

# Modul 319/403

**Applikationen entwerfen und implementieren**

# Willkommen im Modul 319

Willkommen auf der Website, die Sie in das Programmieren einführt, und zwar je nach EFZ-Fachrichtung mit den Modulen 319 und 403.

Hier finden Sie Informationen und Unterlagen zum Unterricht.



## TIP

- Ausserdem finden Sie auch weiterführende Links.
- Auf diese werden Sie im Unterricht nicht ausdrücklich hingewiesen.

## Webseite

### Lektionen

Unter [Lektionen](#) finden Sie die Übersicht aller Lektionen inklusive der Inhalte die pro Woche anstehen.

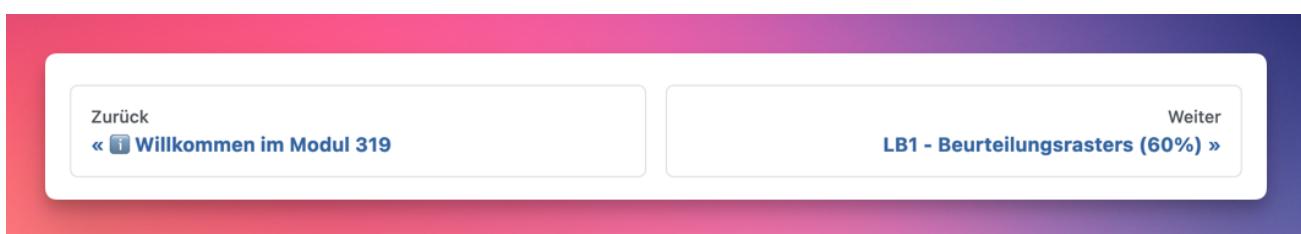
### Navigation

Links ist die Unternavigation sichtbar.

Die **Nummern pro Hauptthema entsprechen der Wochennummer** in welchem das Thema besprochen wird.

- **1a - Vom Algorithmus zum Programm** wird also in der **Woche 1** relevant sein.
- **7 - Arrays und for(each)** erst in der **Woche 7**

Pro Seite wird unten einen "Zurück" und "Weiter" Link dargestellt. Ich könnt also theoretisch alle Themen nacheinander durchgehen und die zugehörigen Aufgaben machen.



### Musterlösungen

Musterlösungen befinden sich immer in einer ausklappbaren Box. Ihr könnt alle Lösungen direkt auf der Webseite abrufen. Die Box ist eingeklappt, damit Ihr es hoffentlich zuerst selber versucht!

## PROGRAMMIEREN BRAUCHT ÜBUNG!

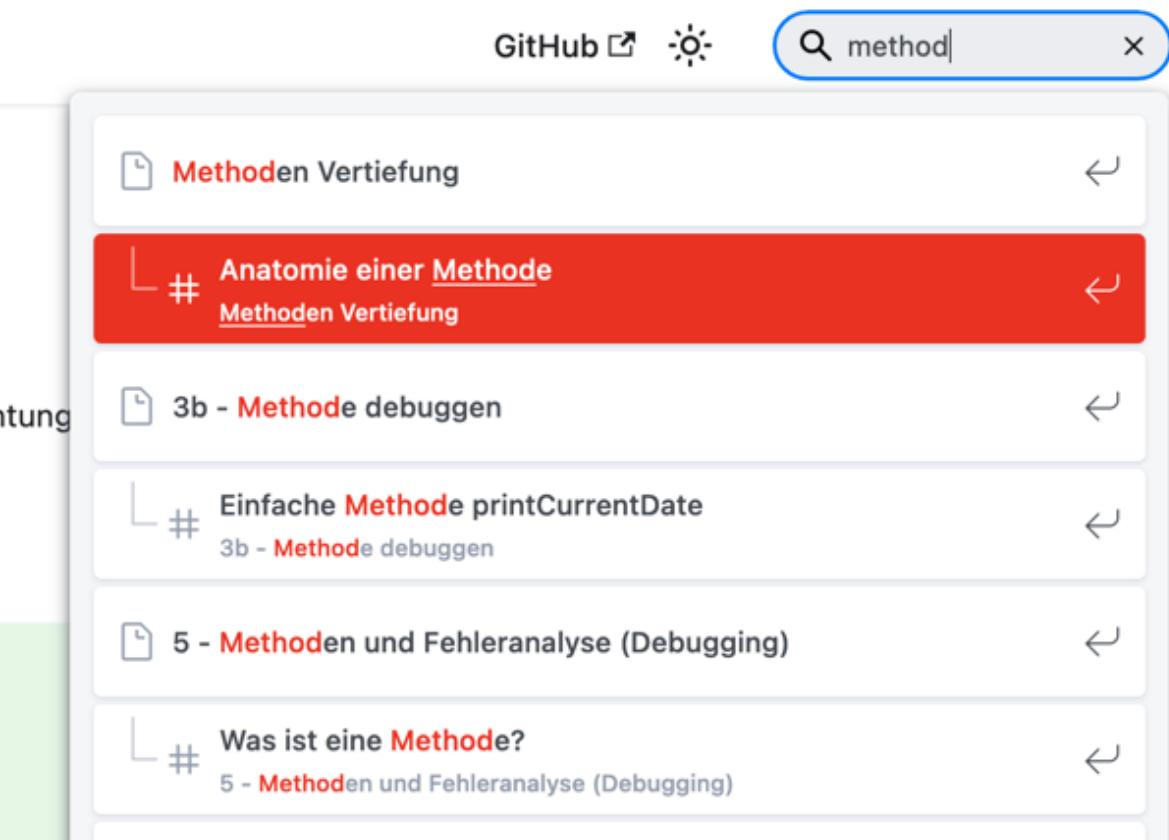
Nur mit **Repetition und Übung** werdet Ihr Programmieren lernen.

Vergleicht es mit einem Sport! Es ist gut ein Buch über Skaten zu kaufen. Natürlich helfen Tips es von Anfang an richtig zu machen.

**ABER:** Ohne selber zu üben schafft man keinen Olly (zumindest ich kann noch keinen :D) und schon gar keinen 1080.

## Suche

Die Webseite ist komplett durchsuchbar. Dies könnt Ihr machen, indem ich `ctrl-k` drückt oder einfach oben rechts im Suchfeld nach Begriff sucht.



The screenshot shows a search interface with a GitHub logo and a search bar containing the word "method". Below the search bar is a list of search results, each with a file icon and a title. The titles are: "Methoden Vertiefung", "# Anatomie einer Methode Methoden Vertiefung", "3b - Methode debuggen", "Einfache Methode printCurrentDate 3b - Methode debuggen", "5 - Methoden und Fehleranalyse (Debugging)", and "# Was ist eine Methode? 5 - Methoden und Fehleranalyse (Debugging)". Each result has a back arrow icon to its right.

## Modul als PDF

Ich könnt den Inhalt der Webseite (Ohne Musterlösungen) als [PDF downloaden](#). Dies geht auch über den Button oben rechts.

So könnt Ihr mit einem beliebigen PDF viewer beliebige Kommentare und Notizen machen. Auch ermöglicht es so die Archivierung für späteres Nachschlagen.

Dieses Feature ist momentan experimentiel! Erwartet nicht das eleganteste PDF. Es wird mit der Zeit verbessert.

## Entwicklungsumgebung

An der Schule steht Ihnen grundsätzlich ein Rechner mit einer virtuellen Windows 10 Maschine zur Verfügung. Darauf sind alle nötigen Programme installiert. Alternativ dürfen Sie natürlich die Programme auch auf Ihrem privaten Gerät installieren und verwenden.

Zusätzlich stellen wir Ihnen eine virtuelle Maschine zur Verfügung, die Sie verwenden dürfen. Diese haben Sie am besten auf einer externen Festplatte dabei. So haben Sie an der Schule wie auch zu Hause immer die selbe Umgebung.



### TIP

- Link zu Classroom: [Google Classroom](#)
- Dann oben rechts auf das + -> Für Kurs anmelden mit dem Kurscode: **v6charw**
- Unter Kursunterlagen finden Sie die virtuelle Maschine **bmWP3** (Standard-VM für Java Module)

## Modulidentifikation

PDF speichern

## LBV

PDF speichern

## Quellen

- Offizielle Webseite: Modulidentifikation
- Offizielle Webseite: LBV

## Historie

### Modul 403

- Alte Modulidentifikation
- ALTE LBV

# LB1 - Beurteilungsraster (60%)

Ab der 2. Woche bis **spätestens Ende 7. Woche** haben Sie Zeit, die Themen des Beurteilungsrasters nachzuweisen.

## ⚠ VERSPÄTETE ABGABE

Wird ein Dokument verspätet abgegeben wird **pro Dokument 0.25 Punkte** abgezogen

## Google Classroom

Die Dokumente zum Beurteilungsraster befinden sich im [Google Classroom](#). Diese können dort bearbeitet und abgegeben werden.

### ❗ KURSCODE:

- Für die Klassen INB23A und B: **xnaua3a**

## Beurteilungsraster

Im Classroom findet Ihr den Link zum Ordner mit den Beurteilungsrastern. In diesem befinden sich meine Beurteilungen welche von jedem für sich eingesehen werden kann.

Es existiert pro Person eine Datei unter folgendem Namen.

- *Beurteilungsraster\_{Name}\_{Vorname}*

## Aufbau

Das Raster ist in **vier Teile Unterteilt**, wobei jeder Teil eine mögliche Punktzahl besitzt.

1. **A (schriftlich)**: kann das Thema differenziert beschreiben (Grundanforderungen erfüllt)
2. **B (mündlich)**: kann am Beispiel eines vorgegebenen Auftrags erklären (erweiterte Anforderungen erfüllt)
3. **C (schriftlich)**: kann zu dem Thema ein eigenes Beispiel mit eigenen Ideen entwickeln und beschreiben (fortgeschrittene Anforderungen erfüllt)
4. **Lernweg (schriftlich)**: Die letzte Spalte des Rasters beurteilt Ihren Lernweg mit Planung sowie Reflexion je Thema.

### **⚠ SCHREIBT DEN "TEIL C" NICHT AB (UND FORMULIERT UM)**

Im mündlichen Gespräch (B), welches für jedes Thema geführt wird, finde ich sehr schnell heraus ob ein eventueller fortgeschritten Teil (C) überhaupt von einer Person stammen kann oder nicht!

### **⌚ Wird der mündliche Teil (B) nicht bestanden ist automatisch auch der schriftliche Teil (C) nichtig.**

- PS: Das heisst nicht, dass Ihr euch nicht gegenseitig helfen dürft. Natürlich! verstehen solltet Ihr aber was Ihr abgibt ;)

## Punktevergabe

- Es gibt vier Themen mit je vier Teilen. Das bedeutet es gibt **16 Teilpunkte**.
- Die Teilpunkte **entsprechen Notenpunkten**. Die Summe der Teilpunkte ergibt somit direkt die Note.
- Die Punkte pro Teil und Thema werden **entweder ganz oder garnicht** erreicht! Nur bei Ausnahmen werden Teilpunkte vergeben.

## Themendokumente

Im Classroom finden Sie **pro Thema ein GoogleDoc für Sie persönlich**, welches Sie direkt bearbeiten und auch darüber abgeben können.

Es existiert für alle vier Themen eine Datei. Folgend sind die Titel der Dateien und den zugehörigen Aufgabenseiten angegeben:

- **Thema 1 - Datentypen und Variablen**
  -  : 2b - Variablen & Datentypen
- **Thema 2 - Selektion und Operatoren**
  -  : 2b - Variablen & Datentypen (Rechnen und Operatoren)
  -  : 3 - Kontrollstrukturen (Präsentation)
- **Thema 3 - Methoden**
  -  4 - Methoden und Fehleranalyse
- **Thema 4 - Debugging und Fehlersituationen**
  -  4 - Methoden und Fehleranalyse

### **⚠ VERGESST NICHT DEN LB2**

Auch wenn das Beurteilungsraster nicht alle Themen abdeckt, bitte möglichst alle Themen und Aufgaben bearbeiten. Es könnte ja sein dass z.B. Schleifen und Arrays im LB2 relevant werden.

## Aufbau

Jedes Dokument ist in drei Teile eingeteilt ([siehe Screenshot unten](#)):

1. Am Anfang werden die **drei Schwierigkeitsstufen A, B und C beschrieben (orange)**.

Rechts sind Gefühls-Indikatoren vorhanden, hier müsst Ihr mit einem Klick angeben wie ihr euch dabei Gefühlt habt.

2. Danach folgt der **Lernweg (gelb)**, dieser ist in zwei Teile unterteilt:

**Am Anfang** des Dokuments muss beschrieben werden wie man genau vorgehen möchte um das Thema sich selbstständig anzueignen.

**Am Ende** wird mit einer Reflexion beschrieben was gut gelungen ist, welche Schwierigkeiten aufgetreten sind und wie diese angegangen wurden.

3. Schlussendlich gibt es drei Boxen, zur Beschreibung der **drei Schwierigkeitsstufen (grün)**.

Hier müsst Ihr für die Schwierigkeitsstufen A und C euer Beispiel-Code einfügen und dieser schriftlich beschreiben.

Für die Schwierigkeit "Erweitert (B)" müsst Ihr nichts schreiben. Es empfieilt sich jedoch zur Vorbereitung des mündlichen Gesprächs notizen zu machen.

### **ⓘ KOMPLEXER CODE?**

Hat ein Beispiel zu viel Code, sodass es in der Box unübersichtlich wird, darf Ihr auch eine Datei mit dem Code in Classroom hochladen und **verlinken**.

- **Nur korrekt verlinkte Dateien werden akzeptiert!** (click darauf macht sie auf.)

## Beispiel Themendokument zur Beschreibung (Teil 1)

The screenshot shows a digital form for a competency certificate. At the top left is the logo of Berufsbildungszentrum Baselland, IT-Ausbildung Pratteln, Modul 319 / 403. At the top right is the logo of BASEL LANDSCHAFT, BILDUNGS-, KULTUR- UND SPORTDIREKTION, BERUFSBILDUNGSZENTRUM BASELLAND. The main title is 'Kompetenznachweis Thema 1 Datentypen und Variablen <Ihr Name>'. The form is divided into three sections:

- 1 Kompetenznachweis:** A box containing a list of tasks:
  - A: kann Datentypen (mit Unterscheidung primitive Datentypen und String) und Variablen beschreiben
  - B: kann Datentypen und Variablen am von den Übungen abgeleiteten Beispiel erklären und variieren
  - C: zeigt Datentypen und Variablen am selbst erarbeiteten Beispiel (sinnhafte Variablennamen)Followed by a row of five smiley face icons with the number '0' next to each, and an orange arrow pointing to the right with the text 'klick mich!'.
- 2 Lernweg - Planung / Vorgehen:** A box containing two numbered questions:
  1. Welche Schritte plane ich, um mir dieses Thema anzueignen?
  2. Mit welcher Systematik gehe ich an die Programmieraufgaben heran?
- 3 Grundlagen (A), Erweitert (B), Fortgeschritten (C):** Three large empty rectangular boxes for writing responses.

At the bottom is a section for reflection:

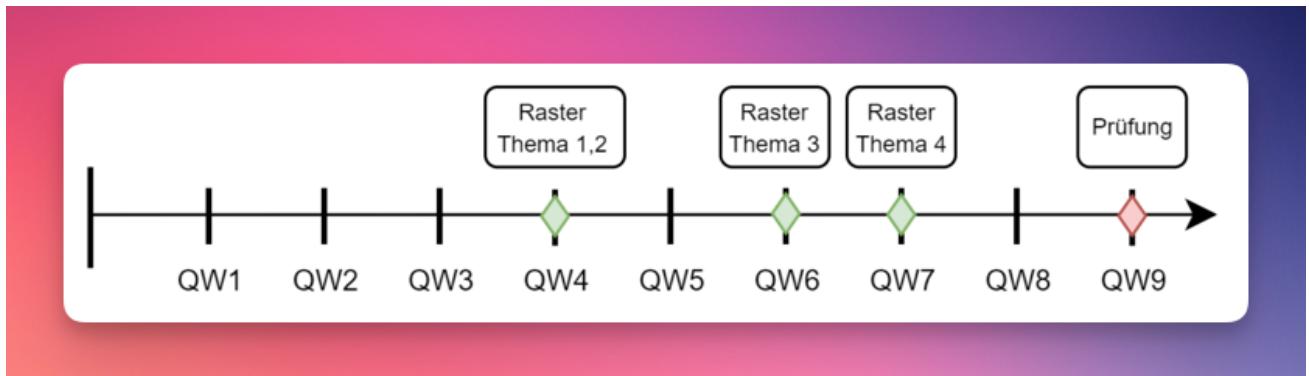
**Lernweg - Reflexion**

1. Was ist mir gut gelungen?
2. Wie bin ich vorgegangen, dass es mir gelungen ist?
3. Womit hatte ich Schwierigkeiten?
4. Wie habe ich auf die Schwierigkeiten reagiert?

## Art der Leistungserbringung

- A: **schriftlich** in das GoogleDoc über Classroom
- B: **mündlich** bei der Lehrperson, dabei sind die Meilensteine einzuhalten (Die Lehrperson kann jederzeit Fragen zu Ihren Erklärungen stellen, Ihr Beispiel variieren oder Ihnen zu weiteren Erläuterungen andere Beispiele vorlegen.)
- C: **schriftlich** in das GoogleDoc über Classroom (ausser Thema 4, dies wird ebenfalls mündlich bei der Lehrperson vorgestellt)
- Lernweg: **schriftlich** in das GoogleDoc über Classroom

## Meilensteine



# LB2 - Prüfung (40%)

Findet am Ende des Moduls statt

- Die gesamte Prüfungsdauer beträgt **90 Minuten**.
- Sie entscheiden, zu welchem Zeitpunkt Sie von Teil 1 der Prüfung zu Teil 2 wechseln.

