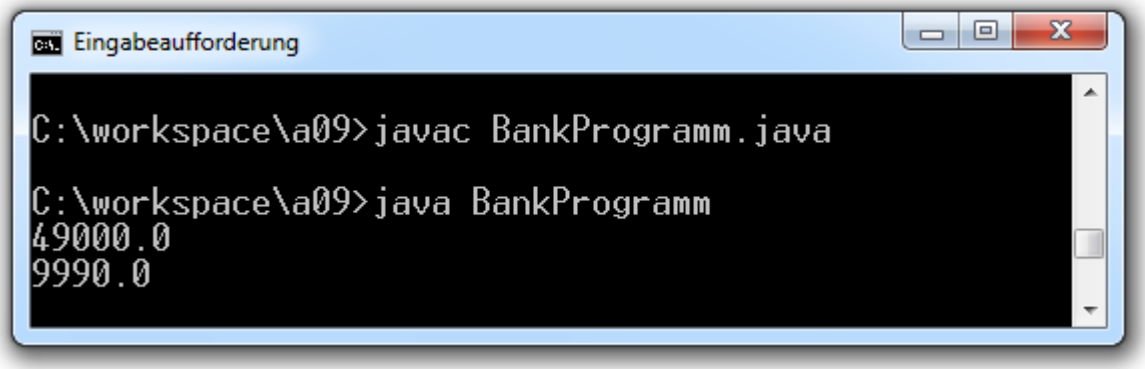

Abb. 30: Objektmethoden der Klasse „Konto“ ausführen

10.6 Der Konstruktor

Der Konstruktor wird verwendet, um Objekte zu initialisieren. Er hat den gleichen Namen wie die Klasse in der er steht, eine Klammer für Parameter und einen Block in dem Befehle stehen:

```
//Datei Konto.java
public class Konto{
  String inhaber;
  double guthaben;
  String iban;
  String bic;

  //Konstruktor
  Konto(String f, double g, String h, String i){
    inhaber = f;
    guthaben = g;
    iban = h;
    bic = i;
  }
  //Quellcode
}
```

Hier hat der Konstruktor `Konto` mehrere `String`- und einen `double`-Wert als Parameter. Sobald der Konstruktor geschrieben wurde, muss man bei der Erzeugung eines Objekts Werte an ihn übergeben. Im Bank-Programm muss das Objekt in der Variablen `giro` deshalb mit `Konto giro = new Konto("Max Muster", 50000, "DE131232123", "A4GS2");` erstellt werden. In der Klammer stehen die Parameter für den Konstruktor. Ein Objekt ohne Parameter kann jetzt nicht mehr erstellt werden:

```
//Datei BankProgramm.java
```

```
public class BankProgramm{
  public static void main(String[] args){
    Konto giro = new Konto("Max Muster", 50000, "DE131232123",
"A4GS2");
    Konto tagesGeld = new Konto(); //Verboten
  }
}
```

10.7 private

Mit dem Wort `private` können Sie Eigenschaften und Methoden einer Klasse vor anderen Klassen verstecken:

```
//Datei Konto.java
public class Konto{
  private String inhaber;
  private double guthaben;
  private String iban;
  private String bic;
  //Quellcode
```

Wenn man die Eigenschaften in `Konto` mit `private` anlegt, hat die Klasse `BankProgramm` keinen Zugriff mehr auf die Eigenschaften von Objekten, die aus der Klasse `Konto` erzeugt werden:

```
//Datei BankProgramm.java
public class BankProgramm{
  public static void main(String[] args){
    Konto giro = new Konto("Max Muster", 50000, "DE131232123",
"A4GS2");

    a.guthaben = a.guthaben - 230; //Verboten
    a.ueberweisung(230); //Erlaubt
  }
}
```

Nachdem `BankProgramm` das Objekt in der Variablen `giro` über seinen Konstruktor initialisiert hat, kann es die Methode `ueberweisung` nutzen, da diese `public` ist. Eine nachträgliche Manipulation der Eigenschaften mit *Variablenname.Eigenschaft* ist nicht möglich.

10.8 Lessons learned

Objekte sind alle Dinge, die man sich vorstellen kann. Ein Objekt hat Eigenschaften und ein Verhalten. Um in Java ein Objekt zu erzeugen, muss man zuerst eine Klasse schreiben. Die Klasse ist der Bauplan des Objekts (hier war es die Klasse `Konto`). Mit Variablen werden die Eigenschaften eines Objekts beschrieben:

```
//Datei Konto.java
public class Konto{
  String inhaber;
  double guthaben;
  String iban;
  String bic;
}
```

Unterhalb der Eigenschaften können Methoden angelegt werden. Die Methoden beschreiben das Verhalten eines Objekts und müssen nicht `static` sein:

```
public void ueberweisung(double x){
  guthaben = guthaben – x;
}
```