## 1. Multiple-Choice-Fragen

- Auf Papier
- **Computer nicht verwenden**

## 2. Programmieraufgaben

- Java Aufgaben
- Zusätzlich vermittelter Stoff
- Wird **am Computer** gelöst

## Vorbereitung

- **Lernprodukte**, mit denen Sie das Beurteilungsraster belegt haben.
- **Übungen**, die Sie in diesem Modul bearbeitet haben.
- **Repetitionsaufgaben**, welche die verschiedenen Themen des Moduls kombinieren.

### HILFSMITTEL

- Das **Dokument zum Modul 403**, welches im Thema *Allgemein -> Hilfsmittel* heruntergeladen werden kann.
- Eine **eigene Zusammenfassung**, welche Sie selbst während des Moduls erstellt haben, verwenden.

 **Programmieren muss man üben!**. Ein Spick hilft oft nicht viel, wenn man es nicht verstanden hat.

# 1a - Vom Algorithmus zum Programm

Was sind überhaupt Programme? und was haben diese mit Algorithmen zu tun?

## ⌚ Ziele

- Sie können erklären, was ein Algorithmus ist und welches die wichtigsten Kontrollstrukturen sind.

## 💻 Präsentation

[Open in Browser](#) | [download PDF](#)

## Code.org

Einige Programmierer geben Einblick darein, was sie am programmieren faszinierte. Bei welchen einfachen Aufgaben die Herausforderungen bestanden. Welche Aufgaben sie zuerst lösen konnten. Und was sie über die Informatik und Bedeutung von Programmen und Algorithmen denken.

"Everybody in this country should learn how to program a computer... because it teaches you how to think." -- Steve Jobs



## Vom Algorithmus zum Programm?

Viele Informatiker argumentieren, dass das Erlernen des Programmierens ohne einen Computer anfangen sollte. Auch im Unterricht zeigt sich oft, dass Einsteiger viel zu schnell aufhören, über die grundsätzliche Aufgabenstellung für ein Programm nachzudenken.

Ein Softwareprogramm Dient immer einer Problemlösung, ohne Nutzen kein Programm.

## Dazu Zitate einiger Informatiker:

"Ich glaube eines der Missverständnisse ist, dass viele bei Informatik zuerst an Computer und Bildschirme denken. Für mich ist Informatik aber etwas anderes: Es ist eine Denkschule, die mir hilft, Probleme auseinanderzunehmen und in einzelnen Schritten zu lösen. Wenn ich eine Lösung vor Augen habe, muss ich sie genau durchdenken, denn eine kleine Variation kann alles verändern. Funktioniert die Lösung, frage ich mich, ob es noch einen einfacheren oder schnelleren Weg gibt. Das ist es, was algorithmisches Denken ausmacht. Und Algorithmen bestimmen – egal, ob wir es wahrhaben wollen oder nicht – zunehmend unsere Welt."

--ETHZ: Gaertner Informatik

"Wer ist sich schon bewusst, dass ‚digital‘ bedeutet, Information als Folge von Symbolen darzustellen? Der erste grosse Schritt der Digitalisierung war die Erfindung der ersten Schriften vor Jahrtausende."  
--ETHZ: Hrmokovic

"Diese Fähigkeit wird auch als Computational Thinking bezeichnet, was deren Erfinderin, Jeannette Wing, 2006 beschrieb als: Computational Thinking is the thought process involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out."  
--Swissinformatics Magazine

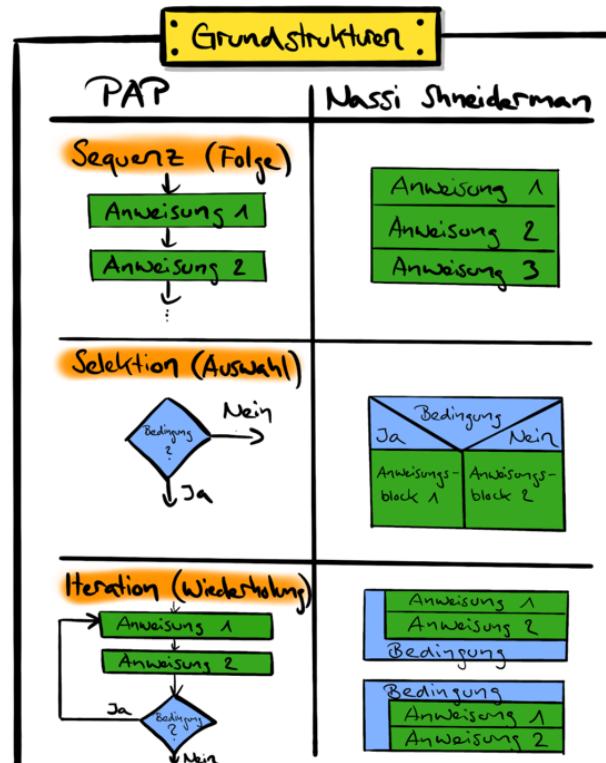
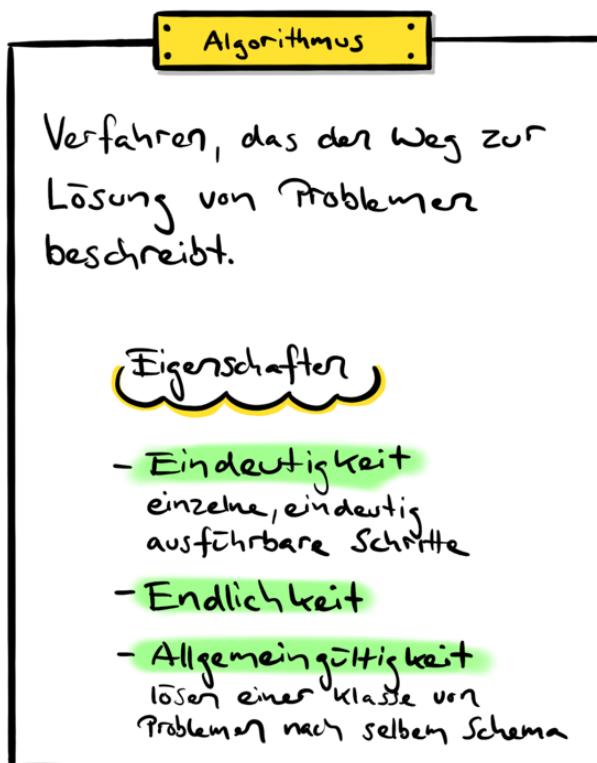
## 📝 Welche Tätigkeiten mit "Problemlösungsschritten" sind Ihnen bekannt?

Sie konnten im Unterricht diskutieren oder dies nun noch selbst als Gedankenexperiment nachholen, welche Tätigkeiten Ihnen einfallen, bei denen Sie Probleme analysiert, Lösungen gesucht und gefunden haben, und einen Bauplan entwickelt haben, um eine Aufgabe oder ein Problem zu lösen.

## Die Grundformen der Programmierung

Die folgenden Grundbegriffe wurden im Unterricht definiert

- Sequenz (Schritt für Schritt)
- Selektion / Bedingung (if, else, ...)
- Iteration (do, while, for, ...)
- Algorithmus
  - Spezifische Kombination von Sequenz, Bedingung und Iteration



## **Definition Algorithmus und Kontrollstrukturen**

In diesem Auftrag geht es darum, zu verstehen was Algorithmen überhaupt sind und welche grundlegenden Kontrollstrukturen sie mitdefinieren.

-  **Aufgabenblatt**

# 1b - HelloWorld Programm

Sie lernen mit der Entwicklungsumgebung [Eclipse](#) ein **HelloWord-Programm** zu schreiben.

## ⌚ Ziele

- Sie wissen, was der Umfang einer Entwicklungsumgebung ist.
- Sie können Eclipse starten, mit einem Workspace umgehen und Projekte anlegen.
- Sie können ein erste Programm, *HelloWorld*, schreiben und dessen Elemente beschreiben.
- Sie können die grundlegenden Konventionen und Kommentare anwenden.

## Infrastruktur zum Programmieren

Um zu programmieren ist es vorteilhaft eine Entwicklungsumgebung einzusetzen, weil viele technische notwendige Abläufe verlässlich im Hintergrund laufen, ohne dass man sich darum kümmern muss.

## Was ist eine Entwicklungsumgebung?

Als Entwicklungsumgebung wird ein Programm bezeichnet, welches es erlaubt:

- den Programmtext resp. Quellcode zu bearbeiten
- das Programm für die Ausführung vorzubereiten (*Kompilieren*)
- das Programm auszuführen und mit dem Programm zu interagieren
- weitere Konzepte und Dienst wie die Unterstützung zur Fehleranalyse bietet (*Debuggen*)

## Eclipse als Entwicklungsumgebung

Im Unterricht verwenden wir als Standard-Entwicklungsumgebung [Eclipse](#). Um auch zuhause mit Eclipse können Sie es von [hier](#) herunterladen. Eclipse ist eine offene (*Open Source*) Software und somit gratis für alle verwendbar.

- [🔗 Eclipse Download](#)

Zusätzlich zu Eclipse gibt es noch viele Andere, teilweise sogar modernere, Umgebungen. Diese sind jedoch häufig kostenpflichtig oder nicht so gut geeignet für Java.

- [JetBrains IntelliJ](#)
- [JetBrains Fleet](#)
  - Ist noch gratis, kann sich aber ändern!
  - Moderner als IntelliJ und Eclipse
- [Visual Studio Code](#)
  - Nicht so geeignet für java!
  - Sonst jedoch super!

Klicken Sie nun auf [Weiter um Eclipse zu installieren](#)

## 📺 Erklärvideos

### Java Grundlagen auf Studyflix

Erklärung der wichtigsten Grundlagen in Java:

- Syntax (Klassen, Kommentare)
- Bugs
- Compiler



### Java Main Methode auf Studyflix

- `args` und `void` in Java
- `public static void main`



# Entwicklungsumgebung

In dieser Aufgabe lernen Sie wie, Sie die Entwicklungsumgebung *Eclipse* aufsetzen, damit Sie Java Programme erstellen und ausführen können.

## 1. Eclipse herunterladen und installieren

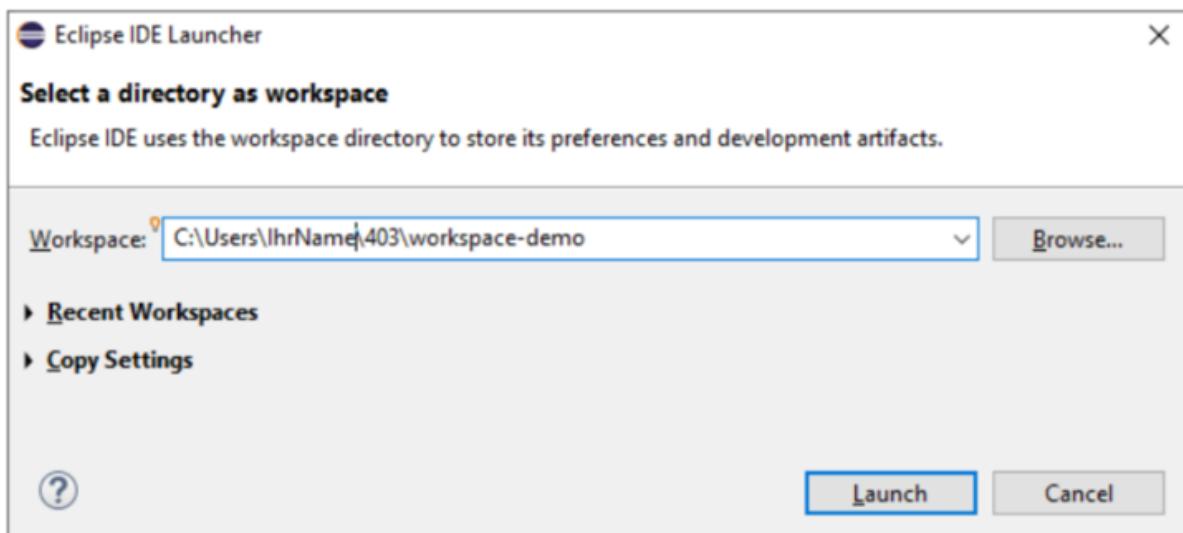
Falls Sie mit Ihrem **eigenen Computer** arbeiten, müssen Sie zuerst *Eclipse* herunterladen und installieren. Eclipse installiert automatisch die aktuelle Java version.

- [Eclipse Download](#)

**(i) WENN SIE AUF EINEM SCHULCOMPUTER ODER DER WINDOWS VM ARBEITEN, KÖNNEN**  
sie diesen Schritt überspringen.

## 2. Eclipse starten und einen Workspace wählen

- Starten Sie nun Eclipse auf Ihrem Computer.
- Als erstes müssen Sie nun den Speicherort für einen sogenannten Workspace wählen.
  - Der Workspace ist das Arbeitsverzeichnis von Eclipse. Darin können mehrere Projekte erstellt werden
  - Am Schulcomputer wählen Sie die **angelegte Modulstruktur in Ihrem Netzwerk-Verzeichnis**, damit Ihnen diese von Woche zu Woche zur Verfügung steht.
  - Abschliessend betätigen Sie «Launch».



- Nun sollte Eclipse starten, und Sie sind parat für ihr erstes "Hello World" ☺

 **TIP**

- Falls Sie zuhause daran arbeiten möchten, können Sie den **Workspace entsprechend nach dem Unterricht kopieren.**
- 📱 Wenn Sie **einen Laptop** besitzen, arbeiten Sie lieber damit.

 **VERMEIDEN SIE DATENVERLUST!**

- Wenn Sie am Schulcomputer nicht ihr Netzwerk-Verzeichnis wählen verlieren Sie Ihre Arbeit!
- Wenn Sie mit Ihrem Laptop arbeiten, kopiert den Workspace nach jedem Unterricht auch auf Ihr Netzwerk-Verzeichnis.
- Sie sind eigenverantwortlich über Ihre Daten
- Wer Git rsp. Github / Gitlab kennt, darf dies gerne verwenden (in einem privaten repo)

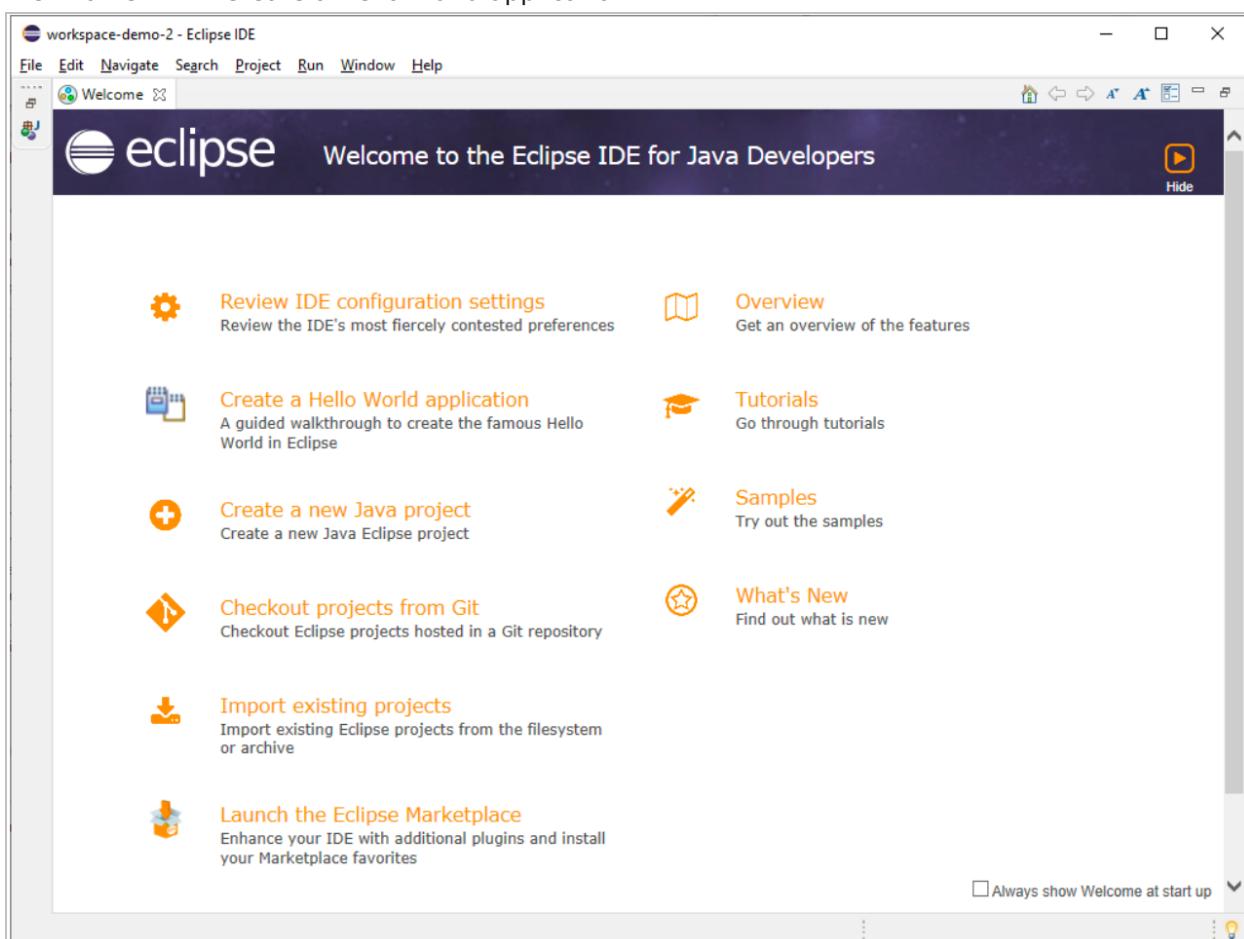
# Hello World Programm

Das «Hello-World»-Programm ist das wohl berühmteste Programm für Programmierer. Es ist das erste Programm, dass man üblicherweise programmiert, wenn man mit dem Programmieren anfängt oder eine neue Sprache (und Umgebung) beginnt.