Um aus einer Klasse ein Objekt zu erzeugen, muss man eine zweite Klasse mit main-Methode schreiben (hier war es `BankProgramm`):

```
Konto giro = new Konto();
```

In der main-Methode wird mit Konto giro = new Konto(); ein Objekt erzeugt, das in der Variablen `giro` gespeichert wird und vom Datentyp `Konto` ist. Wenn man einen Konstruktor definiert, kann bzw. muss das Objekt direkt bei seiner Erzeugung initialisiert werden:

```
Konto giro = new Konto(("Max Muster", 50000, "DE131232123",
"A4GS2");
```

Literaturverzeichnis

1. **Balzert, Heide:** Objektorientierung in 7 Tagen – Vom UML-Modell zur fertigen Web-Anwendung, Heidelberg; Berlin: Spektrum Akademischer Verlag 2000.
2. **Balzert, Helmut:** Java: Der Einstieg in die Programmierung, 2. Auflage, Herdecke; Witten: W3L GmbH 2008.
3. **Daum, Berthold:** Java 6 – Programmieren mit der Java Standard Edition, München: Addison-Wesley Verlag 2007.
4. **Deininger, Marcus; Faust, Georg; Kessel, Thomas**: Java leicht gemacht – Eine verständliche Einführung in die Programmiersprache, München: Oldenbourg Wissenschaftsverlag GmbH 2009.
5. **Doberkat, Ernst-Erich; Dißmann, Stefan:** Einführung in die objektorientierte Programmierung mit Java, 2., überarbeite Auflage, München; Wien: Oldenbourg Wissenschaftsverlag GmbH 2002.
6. **Dornberg, Rolf; Telesko, Rainer:** Java-Training zur Objektorientierten Programmierung, München: Oldenbourg Wissenschaftsverlag GmbH 2010.
7. **Flanagan, David:** JAVA in a Nutshell, 4. Auflage, Köln: O´Reilly Verlag GmbH & Co. KG 2003.
8. **Krüger, Gido; Stark, Thomas:** Handbuch der Java Programmierung, 6., aktualisierte Auflage, München: Addison-Wesley Verlag 2009.
9. **Sedgewick, Robert; Wayne, Kevin:** Einführung in die Programmierung mit Java, München: Pearson Deutschland GmbH 2011.
10. **Wolff, Christian:** Einführung in Java – Objektorientiertes Programmieren mit der Java 2-Plattform, Stuttgart; Leipzig: Teubner 1999.

Impressum

Reihe: **Arbeitspapiere Wirtschaftsinformatik** (ISSN 1613-6667)

Bezug: https://wi.uni-giessen.de

Herausgeber: Prof. Dr. Axel Schwickert
Prof. Dr. Bernhard Ostheimer

c/o Professur BWL – Wirtschaftsinformatik
Justus-Liebig-Universität Gießen
Fachbereich Wirtschaftswissenschaften
Licher Straße 70
D – 35394 Gießen
Telefon (0 64 1) 99-22611
Telefax (0 64 1) 99-22619
eMail: Axel.Schwickert@wirtschaft.uni-giessen.de
https://wi.uni-giessen.de

Ziele: Die Arbeitspapiere dieser Reihe sollen konsistente Überblicke zu den Grundlagen der Wirtschaftsinformatik geben und sich mit speziellen Themenbereichen tiefergehend befassen. Ziel ist die verständliche Vermittlung theoretischer Grundlagen und deren Transfer in praxisorientiertes Wissen.

Zielgruppen: Als Zielgruppen sehen wir Forschende, Lehrende und Lernende in der Disziplin Wirtschaftsinformatik sowie das IT-Management und Praktiker in Unternehmen.

Quellen: Die Arbeitspapiere entstehen aus Forschungs-, Abschluss-, Studien- und Projektarbeiten sowie Begleitmaterialien zu Lehr-, Vortrags- und Kolloquiumsveranstaltungen der Professur BWL – Wirtschaftsinformatik, Prof. Dr. Axel Schwickert, Justus-Liebig-Universität Gießen sowie der Professur für Wirtschaftsinformatik, insbes. medienorientierte Wirtschaftsinformatik, Prof. Dr. Bernhard Ostheimer, Fachbereich Wirtschaft, Hochschule Mainz.

Hinweise: Wir nehmen Ihre Anregungen zu den Arbeitspapieren aufmerksam zur Kenntnis und werden uns auf Wunsch mit Ihnen in Verbindung setzen.

Falls Sie selbst ein Arbeitspapier in der Reihe veröffentlichen möchten, nehmen Sie bitte mit einem der Herausgeber unter obiger Adresse Kontakt auf.

Informationen über die bisher erschienenen Arbeitspapiere dieser Reihe erhalten Sie unter der Web-Adresse https://wi.uni-giessen.de/