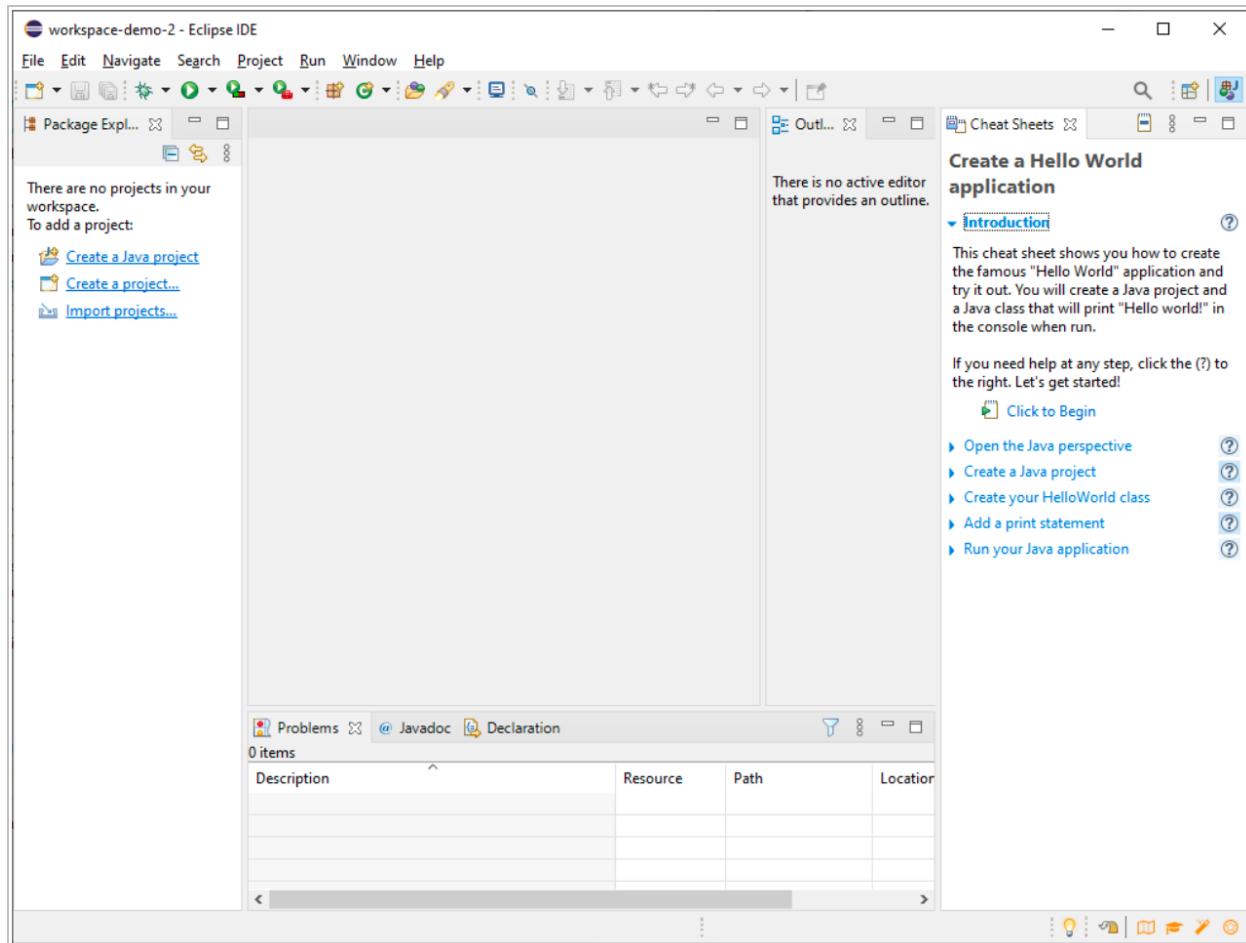
## 1. Hello-World Tutorial in Eclipse starten

Nach dem ersten Start mit einem neuen Workspace zeigt Eclipse den folgenden Splash-Screen.

- Hier wählen wir «Create a Hello World application».

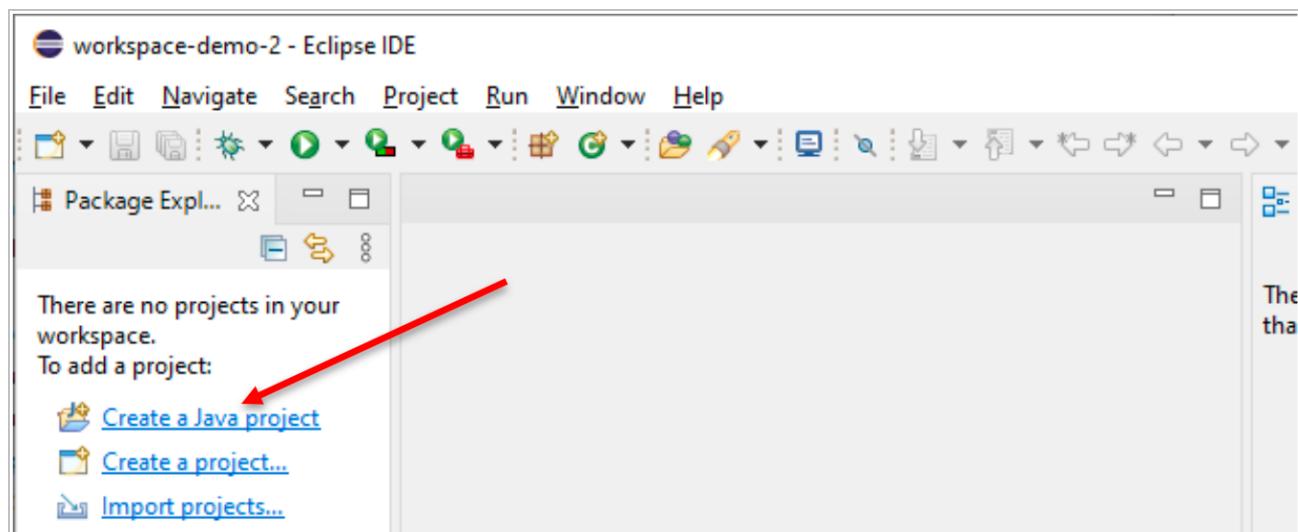


- Dann sieht Eclipse folgendermassen aus. Rechts werden wir nun auf Wunsch durch alle wesentlichen Schritte geleitet:

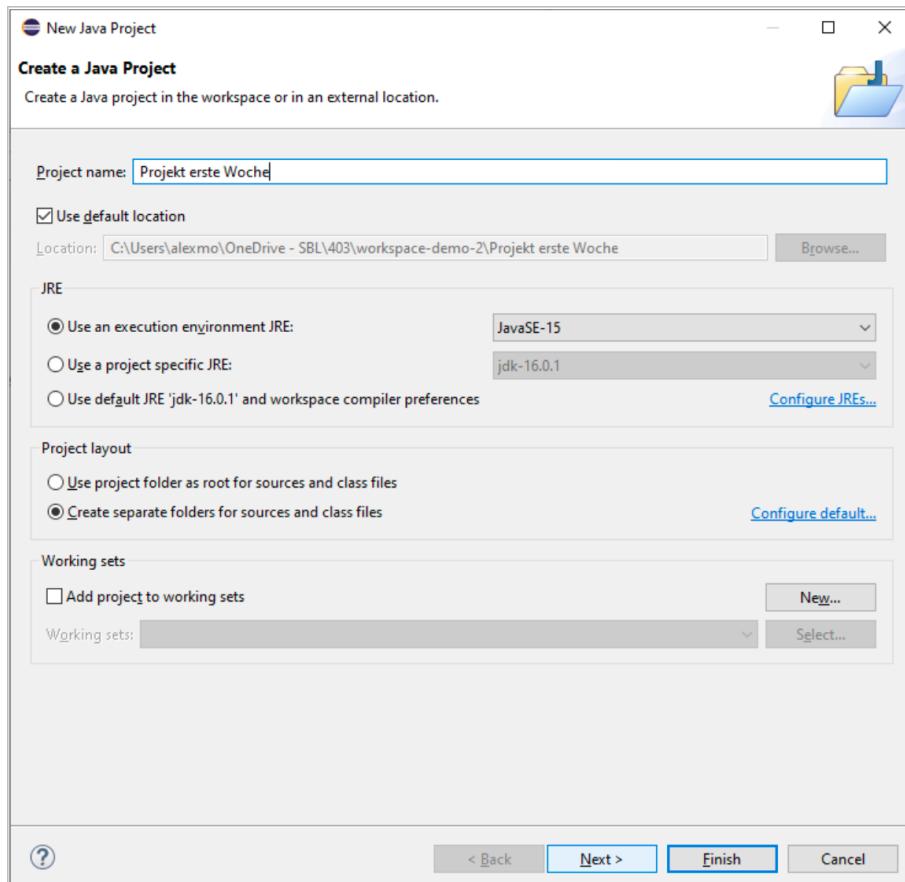


## 2. Ein Java Projekt erstellen

Wir wählen oben rechts «Create a Java project». Ohne diese Hilfe von Eclipse lässt sich das gleich erreichen via Menü **File** → **New** → **Java Project** (je nach Version erst Other, dann Java öffnen).

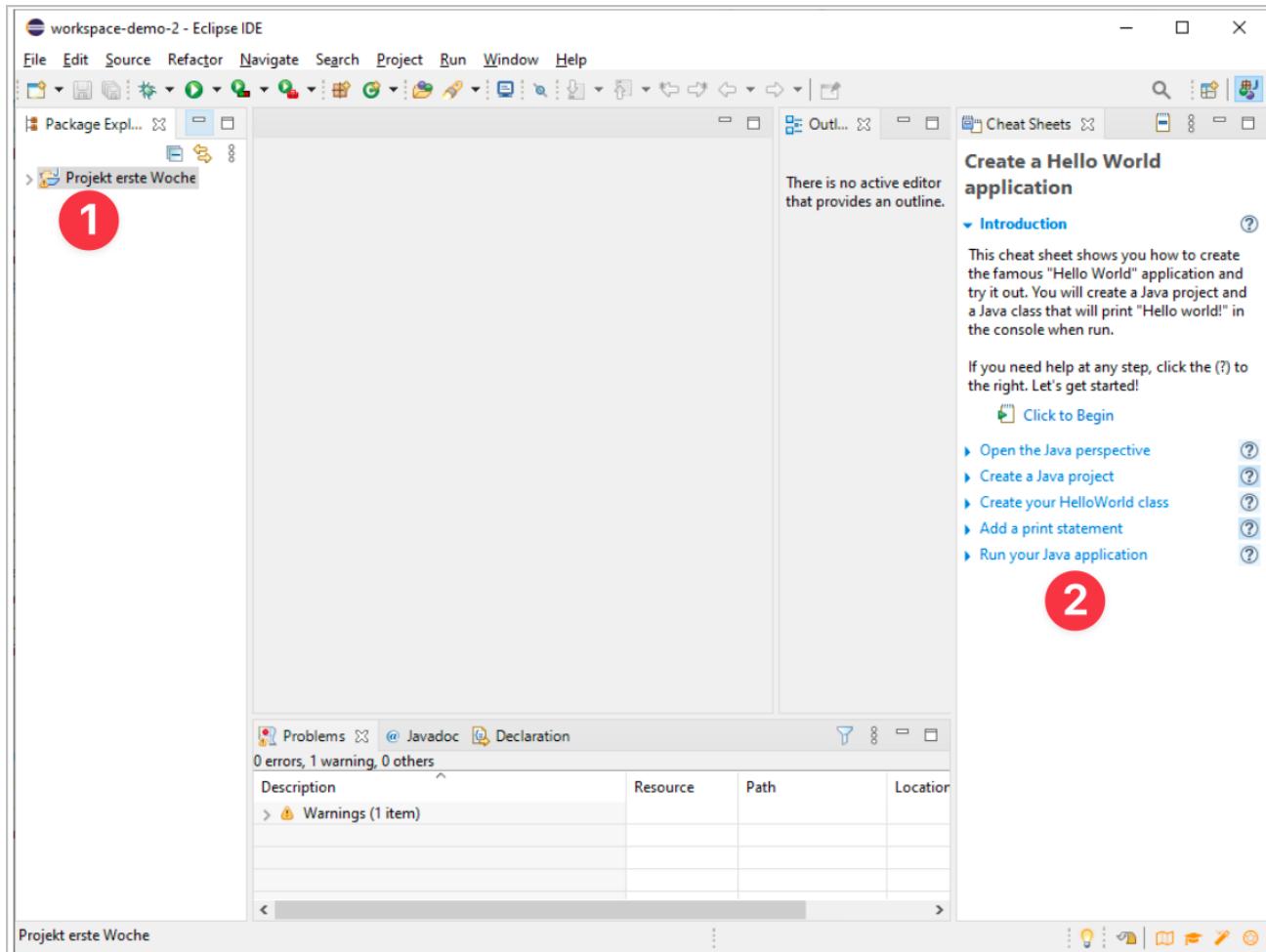


- Wir geben als Projektname «**Projekt erste Woche**» ein, und lassen die anderen Einstellungen unverändert und klicken dann auf **Next**
- Im nächsten Dialog drücken wir direkt auch «Finish».
- Die Rückfrage «Create module-info.java» beantworten wir mit «Don't create».

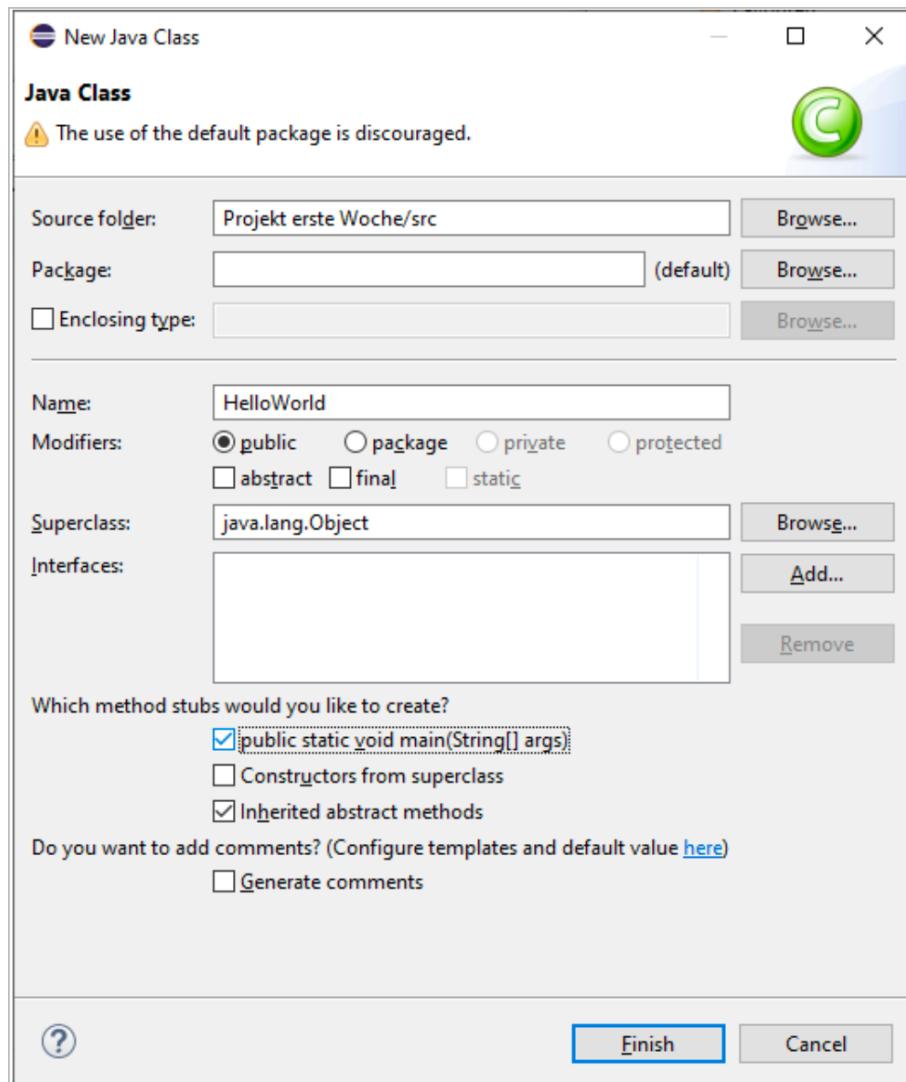


### 3. Eine Klasse anlegen

- Links (**Punkt 1**) ist der «Package Explorer», eine baumartige Darstellung unseres Workspace und seines Projektes.
- Rechts (**Punkt 2**) werden verschiedene Hilfsangebote angezeigt, die jeweils zur Anzeige entsprechender Hinweise führen, um die gewählte Aufgabe "Create Hello World application" zu lösen.

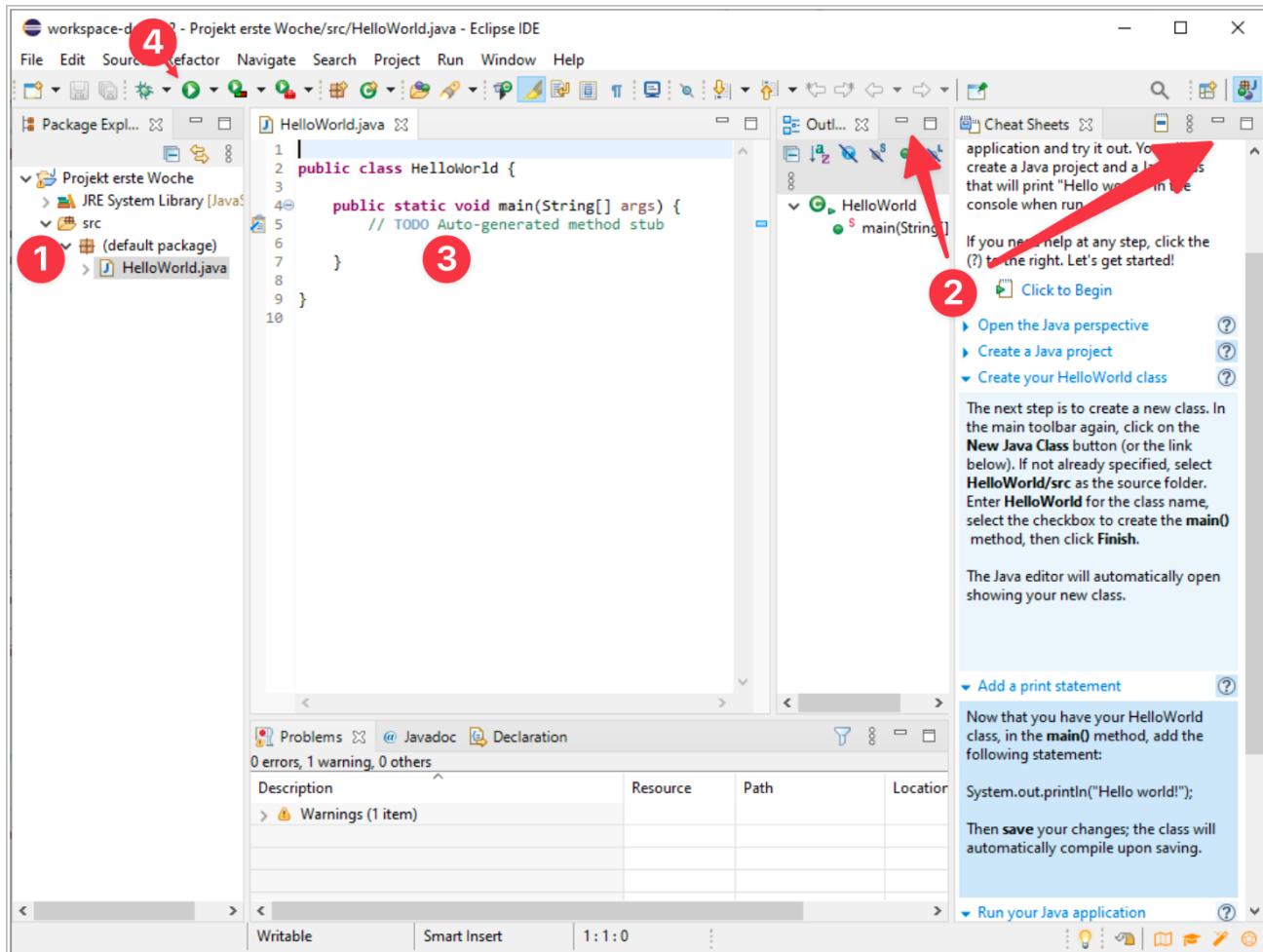


- Via Menü **File** → **New** → **Class** gelangen Sie in den folgenden Dialog, in welchem Sie den Namen "**HelloWorld**" eingeben und bei `public static void main(String[] args)` ein Häckchen setzen.
- In Java sind alle Programme in Klassen (Classes) organisiert, daher dieser Begriff.



## 4. "Hello World" programmieren

1. Links vom Projektnamen im «Package Explorer» wird dies ⌂ Zeichen angezeigt. Wenn Sie darauf klicken, dreht es sich, so dass es nach unten zeigt und damit den geöffneten Zustand anzeigt.
2. Auf der rechten Seite beanspruchen «Outline» und «Cheat Sheets» bereits mehr als 1/3 der Breite, und diese Darstellung kann man optimieren. Solche Views können maximiert, minimiert, geschlossen und gestapelt werden. Probieren Sie es aus.

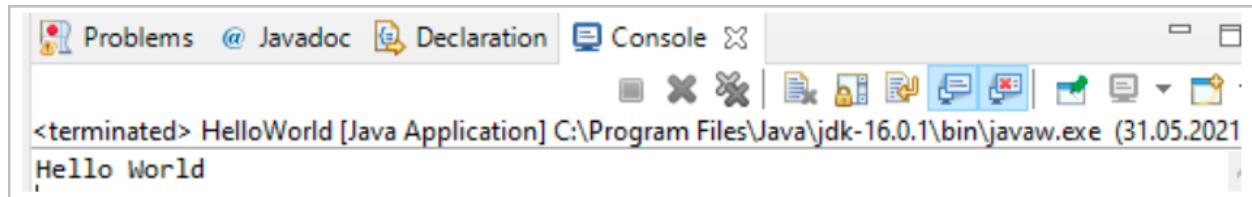


3. In der Mitte des Programms beginnt auf **Zeile 4** die **main-Methode**, die Java in allen ausführbaren Programmen erwartet und die Sie auf **Zeile 6** mit folgenden Quelltext ergänzen:

```
System.out.println("Hello World");
```

4. Klicken Sie nun auf den **Play-Button**, damit das Programm ausgeführt wird.

In der Console sehen Sie die Ausgabe des Programms:



**FALLS DIE CONSOLE NICHT SICHTBAR IST, KÖNNEN SIE DIESE WIE ALLE VIEWS VIA**  
**Menü *Window* → *Show View* → *Console*** wieder aufrufen

# Vom Quellcode zum ausführbaren Programm

## 1. Sourcecode, Compiler, Interpreter

Programme entstehen aus Quelltext, hier Java, der durch einen Compiler entweder in ein Maschinenprogramm oder in ein Zwischenformat übersetzt wird, und dann direkt von einem Computer, einem Betriebssystem oder einer spezifischen Laufzeitumgebung ausgeführt werden kann.

Java verfolgt einen zweistufigen Übersetzungsprozess. Der Programmcode in Java wird nicht zu einem ausführbaren Programm, sondern in einen Zwischencode, den sogenannten Bytecode, kompiliert. Dieser Code ist für alle Plattformen gleich und kann mithilfe des entsprechenden plattformspezifischen Interpreters auf der jeweiligen Plattform ausgeführt werden. Java-Interpreter werden auch virtuelle Maschinen (JVM) genannt.

### Interpretationsversuch

Ihr versteht nur Bahnhof? Hier, ein Versuch das Obrige an einem Beispiel zu erläutern:

Reale-Welt	Java
Stellt euch vor Ihr seit die Chef:in eines Internationalen Unternehmens.	Viele Computer mit verschiedenen Betriebssystemen.
Die Aufgaben und Befehle werden alle von Spezialisten in schweizerdeutsch erfasst.	Java ist in diesem Falle Schweizerdeutsch
Euer Unternehmen hat an jedem Standort lokale Arbeiter die nur die Landessprachen sprechen und dadurch auch nur in dieser Sprache Befehle ausführen können.	Diese können mit den Betriebssystemen verglichen werden (MacOs, Windows, Linux). Jedes Betriebssystem hat eigene Codierungen! ( <i>Deswegen funktionieren die meisten Games nur auf Windows</i> )
Da Schweizerdeutsch sehr ineffizient (besitzt viele Floskeln) und wenig verbreitet ist, wird vom einem Mitarbeiter alles Schweizerdeutsche ins Englische Übersetzt. Bei der Übersetzung ins Englische wird zudem darauf geachtet die <b>Aufgaben zu optimieren</b> .	<b>Das ist die Arbeit vom Compiler!</b> Java (Schweizerdeutsch) in Bytecode (Englisch). Es wird auch geschaut dass das Programm optimiert werden kann! ( <i>Compiler Programmierer sind die wahren Helden</i> )
An jedem Standort hat die Firma Personen angestellt die Englisch können und vom Englischen in die jeweilige Landessprache übersetzen kann. Damit die lokalen Arbeiter effizient ihre Aufgaben erledigen können.	Der Bytecode (Englisch), wird also bei jedem Betriebssystem lokal übersetzt resp. interpretiert, damit es auf dem jeweiligen Betriebssystem ausgeführt werden kann.

Reale-Welt	Java
	<b>Dies ist die Arbeit der Java Virtual Machine.</b>

Nun nach dieser Tabelle, liest bitte nochmals [von vorne](#)

### DON'T PANIK!

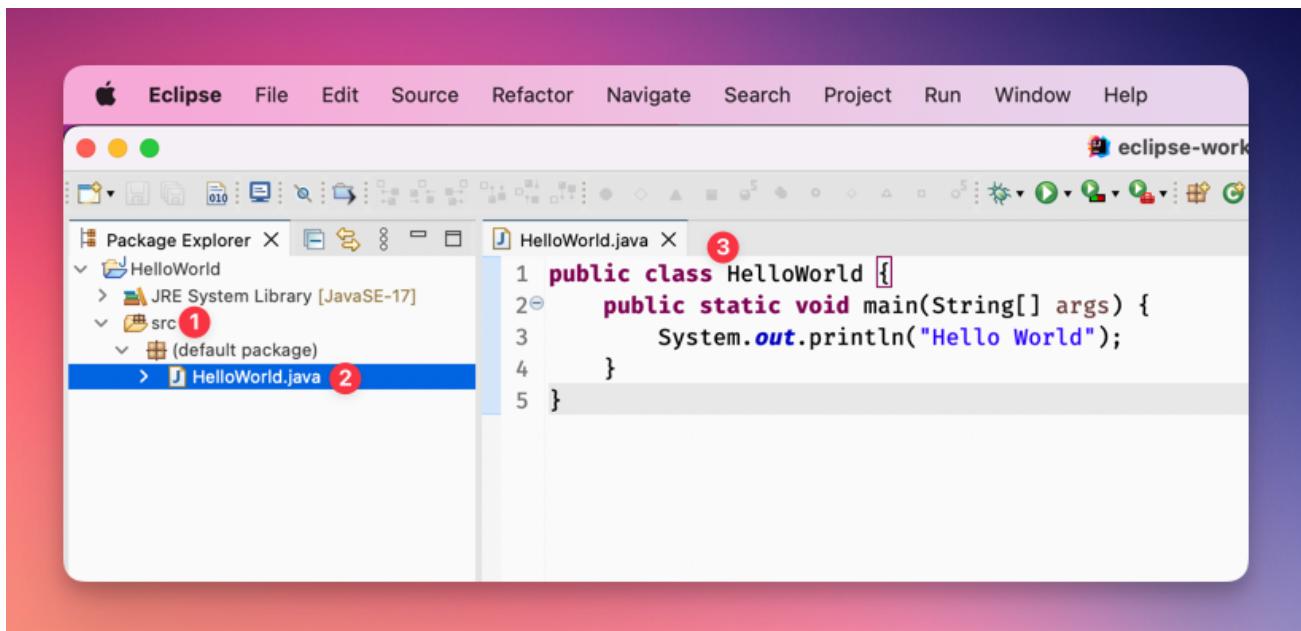
- Man muss das oberige nicht zu 100% verstanden haben um ein guter Programmierer zu werden!
- Ein guter Rennfahrer muss auch nicht ein guter Mechaniker sein ;)

## 2. Sourcecode in Form von Klassen

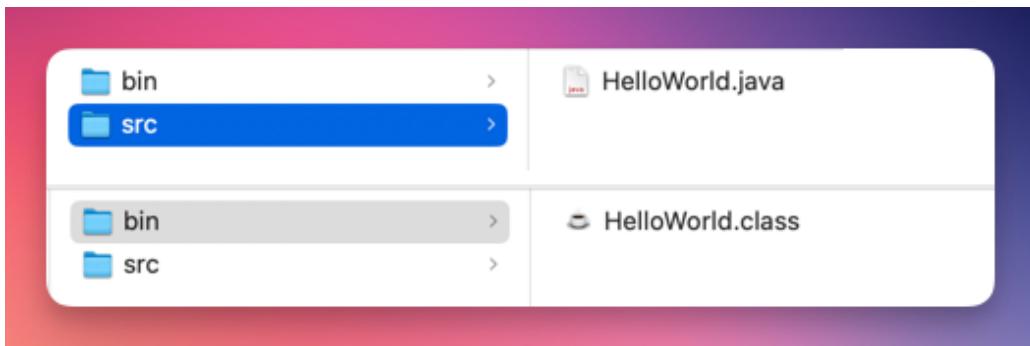
Java-Programme bestehen aus Klassen. Diese Klassen werden dann als Programme ausgeführt.

Während ein Programm resp. Klassen **(3)** geschrieben werden, werden sie in einem Verzeichnis namens **src (1)** gespeichert und haben eine **.java**-Dateiendung **(2)**.

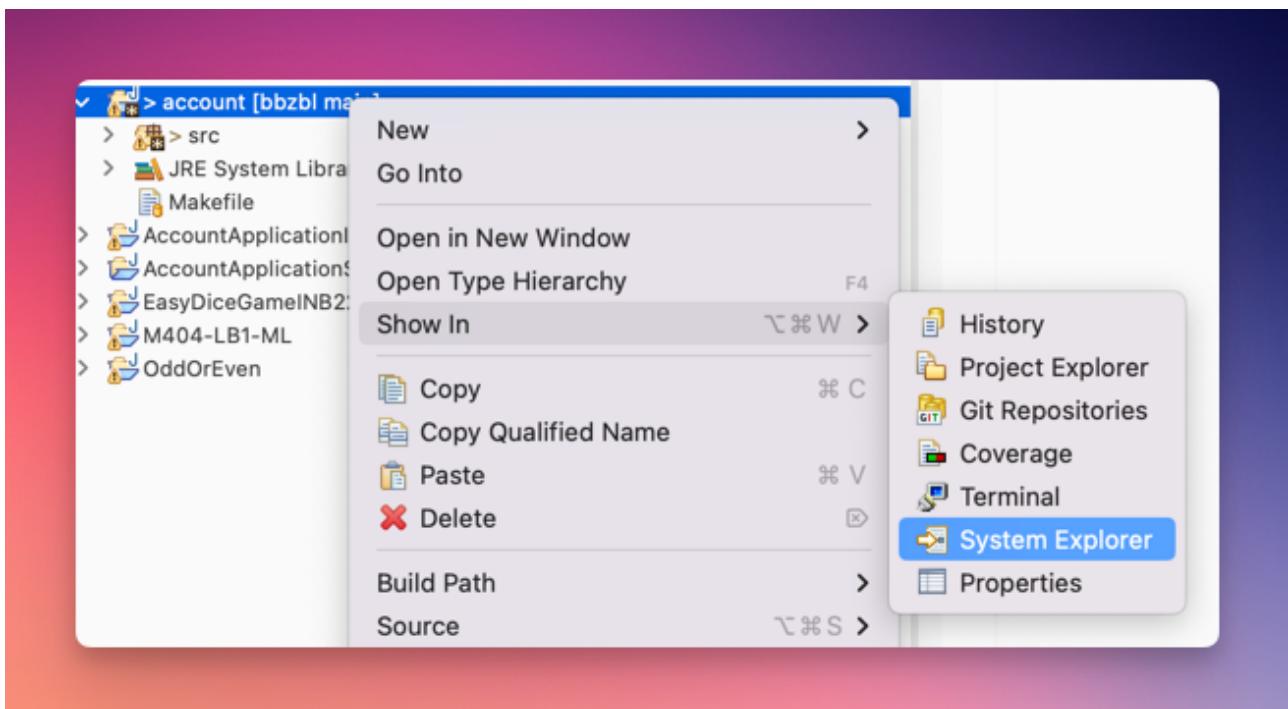
- Der Dateinamen ist immer gleich zum Klassennamen, inklusive Gross- und Kleinschreibung!
- Die Klasse **HelloWorld** befindet sich also in der Datei **HelloWorld.java**. Eclipse generiert einen Error, wenn dies nicht übereinstimmt.



Wenn die Klassen ausgeführt werden, erfolgt durch die Entwicklungsumgebung eine Kompilierung im Hintergrund (durch den Compiler `javac`), und aus den **.java**-Dateien werden **.class**-Dateien, die in einem Verzeichnis namens **bin** (für binaries) abgelegt werden. Diese Klassen können dann durch das Java-Dienstprogramm **java** ausgeführt werden.



Sie können sich diese auf der Harddisk ansehen, indem Sie z.B. auf dem `src`-Ordner die rechte Maustaste betätigen und dann *Show in => System Explorer* wählen.



In der `.java`-Datei kann zuerst das `package` angegeben werden, in welchem sich die aktuelle Klasse befindet.

- Wird kein `package` angegeben, wie beim `HelloWorld` programm der Fall, befindet sich die Datei direkt unter dem `src` Ordner, `src/HelloWorld.java`.
- Ist ein `package` angegeben, befindet sich die Klasse in einem Unterordner der gleich heisst wie das `package`. In folgenden Fall unter `src/helloworld/HelloWorld.java`

```
package helloworld;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Dann folgen optional `import`-Anweisungen, die andere Java- und Programmteile im eigenen Programm nutzbar machen.

```
package helloworld;  
import some.other.package;  
  
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

Die eigentliche Klasse beginnt mit der Anweisung `public class HelloWorld`, was aussagt, dass es sich um eine öffentliche Klasse mit dem Namen HelloWorld handelt. Nach dieser folgt eine öffnende geschweifte Klammer, die mit der letzten schliessenden geschweiften Klammer zusammen den Klassenkörper bildet.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

Dann folgt die main-Methode `public static void main(String[] args)`. Diese Zeile wird auch als Methodensignatur bezeichnet. Im Detail wird die Methodensignatur später angesehen. Die `main`-Methode ist eine ganz besondere, **sie dient immer als Startort des Programms**.

- ⚒ Sie ist sozusagen die Pforte zum Schloss! ⚒

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

Das nächste paar geschweifter Klammern bildet den Methodenkörper. Darin steht das, was die eigentliche Leistung des Programms ausmacht. Über die Java-Klasse System erfolgt die Ausgabe auf die Console (out), wo eine Zeile ausgegeben wird (`println`).

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

## Aufgabe: Visualisierung

Erstellen Sie eine eigene Visualisierung, welche für Ihr HelloWorld Programm aufzeigt, wie der Sourcecode zu einem ausführbaren Programm überführt wird, welcher von einem Interpreter ausgeführt werden kann.

# Analyse des bisherigen Programms

Nun haben wir schon ein ausführbares Programm. Hier wird nun Analysiert wie es genau Aufgebaut ist. Wie ist die Struktur?, Wo liegen die Dateien?

## Von Klammerpaaren und Blöcken

Klammerpaare dienen in Java zum Gruppieren von zusammengehörigem Code. Diese Gruppen werden auch Blöcke genannt. Als Eselsbrücke könnt Ihr euch einen Schrank vorstellen. Dieser ist in Schubladen unterteilt, wobei jede Schublade eine gewisse Kategorie von Gegenstände beherbergt. Z.B. Socken, Unterhosen, T-Shirts, usw.

### STRUKTUR IST DIE HALBE MIETE, WENN NICHT MEHR

Natürlich kann man einen Schrank wild füllen, nur ist es dann nicht mehr so einfach etwas zu finden. Und genau so ist es beim Programmieren!

Achtet darauf Programme gut zu Strukturieren. Dazu dienen Code-Blöcke.

Es ist nun wichtig, dass Sie bei allen Klammern immer daran denken, dass diese **paarweise eingesetzt werden**. Sie müssen also geschlossen werden. Nur so können Sie Code Gruppieren. Wird eine Klammer nicht geschlossen, gibt es keine klare Abgrenzung (ähnlich wie bei normalem Text... Da fehlt wohl eine Klammer!)

In Java werden folgende Klammentypen verwendet:

- `{ }` Geschweifte Klammern für **Codeblock**
- `( )` Runde Klammern für **Methoden/Funktions Signatur** (meistens gefolgt von einem Codeblock)
- `[ ]` Eckige Klammern für Auflistungen, auch **Arrays** genannt

Als eine Besonderheit kennzeichnet dieses Klammerpaar `{ }` sogenannte Blöcke, die man auch als Programmabschnitte bezeichnen kann.

### INFO

- Nach einem Codeblock `}` folgt **kein** Semikolon
- Nach einer Schliessenden normalen Klammer `)` folgt in der Regel eine geschweifte Klammer `{`
  - Ausnahmen sind in dem Modul nicht relevant.
- Jeder Befehl endet mit einem Semikolon `;`
  - Nach jedem Semikolon sollte eine neue Zeile beginnen, muss aber nicht!
  - Dies nicht zu machen ist **schlechter Stiel und gibt Abzug!**

## 1. Blöcke finden

Wenden Sie nun das im oberen Abschnitt vorgestellte Konzept der Blöcke an. Bisher haben wir die Klassen und `main`-Methode von Java kennen gelernt.

- Tragen Sie im folgenden HelloWorld-Programm die **vorhandenen Blöcke** ein.
- Zeichnen Sie ein, welche anderen Klammern Paare bilden.

HelloWorld.java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```



TIP

- Sie können in Eclipse mit `// Kommentaren` die Blöcke benennen
- Oder Sie können den Block auch mit dem Snipping-Tool "**screenshoten**" und in einem Programm Ihrer Wahl, mit dem **Markertool** durchführen. (Paint / ...)
- Order, wenn Sie diese Seite **ausdrucken**, können Sie die Aufgabe mit einem konventionellen Marker von Hand durchführen.

## Dateienstruktur

Wenn Ihr in Eclipse Dateien erstellt, werden automatisch auch Dateien auf dem Dateisystem angelegt. Die Struktur ist meistens gleich zum Project-Explorer von Eclipse, aber nicht immer.

## HelloWorld.java Datei finden

Und wo liegt nun die `HelloWorld.java` Datei?

- Klicken Sie in Eclipse mit der rechten Maustaste auf die Klasse `HelloWorld` und wählen Sie «Show in», dann «System Explorer».
- Anschliessende sehen Sie die Klasse im Windows Explorer rsp. Finder.



ACHTUNG!

Der **Klassenname und Dateiname müssen immer identisch sein**. Probieren Sie es aus. Nun sollte Eclipse einen Error anzeigen.

## Das `bin`-Verzeichnis und `class`-Dateien suchen

Suchen Sie von hier aus auch das `bin`-Verzeichnis. Sie sollten darin die `class`-Dateien sehen. Diese werden jeweils bei jeglicher Veränderung durch Eclipse neu an dieser Stelle erzeugt.

# Code-Konventionen

Wenn programmiert wird, kommen meist eine ganze Reihe an Konventionen zur Anwendung. Je nach Betrieb können die sich erheblich unterscheiden. Wir stellen hier einige wichtige Konventionen vor.

## Keine Umlaute im Code

Programmcode ist wenn möglich in englisch gehalten. Umlaute können auf verschiedenen Systeme zu Fehlern führen und sind nicht international verständlich. Deswegen sollen Umlaute beim Programmieren vermieden werden!

## Klassennamen

Jede **Klasse**

- beginnt mit einem **Grossbuchstaben**
- hat einen **AusdrucksstarkenNamen** in  **UpperCamelCase**

```
public class MeinTollerKlasseName {  
}
```

## Methodennamen

Jede **Methode**

- beginnt mit einem **Kleinbuchstaben**
- hat einen **ausdrucksstarkenNamen** in  **lowerCamelCase**

```
public class MeinTollerKlasseName {  
    public void meinTollerMethodenName() {  
    }  
}
```

## Codeblöcke einrücken

- Blöcke  werden **eingerückt**
  -  Ctrl-Shift-F
  -  Command-Shift-F

```
public class MeinTollerKlasseName {  
    // Dieser Block ist eingerückt  
    public void meinTollerMethodenName() {  
        }  
    }
```

## UTF-8 als Standard-Encoding

Wenn UTF-8 verwendet wird, sollten theoretisch auch Umlaute auf allen Systemen funktionieren. Diese werden jedoch trotzdem vermieden ;) Sie gelten als schlechter Stiel und geben abzug!

- Standard-Encoding `UTF-8`
- `Preferences > General > Workspace` -> `UTF-8`

## Kommentare

Es gibt gute Gründe für Kommentare:

- eine **öffentliche Methode** für JavaDoc kurz Beschreiben
- erläutern **warum** eine Entscheidung getroffen wurde
- "TODO-Kommentare" für Infos was man in Zukunft verbessern sollte

## Einzeiliger Kommentar

```
// Ich bin ein einzeiliger Kommentar
```

- Kommentare beginnen mit Zwei Fronslashes `//` und gelten für die ganze Zeile danach
- Man kann also nach einem `//` kein ausführbaren code mehr schreiben

## Mehrzeiliger Kommentar

```
/*  
Ich  
bin  
ein  
Mehrzeiliger  
Kommentar  
*/
```

- Mehrzeilige Kommentare beginnen mit `/*` und enden mit `*/`
- Jeglicher Code dazwischen wird nie ausgeführt!

## JavaDoc Kommentare

```
/*
 * Ein JavaDoc Klassen Kommentar wird angezeigt,
 * wenn die Methode von Eclipse vorgeschlagen wird.
 * Er steht immer direkt vor der Klasse.
 *
 * @author HerrLehrer
 * @version 1.0.0
 */
public class MeinTollerKlasseName {

    /*
     * Die diese Methode wird ganz tolle Sachen machen
     * die man dann irgend wann auch verwenden kann.
     *
     * @param einParameter Parameter können so beschrieben werden
     * @return es wird einfach nur der "einParameter" zurückgegeben
     */
    public String meinTollerMethodenName(String einParameter) {
        // TODO: Hier wird noch was tolles programmiert
        return einParameter;
    }

}
```

- JavaDoc Methodenkommentare beschreiben Methoden so, dass direkt eine Dokumentation daraus generiert werden kann.
- [🔗 JavaDoc Beschreibung auf Wikipedia](#)

### 🔥 ES GIBT ABER AUCH SCHLECHTE KOMMENTARE!

Kommentare **sollten nicht beschreiben was der Programmcode im Detail macht!** Das steht im Code. Wenn dafür Kommentare nötig sind, sollte der Code überdacht werden:

- Kleinere Methoden wo der Name bereits beschreibt was gemacht wird
- Komplexe Abhängigkeiten entkoppeln / auseinandernehmen

⌚ Kommentare **veralten schnell!** Nichts ist schlimmer als ein Kommentar der nicht mehr stimmt!

## 📝 Auftrag: Kommentieren Sie Ihr «HelloWorld»

Beschreiben Sie mit Kommentaren Ihr «HelloWorld»-Programm, so dass Sie sich später wieder an alle Schritte der Erstellung erinnern.

- Diese Kommentare sind zwar nicht sinnvoll für die Softwareentwicklung -> siehe roter Kasten oben.

- Hier sind sie jedoch sinnvoll für sie selbst zur Dokumentation!

## 2a - Das EVA-Prinzip

Ohne Benutzereingaben machen viele Programme keinen Sinn. Sie lernen daher hier das EVA-Prinzip kennen, und wie Sie es mit einer Bibliothek in Ihrem Projekt nutzen, um Eingaben von Benutzern einzulesen. Sie kennen die vorgestellten Datentypen und können diese in Programmen nutzen.

### ⌚ Ziele

- Sie können das EVA-Prinzip erklären.
- Sie können eine Bibliothek in Ihr Eclipse-Projekt einbinden. Sie können die Bibliothek dazu verwenden, um Benutzereingaben einzulesen.

## Das EVA Prinzip

Grundsätzlich befolgen alle Programme das EVA Prinzip. Selbst Smartphones, die auf Wisch-Gesten reagieren, folgen diesem Prinzip, einfach viel schneller. Mit diesen Aufträgen lernen Sie, wie Sie Daten in Java-Programme eingeben können.

### ⓘ EVA-PRINZIP

**Eingabe -> Verarbeiten -> Ausgabe**

### Eingabe

Hier ein paar Eingabearten, natürlich können diese beliebig erweitert werden.

- Formular einer Applikation
- Mausbewegung
- Joystick bewegung
- Gamepad
- Wischgeste auf einem Smartphone
- Spracheingabe
- Kamera
- ...

### Verarbeiten

Diese Eingaben werden vom Programm verarbeitet. Je nach Eingabe und Programmierung wird ein Befehl ausgeführt. z.B. Wenn Ihr die Maus bewegt, weiss das Betriebssystem, dass es den Cursor auf dem Bildschirm bewegen soll.

Wenn man nicht gerade ein Spiel oder Betriebssystem programmiert handelt es sich jedoch zu 90% um das Auswerten von Formularen jeglicher Form.

## Ausgabe

Durch das Verarbeiten erstellt das Programm ein entsprechendes **Resultat!** Dieses Resultat wird dann wieder zurückgegeben. Beim Mausbewegen bedeutet dass, dass der Mauszeiger auf dem Bildschirm sich bewegt.

Wenn man ein Formular auswertet ist die Ausgabe meistens eine Bestätigung, dass die Formulardaten in einer Datenbank gespeichert wurden.

# Bibliothek MyTools einbinden

## 1. Neues Projekt und Klasse erstellen

- Erstellen Sie in Ihrem persönlichen Workspace ein neues Projekt namens "EVA".
- Legen Sie darin die Klasse `EinUndAusgabe` an.

## Eclipse für die Bibliothek einrichten

1. Legen Sie im Projekt ein Verzeichnis «lib» an
  - Rechtsklick auf das **Projekt** -> **New** -> **Folder**
2. Laden Sie die Bibliotheks-Datei `MyTools.jar` herunter.
3. Kopieren Sie die Datei in das erstellte lib-Verzeichnis
  - Falls Sie die Datei auf Filesystem Ebene kopiert haben, muss allenfalls die **Ansicht in Eclipse aktualisiert** werden
  - Rechtsklick auf das **Projekt** -> **Refresh**
4. Klicken Sie nochmals mit der rechten Maustaste auf den Projekteintrag im Package Explorer.
  - Wählen Sie diesmal «Build Path», dann «Configure Build Path».
  - Wählen Sie das Register Libraries, dann den Eintrag Classpath.
  - Klicken Sie auf «Add JARs...». Öffnen Sie das Projekt und das lib-Verzeichnis, und wählen MyTools.jar.
  - Klicken Sie auf «Apply and Close».

### ⓘ MYTOOLS.JAR DOWNLOAD

- Die Datei `MyTools.jar`  hier downloaden!

## Die Klasse StdInput

Die Bibliothek beinhaltet die Klasse `StdInput`. Mit dieser können verschiedene Datentypen vom Terminal aus eingelesen werden.

Datentyp	Methode
String	<code>StdInput.readString()</code>
boolean	<code>StdInput.readBoolean()</code>
char	<code>StdInput.readChar()</code>
double	<code>StdInput.readDouble()</code>

Datentyp	Methode
int	<code>StdInput.readInt()</code>

 **TIP**

- Falls Sie also z.B. eine Ganzzahl einlesen wollen wäre dies: `int number = StdInput.readInt();`
- Falls Sie ein String einlesen wollen wäre dies: `String textEntry = StdInput.readString();`

# Text mit StdInput einlesen

- Informieren Sie den Benutzer, dass er nun seinen Namen eintippen soll.
- Legen Sie eine String-Variable an, um die Antwort zu speichern.
- Die Funktion, um Daten eingeben zu können, ist in der Klasse mit dem Namen `StdInput` enthalten.

## AUTOMATISCHE VERVOLLSTÄNDIGUNG

Fangen Sie an zu schreiben und nach «Std» drücken Sie **CTRL+LEERTASTE**, worauf automatisch die Klasse vervollständigt wird und der korrekte Import für die Klasse hinzugefügt wird.

EinUndAusgabe.java

```
import mytools.StdInput;

public class EinUndAusgabe {
    public static void main (String[] args) {
        System.out.println("Please type your name:");
        String name = StdInput // hier fehlt noch was!
    }
}
```

Nun wurde Ihre Klasse gerade zu oberst mit folgender Zeile ergänzt, die dafür sorgt, dass `StdInput` in Ihrer Klasse genutzt werden kann: `import mytools.StdInput;`

Geben Sie direkt hinter `StdInput` **einen Punkt** ein. Nun erscheint folgender Dialog, über welchen Sie die **Methoden der Klasse StdInput aufrufen** können:

```
import mytools.StdInput;

public class EinUndAusgabe {

    public static void main(String[] args) {
        System.out.println("Please type your name:");
        String name = StdInput.
    }
}
```

A screenshot of an IDE interface showing code completion for the variable 'StdInput'. A red arrow points from the period character in 'StdInput.' to a dropdown menu that lists several methods: 'readString()' (highlighted in blue), 'readString(String var0)', 'class', 'readBoolean()', 'readChar()', 'readDouble()', 'readDouble(String var0)', 'readInt()', and 'readInt(String var0)'. The menu has a light gray background with blue outlines around the listed items.

## ① METHODENZUGRIFF ÜBER PUNKT

Auf statische **Methoden** einer statischen Klasse wird immer **über einen Punkt** zugegriffen.

```
KlassenName.methodName([optionaleParemeter]);  
//           ^  
// wichtiger Punkt!
```

Da wir einen String-Wert einlesen wollen, wählen gleich den obersten Eintrag `readString()` und schliessen die Zeile mit einem Semikolon `;` ab. Als nächstes geben wir den eingegebenen Namen aus, wie in der Programmübersicht gezeigt:

EinUndAusgabe.java

```
import mytools.StdInput;  
  
public class EinUndAusgabe {  
    public static void main (String[] args) {  
        System.out.println("Please type your name:");  
        String name = StdInput.readString();  
        System.out.println("Your name is " + name);  
    }  
}
```

Führen Sie das Programm aus. Funktioniert es? Begrüssst Sie Ihr Programm mit dem eingegebenen Namen?

## 💡 STRINGS ZUSAMMENSETZEN

- Mit einem `+` können Strings Zusammensetzen werden
- Der erste String sollte mit einem Leerzeichen Enden, wieso?

Strings zusammensetzen

```
String name = "Herr Lehrer";  
System.out.println("Your name is " + name);  
//           ^ Leerzeichen!  
// Your name is Herr Lehrer  
  
System.out.println("Your name is" + name);  
// Your name isHerr Lehrer
```

## Weitere Datentypen einlesen

Die Bibliothek unterstützt verschiedene Datentypen, wie aus der Tabelle der Methoden ersichtlich ist. Anstelle von `readString()` rufen Sie demnach lediglich eine andere `read...-Methode` auf.

Datentyp	Methode
int	StdInput.readInt()
char	StdInput.readChar()
String	StdInput.readString()
double	StdInput.readDouble()
boolean	StdInput.readBoolean()

### ① DATENTYPEN EINLESEN

```
int      ganzZahl      = StdInput.readInt();
char    character      = StdInput.readChar();
String   text          = StdInput.readString();
double  gleitKommaZahl = StdInput.readDouble();
boolean bool          = StdInput.readBoolean();
```

# Zusatz - Scanner

## Die Klasse Scanner

Java beinhaltet bereits viele vorgefertigte Tools um gängige Aufgaben zu lösen. Eine davon ist das Einlesen und Verarbeiten von Benutzereingaben. Wir werden in diesem Modul nur Programme schreiben, welche mit dem Benutzer über die Konsole interagieren. Diese nennt man auch Konsolen-Applikationen.

Dafür bietet Java die Klasse `java.util.Scanner` an. Sie "scannt" sozusagen die Eingabe von Benutzer und gibt diese zurück.

### MUSTERLÖSUNGEN MIT MYTOOLS.STDINPUT

Die Klasse `Scanner` wird hier für wissbegierige vorgestellt, da Sie zum Standard-Repetoir von Java gehört. Das Package `mytools` wird ausserhalb der BBZBL nicht verwendet.

Die BBZBL verwendet für dieses Modul die Klasse `mytools.StdInput`. Ihr dürft alle Aufgaben auch mit dem `Scanner` lösen. Dafür gibt es jedoch keine Musterlösungen.

## Verwendung

Um den Scanner zu verwenden, müssen **3 Punkte** beachtet werden:

Verwenden vom Scanner

```
import java.util.Scanner;

class MyClassWithScanner { // INFO: Der Klassenname ist beliebig!

    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        // hier kann nun der scanner verwendet werden
        String eingabe = scanner.next(); // next() gibt immer einen String zurück
        System.out.println("Ihre eingabe war: " + eingabe);
    }
}
```

1. Er muss importiert werden mit `import java.util.Scanner;`
2. Er muss initialisiert werden mit `private static Scanner scanner = new Scanner(System.in);`
3. Er muss verwendet werden mit `scanner.next...`

## Weitere Methoden

Methode	Datentyp
next()	String
nextLine()	String bis Enter
nextInt()	int
nextLong()	long
nextFloat()	float
nextDouble()	double

## Aufgabe: Text mit Scanner einlesen

- Informieren Sie den Benutzer, dass er nun seinen Namen eintippen soll.
- Legen Sie eine `String`-Variable an, um die Antwort zu speichern.
- Die Funktion, um Daten eingeben zu können, ist in der Klasse mit dem Namen `Scanner` enthalten.

## Schritt für Schritt Anleitung

1. Kopieren Sie den folgenden Code in eine Datei namens `EinUndAusgabe.java`

`EinUndAusgabe.java`

```
1 public class EinUndAusgabe {  
2  
3     private static Scanner scanner = new Sc // hier fehlt noch was  
4  
5     public static void main (String[] args) {  
6         System.out.println("Please type your name:");  
7         String name = scanner // hier fehlt noch was!  
8     }  
9 }
```

2. **Zeile 3:** Vervollständigen Sie `new Sc` mit `new Scan` und drücken Sie danach **CTRL+LEERTASTE**.

- Dadurch wird eine automatische Vervollständigung von Eclipse aktiviert welche ebenfalls den korrekte Import für die Klasse hinzufügt.
- Nun wurde Ihre Klasse gerade zu oberst mit folgender Zeile ergänzt, die dafür sorgt, dass `Scanner` in Ihrer Klasse genutzt werden kann: `import java.util.Scanner;`

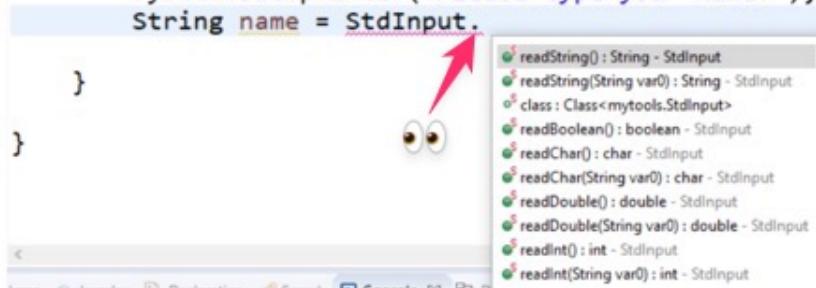
3. **Zeile 7/8:** Geben Sie direkt hinter `scanner` **einen Punkt** ein.

- Nun erscheint folgender Dialog, über welchen Sie die **Methoden der Klasse scanner aufrufen** können:

```
import mytools.StdInput;

public class EinUndAusgabe {

    public static void main(String[] args) {
        System.out.println("Please type your name:");
        String name = StdInput.readString();
    }
}
```



- Da wir einen String-Wert einlesen wollen, wählen gleich den obersten Eintrag `next()` und schliessen die Zeile mit einem Semikolon `;` ab.

#### 4. Als nächstes geben wir den eingegebenen Namen aus

- Dafür verwenden wir `System.out.println()`. Dies ist der Standardweg in Java text auf die Konsole auszugeben.

##### EinUndAusgabe.java

```
import java.util.Scanner;

public class EinUndAusgabe {

    private static Scanner scanner = new Scanner(System.in);

    public static void main (String[] args) {
        System.out.println("Please type your name:");
        String name = scanner.next();
        System.out.println("Your name is " + name);
    }
}
```

#### 5. Führen Sie das Programm aus. Funktioniert es? Begrüssst Sie Ihr Programm mit dem eingegebenen Namen?

## 💡 METHODENZUGRIFF ÜBER PUNKT

Auf statische **Methoden** einer statischen Klasse wird immer **über einen Punkt** zugegriffen.

```
KlassenName.methodName([optionaleParameter]);  
//           ^  
// wichtiger Punkt!
```

## 💡 STRINGS ZUSAMMENSETZEN

- Mit einem **+** können Strings Zusammensetzen werden
- Der erste String sollte mit einem Leerzeichen Enden, wieso?

Strings zusammensetzen

```
String name = "Herr Lehrer";  
System.out.println("Your name is " + name);  
//                           ^ Leerzeichen!  
//                         Your name is Herr Lehrer  
  
System.out.println("Your name is" + name);  
//                         Your name isHerr Lehrer
```

## Weitere Datentypen einlesen

Die Bibliothek unterstützt verschiedene Datentypen, wie aus der Tabelle der Methoden ersichtlich ist.  
Anstelle von `readString()` rufen Sie demnach lediglich eine andere `read...-Methode` auf.

Datentypen mit Scanner einlesen

```
int      ganzzahl      = scanner.nextInt();  
char     character      = scanner.next().charAt(0);  
String   text          = scanner.next();  
double   gleitKommaZahl = scanner.nextDouble();  
boolean  bool          = scanner.nextBoolean();
```

## 2b - Variablen & Datentypen

Wenn Programme Daten bearbeiten, so sind diese in **Variablen (änderbar)** und **Konstanten (nicht änderbar)** gespeichert. Die Variablen und Konstanten besitzen einen Namen sowie einen für die zu speichernden Daten passenden **Datentyp**.

### ⌚ Ziele

- Sie kennen die primitiven Datentypen und den höheren Datentyp String
- Sie können diese als Variablen in Programmen nutzen.

### 💻 Präsentation

[Open in Browser](#) | [download PDF](#)

## Deklaration und Initialisierung von Variablen

Das generelle Muster (*die Syntax*), um eine Variable zu deklarieren (*dem Programm bekannt machen*), hat zwei Varianten. Die Variable kann "nur" deklariert oder aber sogleich mit einem entsprechenden Wert initialisiert werden.

1. bei der Ersten Variante wird die Variable durch Angabe des **Datentyps gefolgt vom Variablenamen, deklariert**

```
char favoriteSign; // Variante 1 Deklaration: Variable ohne Wert
```

2. die Zweiten Variante **ergänzt die Erste Variante** mit einem Wert welcher durch den **Zuweisungsoperator** `=` zugewiesen wird. Damit wird die Variable deklariert und direkt initialisiert.

```
char favoriteSign = 'b'; // Variante 2 Deklaration + Initialisierung:
```

#### ⓘ NOTE

Wir verwenden in den Beispielen der Datentyp `char`, das Muster ist jedoch für alle Datentypen gleich. Die Ausnahme ist der Wert, der pro Datentyp anders ist.

## Deklaration von Konstanten

Sollen Variablen **nicht änderbar** sein, dann werden diese als Konstanten bezeichnet, was durch das Schlüsselwort `final` erfolgt. Der Variablenname wird gemäss Konvention in **GROSSBUCHSTABEN** geschrieben: Zudem sind die Konstanten `static`, werden also nur einmal angelegt. Mit den Schlüsselwörtern `private` und `public` lässt sich die Nutzung steuern.

```
static final float MWST = 7.7;  
// ^^^^^ GROSSBUCHSTABEN als Konvention für Konstante  
//       ^^^^^ final definiert eine Konstante  
// ^^^^^^ static definiert dass ein Wert fest ist.
```

## Primitive Datentypen

In Java gibt es eine beachtliche Anzahl an Datentypen gemäss folgender Tabelle. Vorerst nutzen wir die *primitiven* Datentypen. Diese sind erkennbar, indem der **Datentyp kleingeschrieben** ist.

Datentyp	Grösse	Beschreibung	Spezifika
boolean	1 bit	Speichert <code>true</code> ( <i>wahr</i> ) oder <code>false</code> ( <i>falsch</i> ) Werte	
byte	1 byte	Speichert ganze Zahlen von <code>-128</code> bis <code>127</code>	
short	2 bytes	Speichert ganze Zahlen von <code>-32'768</code> bis <code>32'767</code>	
char	2 bytes	Speichert ein einzelnes Zeichen oder <b>ASCII</b>	' ''
int	4 bytes	Speichert ganze Zahlen <code>-2'147'483'648</code> bis <code>2'147'483'647</code>	
float	4 bytes	Speichert Gleitkommazahlen von <code>6</code> bis <code>7</code> Dezimalstellen	f
long	8 bytes	Speichert ganze Zahlen von <code>-9'223'372'036'854'775'8081</code> bis <code>9'223'372'036'854'775'8071</code>	l
double	8 bytes	Speichert Gleitkommazahlen von <code>15</code> Dezimalstellen	d

## Initialisierung

Beispiele, wie Variablen initialisiert werden können. Die Leerzeichen dienen nur der Übersichtlichkeit.

## Deklarierung von Variablen

```
// Datentyp      Variablenname   Semikolon
int            number          ;
char           sign           ;
...
```

Die Initialisierung der Werte verwendet spezifische Zeichen für die verschiedenen Datentypen. So endet ein `float`-Wert immer mit `f` oder `long` mit `l`. Die Spezifika pro Datentyp finden Sie in der Tabelle oberhalb unter "Spezifika".

## Deklarierung & Initialisierung von Variablen

```
// Datentyp      Variablenname   Zuweiseoperator   Wert   Semikolon
int            number          =             5       ;
char           sign           =             'c'     ;
//                                     ^ ^ spezifisch für char sind ('')
long           longNumber     =             1231    ;
//                                     ^ spezifisch für long (l)
...
```

### ⚠ SPEICHERGRÖSSE

Früher hatte man wehnig Speicher zur Verfügung und hat immer abgewägt, welchen Datentyp verwendet werden soll. Heute nimmt man für gewöhnlich einfach den Grössten.

- Dies verkleinert die Liste auf folgende vier: `boolean`, `long`, `double` und `char`.
- Für den Test müsst ihr trotzdem alle kennen 😊

## Höhere Datentypen

Höhere Datentypen bauen auf den *primitiven* Datentypen auf. Es handelt sich bei diesen auch um Objekte.

### ⚠ ABGRENZUNG

- Was genau Objekte sind müsst Ihr noch nicht verstehen.
- Ihr solltet in dem Modul den Datentyp `String` anwenden können.

## Der Datentyp `String`

Der Datentyp `String` **dient zur Speicherung von Zeichenfolgen**, also allgemeiner Text. Der Text muss immer zwischen zwei "**doppelten Anführungszeichen**" gestellt werden.

Datentyp	Grösse	Beschreibung	Spezifika
String	2 byte pro Zeichen	Speichert beliebigen Text	" "

## Strings initialisieren

```
// Datentyp      Variablenname   Zuweisungsoperator   Wert      Semikolon
String        zeichenFolge    =           "abc"    ;
//                                         ^          ^
//                                         "doppelten Anführungszeichen"
```

## Strings zusammensetzen

Mehrere Variablen vom Datentyp `String` können durch ein Plus-Zeichen `+` zusammengesetzt werden. Dabei sollte man darauf achten, dass der erste Text mit einem Leerzeichen enden soll. Wieso, sieht ihr im Beispiel:

### Mit `+` Strings zusammensetzen

```
String name = "Mr Robot";
System.out.println("Your name is " + name);
//                           ^
//                           Leerzeichen!
//                         Your name is Mr Robot

System.out.println("Your name is" + name);
//                         Your name isMr Robot
```

### ① HÖHERE DATENTYPEN SIND GROSSGESCHRIEBEN

- `String` ist Gross geschrieben, da es sich um einen *höheren* Datentyp handelt.
- Ein `String` baut auf dem *primitiven* Datentyp `char` auf (*daher höher*)
  - Evt. Hilft die Analogie von "Atome (primitiv)" zu "Moleküle (höher)".
- *Höhere* Datentypen sind auch Objekte.
  - Was Objekte genau sind, erfahrt ihr im Folgemodul 404 und ist noch nicht relevant!

## Strings mit Zahlen zusammensetzen

Strings können auch mit allen *primitiven* Datentypen, also auch mit numerischen Werten, durch das Plus-Zeichen `+` zu einer Zeichenfolge kombiniert werden. Der *primitive* Datentyp wird dadurch automatisch zu einem String!

String mit Zahlen kombinieren

```
System.out.println("Ihre Geschwindigkeit lautet " + 21);  
//                     Ihre Geschwindigkeit lautet 21
```

### EINE ZAHL IN EINEN STRING UMWANDELN

Werden Zahlen mit einem **leeren String ""** konkatiniert, wird die Zahl alleine in einen String umgewandelt. Dies ist ein gängiger Java "Hack".

Zahl in String umwandeln

```
String zahl = "" + 21;  
// zahl ist nun "21"
```

## Der Datentyp `LocalDate` für Datumswerte

Mit der Klasse `java.time.LocalDate` lassen sich Datumswerte speichern resp. das aktuelle Datum erzeugen, wie nachfolgendes Beispiel zeigt:

Momentane Zeit (jetzt, now) ausgeben

```
LocalDate d = LocalDate.now();  
System.out.println(d);
```

## Datentypen konvertieren (Casting)

Manchmal muss man oder will den aktuellen Datentypen ändern, und Daten in einen anderen Datentyp konvertieren. In manchen Fällen ist das unproblematisch, wie hier, da Java ermitteln kann, dass kein Genauigkeitsverlust auftritt:

```
int smallNumber = 123;  
long convertedSmallNumber = smallNumber;
```

In anderen Fällen wird die Entwicklungsumgebung hingegen eine Fehlermeldung anzeigen, wie hier:

```
long bigNumber = 1112223334445566L;  
int convertedBigNumber = bigNumber;
```

In wieder anderen Fällen, werden Sie Berechnungen programmieren, deren Ergebnis ganzzahlig sein muss. Angenommen ein strenges Notensystem liesse nur ganze, abgerundete Modulnoten zu, während diese den Semestertests mit Zehntelsnoten berechnet werden, dann könnte Eclipse darauf hinweisen, dass folgende Zuweisung ungültig ist.

```
int grade = (3 + 4 + 5.5) / 3;
```

Die obige Fehlermeldung kann nun beim Programmieren übersteuert werden, indem ein Casting erzwungen wird, dazu wird der Datentyp in Klammern dazwischen geschrieben:

```
int grade = (int) ((3 + 4 + 5.5) / 3);
```

Casting kann sichtbar machen, dass alle Buchstaben am Computer durch Zahlen repräsentiert werden (ASCII-Tabelle u.ä.).

```
int a = 65;  
System.out.println((char)a);
```

Je nach Reihenfolge und Stelle des Castings können unterschiedliche Ergebnisse berechnet werden, wie das folgende Beispiel zeigt (Ergebnis einmal 70, einmal 60: aber warum?):

```
int i = (int) (20.0 * 3.5);  
System.out.println("i " + i);  
int j = (int) 20.0 * (int) 3.5;  
System.out.println("j " + j);
```



#### IN ECLIPSE AUSFÜHREN!

Führt die Code-Blöcke selbst in Eclipse aus und sieht was genau für Ergebnisse oder Fehler angezeigt werden!

## Rechnen und Operatoren

### Arithmetische $+$ , $-$ , $/$ , $*$ , $\%$

Arithmetische Operatoren kennt Ihr bereits von der Mathematik. Damit lassen sich die gängigen Mathematischen Operationen durchführen. Neu ist einzig der Rest Operator  $\%$ . Dieser dividiert eine Zahl und gibt den Rest zurück.

## + - \* % Arithmetische Operatoren

```
int result;  
int number = 9;  
int anotherNumber = 3;  
  
result = number + anotherNumber; // Addition  
result = number - anotherNumber; // Subtraktion  
result = number / anotherNumber; // Division  
result = number * anotherNumber; // Multiplikation  
result = number % anotherNumber; // Rest der Division
```

### GERADE/UNGERADE BERECHNEN MIT %

Der Rest-Operator `%` gibt bei einer division immer den Rest zurück. Wenn man nun eine Division durch 2 durchführt lässt sich herausfinden ob eine Zahl gerade oder ungerade ist.

```
9 % 2 // ergibt 4 * 2 Rest 1 also ungerade  
10 % 2 // ergibt 5 * 2 Rest 0 also gerade  
  
public boolean even(int number) {  
    return number % 2 == 0;  
}
```

## Verkürzte arithmetische Operation mit sich selbst `+=`, `-=`, `*=`, `/=`

Oft möchte man den Wert einer Variablen direkt verändern. Das Resultat also nicht in eine neue Variable, sonder in sich selber speichern. Gegeben ist z.B. die Variable `number` vom Typ `int` mit dem Initialwert `3`.

```
int number = 3;
```

Möchte man dieser Variable `4` hinzuzaddieren geht das folgendermassen:

```
number = number + 4; // Addition und Zuweisung zu sich selbst
```

Da dies sehr oft vorkommt ist in allen gängigen Programmiersprachen dafür ein kombinierten Operator vorgesehen. Es wird dem Zuweisungsoperator den arithmetische Operator **vorangestellt**.

```
number += 4; // Verkürzte Addition und Zuweisung zu sich selbst
```

Aus `number = number + 4;` wird somit `number += 4;` und erspart uns die Dopplung der Variable. Dies geht natürlich auch für alle anderen arithmetischen Operatoren.

```
number -= 7; // Subtraktion und Zuweisung zu sich selbst
number *= 9; // Multiplikation und Zuweisung zu sich selbst
number /= 2; // Division und Zuweisung zu sich selbst
```

## Unäre (einstellige) Operatoren `++`, `--`

Noch häufiger als die verkürzte arithmetische Operation mit sich selbst wird im Programmieren schrittweise hoch und runtergezählt **was auch Iteration genannt wird**.

Möchte man also von 0 - 3 hochzählen geht dies so:

```
int number = 0;
number += 1;
number += 1;
number += 1;
```

Der Unäre Operator `++` zählt der links vorangestellten Variable eines nummerischen Typ eine 1 hinzu.  
Der obere Code ist somit identisch zu diesem:

```
int number = 0;
number++;
number++;
number++;
```

### ⚠️ UNÄR => EINSTELLIG

Unär bedeutet einstellig, es braucht daher **nur der linke** und nicht auch einen rechten Teil beim Operator.

### 💡 ITERIEREN DURCH ARRAYS

Der unäre Operator `++` wird insbesondere beim **Iterieren durch Arrays** wie im folgenden Beispiel verwendet. Was gibt der obere Code wohl aus?

```
char[] text = {'h', 'a', 'l', 'l', 'o', ' ', 'w', 'e', 'l', 't'};

for (int i = 0; i < text.length; i++) {
    System.out.print(text[i]);
}
```

- Es wird Schrittweise jede Stelle vom Array `char[] text` in einem `for`-Loop ausgegeben
- Die Variable `i`, Iterator, wird durch `i++` für jeden Schritt +1 hochgezählt
- `i++` könnte auch mit `i += 1` oder `i = i + 1` ersetzt werden.
- `i++` ist jedoch viel kürzer.

### ⓘ FUNFACT

C++ erweitert die Programmiersprache C. Um diese Verwandtschaft ein bisschen NERDisch Auszudrücken wurde das Wortspiel C++ gewählt. C++ ist eine weitere Iteration von C.

## Vergleichsoperatoren `==`, `!=`, `!`

Vergleichsoperatoren ergeben immer einen **boolean (true/false)**. Sie werden in Kontrollstrukturen als Bedingungen verwendet.

### `==` Gleichheit

```
true == true; // true
1 == 1 // true

false == true; // false
1 == 2 // false
```

### `!=` Ungleichheit

```
false != true; // true
1 != 2 // true

true != true; // false
1 != 1 // false
```

### `!` Negation

```
!false // true
!(1 == 2) // true

!true // false
!(1 == 1) // false
```

Mit den numerischen Datentypen kann mit den bekannten Operatoren gerechnet werden. Also jene die aus der Mathematik bekannt sind: `+`, `-`, `*`, `/`.

Dann gibt es noch einige Spezialfälle in Java, die Sie früher oder später kennen lernen werden. Darum werden diese hier vorgestellt:

- `++` erhöht die Zahl um 1
- `--` reduziert die Zahl um 1

### Spezial-Operatoren in Java

```
int a = 1;
int b = 1;
a++; // a = a + 1;
System.out.println(a); // 1 + 1 = 2

b--; // b = b - 1
System.out.println(b); // 1 - 1 = 0
```

## 📺 Erklärvideos von Studyflix

- Java Datentypen einfach erklärt
- Java Variablen deklarieren und initialisieren
- Ausdrücke und Operatoren in Java
- Boolesche ausdrücke in Java

## Datentypen bestimmen

Bestimmen Sie die die kleinst möglichen Datentypen für die folgenden Werte:

11.39 ..... .

'b' ..... .

37 ..... .

true ..... .

"Hello" ..... .

## Datentypen initialisieren

Erstellen Sie ein Programm, in welchem Sie:

- für jeden Datentypen eine Variable **deklarieren**
- einen passenden Wert fest **zuweisen**
- und anschliessend die Variablen **ausgeben**
  - Versucht evt. Datentypen zu verbinden!

## Bonusauftrag einlesen

Sobald Sie den **obigen Auftrag abgeschlossen** haben, können Sie sich nach Vorgabe Ihrer Lehrperson (und sofern genügend Zeit verfügbar ist) eine Variante des Programms erstellen, bei der Sie die Werte mit dem `StdInput` von `mytools` einlesen.

### NICHT ALLE DATENTYPEN MÖGLICH

Die Klasse `StdInput` besitzt keine Methoden für die Datentypen `byte`, `short`, `long` und `float`. Diese müssen (können) also nicht eingelesen werden.

### SCANNER

Java besitzt die Klasse `Scanner`. Mit dieser können alle Datentypen eingelesen werden. Dies ist jedoch nicht statisch, braucht also ein Objekt, desswegen wird sie im Unterricht noch nicht vorausgesetzt. Interessiere, dürfen diese gerne testen!

## Noten berechnen

Erstellen Sie ein Programm, welches vom Benutzer

- drei Semesternoten einliest (`StdInput.read...`)
- den Durchschnitt berechnet
- sich dabei an ein strenges Notensystem hält, dass nur ganze Noten zulässt und immer abrundet
- den Durchschnitt ausgibt

### NOTE

- Anhand dieser Übung erkennen Sie, dass man mit Variablen und Zahlen ganz normal rechnen kann.
- Auch sollte nun das **EVA-Prinzip** klar werden

## Verwendung von Konstanten

- Erstellen Sie eine Klasse, welche die unten definierte Konstante MWST enthält.
- Erstellen Sie eine zweite Klasse mit der main-Methode.
- In der main-Methode verlangen Sie vom Benutzer einen Betrag.
  - Es soll also eine **Fliesskommazahl eingeben** werden können.
- Die eingegebene Zahl betrachten wir als Rechnungsbetrag ohne Mehrwertsteuer.
- Berechnen Sie nun den Mehrwertsteuerbetrag und geben Sie diesen auf die Konsole aus.

Konstante MWST in Prozent

```
public static final float MWST = 7.7;
```



**DIE KONSTANTE KÖNNEN SIE ÜBER DEN KLASSENNAMEN, GEFOLGT VON EINEM PUNKT ansprechen.**

## 3 - Kontrollstrukturen

Mit `if` und `switch` können Sie auf unterschiedliche Werte in Ihren Variablen unterschiedlich, **spezifisch reagieren**.

- Diese zwei Kontrollstrukturen benötigen **Bedingungen** um zu Entscheiden was für welche Situation gemacht werden soll.
- Dafür existiert in Java der Datentyp `boolean` und die Vergleichsoperatoren.

### Ziele

- Sie können die if Kontrollstruktur erklären und anwenden, indem Sie auch mit Operatoren und boolean-Werten umgehen können.
- Sie können die switch Kontrollstruktur nutzen, um effizienter auf eine kleinere Anzahl unterschiedlicher Werte reagieren zu können als mit einem if.

### Präsentation

 Open in Browser |  download PDF

### Erklärvideos von Studyflix

- [if-Anweisung einfach erklärt](#)
- [Switch-Anweisung einfach erklärt](#)

# if - Kontrollstruktur

Mit dem Schlüsselwort **if** (*engl. falls*) in Kombination mit dem **Datentyp boolean** können Programme erstellt werden, die abhängig von Benutzereingaben oder Variablenwerten unterschiedlich reagieren.

## if-Anatomie

Eine Kontrollstruktur mit einem **if** sieht so aus, wie im nächsten **formellen Beispiel** dargestellt. Es folgt ein **praktisches Beispiel** um es zu veranschaulichen.

## Formelles Beispiel

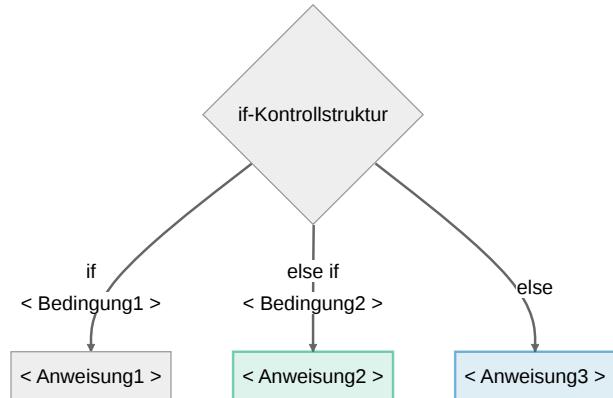
Hier ist nun eine **if-Kontrollstruktur** mit drei möglichen Codeblock Anweisungen dargestellt. Im laufenden Programm wird immer nur eine der Anweisungen ausgeführt. Dies ermöglicht es dynamisch auf die Benutzereingabe zu reagieren indem je nach Eingabe einen anderen Code ausgeführt wird.

Die Bezeichnungen **<Bedingung>** und **<Anweisung>** dienen als **Platzhalter**. Sie sind sogenannter Pseudocode und somit **nicht korrekter Java-Code**. Im späteren praktischen Beispiel werden diese mit korrektem Java-Code ersetzt.

### Aufbau einer if-Kontrollstruktur

```
if (<Bedingung1>) {  
    <Anweisung1>  
}  
else if (<Bedingung2>) { // Optionaler  
    Block  
    <Anweisung2>  
}  
else { // Optionaler  
    Block  
    <Anweisung3>  
}
```

### Flow Diagramm einer if-Kontrollstruktur



1. Der erste Block **if (<Bedingung1>) { <Anweisung1> }**
  - ist zwingend
  - und zwar von **if** bis zur ersten schliessenden geschweiften Klammer.
2. Der nächste Block **else if (<Bedingung2>) { <Anweisung2> }**
  - ist optional
  - kann auch noch mehrfach wiederholt folgen
  - kann nie alleine stehen (*nie ohne vorausgehender if-Block*)
  - wird ausgeführt, sofern die Bedingung vom vorausgehendem **if** oder **else if** Block **false** war
3. Der letzte Block **else { <Anweisung3> }**

- ist optional
- besitzt keine Bedingung
- darf nur ein Mal vorkommen, und zwar ganz am Schluss
- wird ausgeführt wenn kein anderer Block ausgeführt wurde

## Erläuterung

1. Wenn die `<Bedingung1>` wahr ist, wird nur die `<Anweisung1>` ausgeführt.
2. Wenn die `<Bedingung1>` falsch und die `<Bedingung2>` wahr ist, wird nur die `<Anweisung2>` ausgeführt.
3. Wenn die `<Bedingung1>` falsch und die `<Bedingung2>` falsch ist, wird nur die `<Anweisung3>` ausgeführt.



### TIP

Es wird immer nur ein Anweisungs-Block pro `if`-Kontrollstruktur ausgeführt!



### KLAMMERN

Die Buchhaltung mit Klammern wird nun wichtiger, und auch eine schöne Formatierung des Programms hilft Strukturen und Zusammenhänge richtig zu erkennen.

In eclipse gibt es den Shortcut `Ctrl + Shift + F`. Damit können Sie Ihren Sourcecode automatisch formatieren lassen.

## Praktisches Beispiel

Im praktischen Beispiel wollen wir herausfinden ob es sich um ein Kind, Jugendlicher oder Erwachsener handelt. Dazu existiert eine Variable `int age` in der das Alter gespeichert ist. In der if-Kontrollstruktur wird nun das Alter geprüft und je nach Situation in die Console geschrieben ob es sich um ein Kind, Jugendlicher oder Erwachsener handelt.

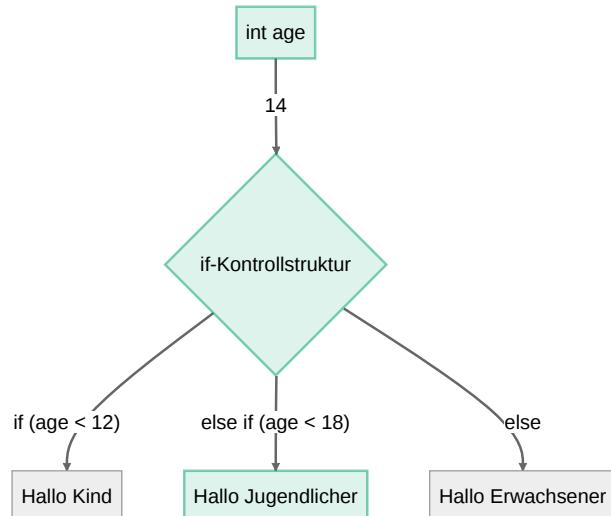
### Programmcode

```
1 int age = 14;
2
3 if (age < 12) {
4     System.out.println("Hallo
5     Kind");
6 } else if (age < 18) {
7     System.out.println("Hallo
8     Jugendlicher");
9 } else {
10    System.out.println("Hallo
11     Erwachsener");
11 }
```

### Flussdiagramm

### TIP

Die Variable `int age` könnte nun auch von der Konsole eingelesen werden und somit "dynamisch" sein. Dann macht die Kondition auch mehr Sinn!



### Erläuterung

1. Es existiert eine Variable `age` vom Typ `int` mit dem Wert `14`
2. Die `if`-Block Bedingung (**Linie 3**) wird ausgeführt
  - Die Bedingung prüft ob es sich um ein Kind handelt, also kleiner als 12 ist
  - Da der Wert von `age` `14` ist, ist der `boolean` der Prüfung `false`, also falsch
  - Der Code-Block (**Linie 4**) wird übersprungen
3. Die `else if` Bedingung (**Linie 6**) wird nun ausgeführt
  - Es wird geprüft ob der Wert von `age` kleiner als 18 ist.
  - Da der Wert `14` kleiner als `18` ist, ist der `boolean` der Prüfung `true`, also richtig
  - Der `else if`-Block (**Linie 7**) wird ausgeführt.
4. Da eine Bedingung `true` war, wird der `else-Block (Linie 10)` übersprungen!

## Bedingungen, Boolesche Ausdrücke, Typ `boolean`

Aber was ist eine Bedingung?

Bedingungen sind Code-Ausdrücke, welche entweder wahr (`true`) oder falsch (`false`) sein können. Diese werden mit dem Datentyp `boolean` ausgedrückt und sind elementar für die `if`-Kontrollstruktur um zu entscheiden, welcher Codeblock genau ausgeführt werden soll.

Beispielsweise könnte eine Kassensoftware Kunden, die für mehr als 100 CHF einkaufen einen Rabatt von 10% einräumen. Dies sieht dann so aus:

```
if (amount > 100) { // Boolescher Ausdruck direkt in der if-Kontrollstruktur
    amount = amount * 0.9; // 10% discount
}
```

Stattdessen könnte man auch eine **Variable** `boolean giveDiscount` (gebe Rabatt) einführen. Dies hat den Vorteil, dass ein guter Variablename direkt beschreibt was gemacht wird. Dies würde dann so aussehen:

```
boolean giveDiscount = amount > 100;

if (giveDiscount) { // Variable vom Datentyp `boolean` in der if-Kontrollstruktur
    amount = amount * 0.9; // 10% discount
}
```

Hier noch mehr Beispiele wie Boole'sche Ausdrücke in boolean Variablen gespeichert und verwendet werden können. Laut Konvention beginnen diese Variablennamen mit `is` (zu Deutsch "ist").

#### Beispiel Boole'sche Ausdrücke in Variablen

```
// mit int
int age = 21; // gegeben ist eine int Variable
boolean is21 = age == 21; // true
boolean isNot21 = age != 21; // false, oder !is21
boolean isAdult = age >= 18; // true
boolean isChild = age <= 18; // false

// Diese Variablen können nun in der if-Kontrollstruktur als Bedingung verwendet werden
if (is21) {
    System.out.println("Das Alter ist genau 21");
} else if (isAdult) {
    System.out.println("Es handelt sich um eine erwachsene Person");
}
```

## Kombination von Bedingungen und Wahrheitswerten

Manchmal müssen mehrere Bedingungen kombiniert werden. Dies kann mit einem logischen UND (`&&`) sein oder mit einem logischen ODER (`||`). Nur wie stellt man dies in Java dar?

Logischer Ausruck	Bedeutung	Beschreibung
<code>&amp;&amp;</code>	Logisches UND	alle Werte müssen <code>true</code> sein
<code>  </code>	Logisches ODER	mindestens ein Wert muss <code>true</code> sein

Beim logischen **UND**, `&&`, müssen beide, resp. **alle Werte** `true` sein. Sobald ein `false` auftritt, ist alles `false`:

Kombination	Resultat
<code>true &amp;&amp; true</code>	<code>true</code>
<code>true &amp;&amp; false</code>	<code>false</code>

Beim logischen **ODER**, `||`, muss **mindestens ein Wert** `true` sein. Sobald ein `true` auftritt, ist alles `true`:

Kombination	Resultat
<code>true    true</code>	<code>true</code>
<code>true    false</code>	<code>true</code>

Kombination	Resultat
false && false	false

Rabatt für Einkäufe über 100 CHF UND Kunden jünger als 18 Jahre:

Beispiel: UND

```
if (amount < 100 && age < 18) {
    amount = amount * 0.9d;
}
```

Kombination	Resultat
false    false	false

Rabatt für Einkäufe über 100 CHF ODER Kunden jünger als 18 Jahre:

Beispiel: ODER

```
if (amount < 100 || age < 18) {
    amount = amount * 0.9d;
}
```



### TIP

Die beiden senkrechten Striche werden auf CH-PC-Tastaturen häufig mit **ALT-GR plus Taste 7** erzeugt (auf der Taste ist das meist mit einem gelegentlich noch unterbrochen vertikalen Strich dargestellt).

## Vergleichsoperatoren

In den obigen Beispielen wurden bereits einige Vergleichsoperatoren verwendet. Unter anderem folgende sechs Vergleichsoperatoren stehen zur Verfügung: `<`, `<=`, `>`, `>=`, `==`, `!=`. Zusätzlich existiert die Methode `equals` für Strings.

Operator	Bedeutung	true Beispiel
<code>==</code>	gleich	<code>1 + 1 == 2</code>
<code>!=</code>	ungleich	<code>1 + 1 != 4</code>
<code>&lt;</code>	kleiner als	<code>1 &lt; 2</code>
<code>&gt;</code>	grösser als	<code>2 &lt; 1</code>
<code>&lt;=</code>	kleiner gleich	<code>2 &lt;= 2, 1 &lt;= 1</code>
<code>&gt;=</code>	grösser gleich	<code>1 &gt;= 1, 2 &lt;= 1</code>
<code>equals</code>	gleicher String	<code>"hallo".equals("hallo")</code>

## Der Spezialfall `String.equals`

Es gibt eine Ausnahme beim gleich Vergleichs-Operator (`==`) und dem Datentyp `String` für Zeichenketten. Strings dürfen nicht mit `==` verglichen werden, sondern mit dem **Aufruf der Methode `equals`** wie hier gezeigt:

```
String vehicle = "auto"; // gegeben ist eine String Variable

if ("auto".equals(vehicle)) { // es wird mit `equals` geprüft
    System.out.println("Es handelt sich um ein Auto!");
}

// oder mit Variable

boolean isAuto = "auto".equals(vehicle); // true
if (isAuto) {
    System.out.println("Es handelt sich um ein Auto!");
}
```

### ! INFO

- **String** Variablen **immer mit `.equals()`** vergleichen, nie mit `==`
- Auch ist es gute Praxis der **bekannte Wert mit dem unbekannten zu vergleichen** und nicht anders rum:
  - ✓ `"auto".equals(vehicle);`
  - ✗ `vehicle.equals("auto");` ergibt ein NullPointer wenn die `vehicle` den Wert `null` besitzt.

## Auftrag 1

Erstellen Sie ein Programm, welches:

- einen Kaufbetrag entgegen nimmt (*Eingabe einlesen*)
- bei Einkäufen über CHF 100 einen Rabatt von 15% gewährt

## Auftrag 2

Erstellen Sie Programm, welches:

- Gewichtsangaben entgegen nimmt (*Eingabe einlesen*)
- Bis und mit 5kg CHF 2 für Kleinpakete verrechnet
- Zwischen 5kg und 10kg CHF 5 für Mittelpakete verrechnet
- Ab 10kg für Grosspakete CHF 10 verrechnet
- Für Pakete  $\geq 15\text{kg}$  (ab 15kg) wird auf die Spedition verwiesen. Diese können nicht versendet werden.

## Auftrag 3

Bauen Sie folgenden Abschnitt in ein Programm ein und untersuchen Sie, warum die Ausgabe des Programms falsch ist. Warum ist dieser Fehler so schwer zu entdecken?

```
int value = 50;  
if (value > 100) ; {  
    System.out.println("The value is larger than 100");  
}
```

## Auftrag 4 - Reflexion

Inzwischen haben Sie eine Vorstellung, was Syntax in Java bedeutet. Sie haben inzwischen auch das eine oder andere Programm erstellt. Vermutlich waren einige dieser ersten Aufgaben nicht einfach lösbar, nur schon das Beachten der Klammern, deren Paare, und auch die Datentypen, wie man Werte in diesen speichert.

Darum kehren wir das Vorgehen nochmals um:

- Wie können Aufgaben in der Programmierung lösbarer werden?
- Was denkt man am besten, in welcher Reihenfolge?
  - Fangen Sie bei den geschweiften Klammern an?

Folgende Bausteine könnten Sie in der Vorarbeit zu einer Aufgabe unterstützen:

- Eigene Problembeschreibung (Ziel des Programms)
- In welcher Abfolge soll das Programm erstellt werden (damit möglichst einfach)
- Welche Daten sind zu bearbeiten und mit welchen Datentypen?
- Welche Kontrollstrukturen sind zu welchem Zweck zu verwenden?

## Aufgabe

Beschreiben Sie nun schriftlich, aus Ihrer Sicht als Programmiererin oder Programmierer, **wie** man eine der vorherigen Aufgaben angeht und löst. Probieren Sie möglichst, praktisch jedes Detail zu beschreiben und ohne Aussagen "ja, das weiss ich einfach" auszukommen.

### ÜBEN

Einer der wichtigsten Punkte beim Programmieren ist das Üben. Das oberhalb beschriebene Vorgehen kann dabei helfen, mit dem Lösen solcher Aufgaben vertrauter zu werden. Probieren Sie hin und wieder, besonders bei anspruchsvollen Aufgaben, sich den idealen Denk- und Arbeitsplan für die Programmierung einer Lösung bewusst zu machen und schriftlich zu dokumentieren.

# Switch - Kontrollstruktur

Wir haben bereits die if-Verzweigung kennengelernt. Dort kann anhand einer Bedingung eine Wahl zwischen zwei Möglichkeiten getroffen werden. Das ist eine sehr gute Möglichkeit, ein Programm flexibel zu machen.

Aber manchmal gibt es Situationen, bei denen man zwischen **mehr als zwei Möglichkeiten** unterscheiden möchte. So eine Situation lässt sich lösen, indem man etliche ifs nacheinander hängt.

Hier ein Auszug aus einer Übungsaufgabe, in der ein Würfelspiel implementiert wurde:

```
int number = rollDice();
if (number == 1) {
    countOne++;
} else if (number == 2) {
    countTwo++;
} else if (number == 3) {
    countThree++;
} else if (number == 4) {
    countFour++;
} else if (number == 5) {
    countFive++;
} else if (number == 6) {
    countSix++;
}
countAll++;
```

## ⓘ NOTE

Das obere Beispiel ist nicht alleine lauffähig, dafür fehlt die Methode `rollDice()` sowie die Variable `countOne`.

Für solche Mehrfachverzweigungen gibt es auch die Kontrollstruktur `switch`. Switch ermöglicht es, **auf genaue**, unterschiedliche Werte zu reagieren (`==`). Dabei können die Wert numerisch oder ein String sein. Das Muster sieht so aus:

```
switch (<Ausdruck>) {
    case value:
        <Anweisung>;
        break;
    case value:
        <Anweisung>;
        break;
    default:
        <Anweisung>;
        break;
}
```

Nach dem Schlüsselwort steht in runden Klammern eine Variable, deren Werte in den anschliessenden case-Blöcken ausgewertet werden. Besteht für den aktuellen Wert der Variablen ein passender case-Block, springt das Programm dort hinein, führt die Anweisungen aus, und springt aufgrund der break-Anweisung aus dem switch heraus (ohne break würden alle nachfolgenden Blöcke auch durchlaufen). Das default am Schluss ermöglicht es, Anweisungen für alle anderen Fälle festzulegen, in denen kein exakter Wert in einem case steht.

Ein konkretes Beispiel sieht so aus:

```
public class SwitchDemo {  
    public static void main(String[] args) {  
        int month = 8;  
        String monthString;  
        switch (month) {  
            case 1:  
                monthString = "January";  
                break;  
            case 2:  
                monthString = "February";  
                break;  
            //... andere Fälle hier ausgelassen  
            case 8:  
                monthString = "August";  
                break;  
            //... andere Fälle hier ausgelassen  
            default:  
                monthString = "Invalid month";  
                break;  
        }  
        System.out.println(monthString);  
    }  
}
```

### TIP

Häufig kann ein `switch` durch einen `array` elegant ersetzt werden:

```
int month = 8;  
int[] monthStrings = {  
    "January", "February", "March", "April", "May", "June",  
    "July", "August", "October", "November", "December"  
}  
if (month < monthStrings.length) { // prüft ob `month` gültig ist  
    System.out.println(monthStrings[month - 1]);  
} else {  
    System.out.println("Invalid month");  
}
```

## Auftrag 1

Erstellen Sie eine Klasse, welche für die Eingabe einer (vereinfachten, ganzen) Note die entsprechende textuelle Bewertung ausgibt resp. Fehlermeldung bei ungültiger Note:

- sehr gut
- gut
- genügend
- ungenügend
- schwach
- sehr schwach
- falsche Eingabe

## Auftrag 2

Programmieren eines Rechners mit switch.

Erstellen Sie ein Programm, das zwei Zahlen plus einen Operator einliest. Berechnen Sie dann mittels des switch-Statements, welche Rechenart erforderlich ist, führen die Berechnung aus und geben das Ergebnis aus.

## Auftrag 3

Die Beaufort-Skala wurde 1806 von dem englischen Admiral Sir Francis Beaufort (1774 – 1857) erarbeitet. Mit ihrer Hilfe kann anhand der Auswirkungen des Windes die Windstärke geschätzt werden. Sie reicht von Stärke 0 (Windstille) bis Stärke 12 (Orkan).

### NUR ZUR INFO

Die Aufgabe lässt sich ohne das genaue Studieren dieser Tabelle lösen ;)

Beaufort	Windgeschwindigkeit m/s      km/h		Bezeichnung	Beispiele für Auswirkungen des Windes im Binnenland
0	0,0 – 0,2	< 1	Windstille	Rauch steigt senkrecht auf.
1	0,3 – 1,5	1 – 5	Leiser Zug	Windrichtung wird nur durch Rauch angezeigt, nicht durch Windfahnen.
2	1,6 – 3,3	6 – 11	Leichter Wind	Wind am Gesicht fühlbar, Blätter säuseln, Windfahne bewegt sich.
3	3,4 – 5,4	12 – 19	Schwacher Wind	Blätter und dünne Zweige bewegen sich, Wind streckt Wimpel.
4	5,5 – 7,9	20 – 28	Mäßiger Wind	Wind hebt Staub und loses Papier, bewegt Zweige und dünnere Äste.
5	8,0 – 10,7	29 – 38	Frischer Wind	Kleine Laubbäume beginnen zu schwanken, Schaumkronen bilden sich auf Seen.
6	10,8 – 13,8	39 – 49	Starker Wind	Starke Äste in Bewegung, Pfeifen in Telegrafen- und Stromleitungen, Regenschirme schwierig zu halten.
7	13,9 – 17,1	50 – 61	Steifer Wind	Ganze Bäume in Bewegung, fühlbare Hemmung beim Gehen im Freien.
8	17,2 – 20,7	62 – 74	Stürmisches Wind	Wind bricht Zweige von Bäumen, erschwert erheblich das Gehen im Freien.
9	20,8 – 24,4	75 – 88	Sturm	Äste brechen von Bäumen, kleinere Schäden an Häusern (Rauchhauben + Dachziegel werden abgeworfen).
10	24,5 – 28,4	89 – 102	Schwerer Sturm	Wind bricht Bäume, größere Schäden an Häusern.
11	28,5 – 32,4	103 – 117	Orkanartiger Sturm	Wind entwurzelt Bäume, verbreitet Sturmschäden.
12	ab 32,5	ab 118	Orkan	Schwere Verwüstungen.

Es gelten die folgenden Annahmen:

- Beaufort 0 - 8: es besteht keine Gefahr
- Beaufort 9: es bestehen mögliche Gefahren
- Beaufort 10 - 12: es bestehen erhebliche Gefahren

Ein Benutzer will wissen, **ob für eine gewisse Windstärke eine Gefahr besteht**. Schreiben Sie ein kleines Programm dafür. Das Programm soll folgendes machen:

- Nach der Windstärke in Beaufort fragen und den Wert einlesen.
- Einen Gefahren-text ausgibt

## Extra: switch oder if ?

Sie werden nachvollziehen können oder bereits festgestellt haben, dass die `switch` Kontrollstruktur fast immer durch mehrfache `if-else` ersetzt werden könnte.

Die folgende Tabelle stellt Argumente gegenüber:

Kontrollstruktur	Beschreibung	Begründung
if	jedes Mal wird das Ergebnis eines boolean Ausdrucks berechnet	überschaubare Anzahl Fälle; alle Datentypen möglich
switch	definierte, einzelne Werte werden geprüft	etwas besser lesbar bei vielen Varianten; nur möglich mit Zeichen, Strings und Ganzzahlen

### ① HERR HODELS MEINUNG

Grundsätzlich kann man auf Switch verzichten. Ist `switch` wirklich leserlicher? Vor allem das `break` nach jeder Anweisung macht es häufig komplexer.

Eigentlich macht `switch` nur dann Sinn, wenn mehrere `<Anweisungen>` bei spezifischen Werten ausgeführt werden sollen. Also dann, wenn man das `break` nicht schreiben müsste. Nun hat sich in der Praxis ergeben, dass dies fast nie der Fall ist.

# Weitere Übungen

## Quelltext / Code lesen / interpretieren können

Überlegen Sie die Wirkung der nachfolgenden Ausschnitte aus grösseren Programmen. Lesen Sie die Frage und schreiben die Antwort auf ein Blatt Papier:

Welcher Wert für r ausgegeben?

```
int a, b, c, d, r;  
a = 5;  
b = -2;  
c = 0;  
d = 1;  
r = a + b + c + d;  
System.out.println(r);
```

Welche Ausgabe erfolgt?

```
int x, y;  
x = 2;  
y = 3;  
  
if (x > y) {  
    System.out.println("case x");  
} else {  
    System.out.println("case y");  
}
```

Welcher Wert wird berechnet?

```
double fahrenheit = 212;  
double celsius = (fahrenheit - 32.0) / 1.800;  
System.out.println(celsius);
```

## Fehler in Quelltext / Code entdecken können

Sehen Sie die nachfolgenden Programme an. Entdecken Sie die Fehler und schreiben Sie die Korrektur direkt in auf dieses Blatt.

Finden Sie den logischen Fehler:

```
public class PriceForQuantity {  
    public static void main(String[] args) {  
        int quantity = 8;  
        double price = 1.8;  
        double amountDue = quantity / price;  
        System.out.println(amountDue);  
    }  
}
```

Finden Sie die 3 Fehler:

```
public class HelloUser {  
    public static void main(String[] args) {  
        String name = readString();  
        Sysout("Hello, " name);  
    }  
}
```

## Quelltext / Code für kleine Programme schreiben können

### Programm 1

Erstellen Sie ein Programm, welches Meilen einliest und in km umrechnet und ausgibt.

Die Berechnung erfolgt so: `km = miles / 0.62137;`

Erweiterung nachdem alle Programme fertig sind:

- Umkehrung, km einlesen, Meilen ausgeben.
- Auswahl durch den Benutzer.

### Programm 2

Erstellen Sie ein Programm, welches

- Distanz und Zugsgeschwindigkeit einliest
- die Reisedauer berechnet.

Folgende Werte können Sie als test verwenden:

- Distanz 120
- Geschwindigkeit 80
- Dauer: 1.5.

Erweiterung nachdem alle Programme fertig sind:

- Eingabe der Distanz und Reisedauer berechnet durchschnittliche Geschwindigkeit.

## Programm 3

Erstellen Sie ein Programm, welches das aktuelle Datum und Uhrzeit ausgibt.

## Programm 4

Erstellen Sie ein Programm, welches die folgende Ausgabe erzeugt (rechnen Sie mit ganzen int-Zahlen):

```
Wollen Sie auf Ihrer Pizza zusätzlich Schinken? (true/false)  
true  
Wollen Sie auf Ihrer Pizza zusätzlich Oliven?(true/false)  
false  
Ihre Pizza setzt sich wie folgt zusammen und kostet daher  
Pizza Fr. 15  
Zusätzlicher Schinkenbelag Fr. 2  
Gesamtpreis Fr. 17
```

## Programm 5

Erstellen Sie ein Programm, welches einen Biletpreis abhängig von der Distanz berechnet.

- Der Basispreis pro Kilometer ist 0.45 Einheiten.

## Aufgabe Club

In dieser Aufgabe können die Themen Ausgabe, Eingabe, Datentypen (Variablen) und die Kontrollstruktur if geübt werden.

In einen Club dürfen maximal 4 Personen auf einmal. Der Türsteher begrüßt die ankommenden Gäste und fragt nach, wie viele Personen Sie sind. Bei mehr wie 4 Personen wird der Einlass mit der Aussage «Sie sind leider zu viele Personen» verweigert. Bei weniger oder gleich 4 Personen wird nach den Personalien einer Person gefragt. Die Daten Vor- und Nachname, Strasse, Plz, Ort und Telefonnummer werden abgefragt und gespeichert.

- Wählen Sie dazu möglichst passende Datentypen.
- Danach folgt die Frage, ob sie schon 18 Jahre alt sind. «true» bedeutet ja, «false» nein.
- Wird «true» eingegeben, wünscht der Türsteher einen schönen Abend und wiederholt die Angaben.
- Bei «false» wird der Zutritt verweigert.

Beispiel Ablauf, wenn alles in Ordnung ist:

Willkommen im Club Flamingo

---

Einlass möglich für maximal 4 Personen auf einmal. Wie viele Personen sind Sie:  
4 Wir benötigen die Personalien von einer Person. Vorname: Hans Nachname: Muster  
Strasse: Musterweg 1 Plz: 4000 Ort: Mustersta  
Telefon: 079 999 99 99 Sind Sie schon 18 Jahre alt? true Einen schönen Abend im  
Flamingo. Ihre Angaben: Hans Muster Musterweg 1 4000 Mustersta  
Tel: 079 999 99 99

Reaktion des Programms, wenn es zu viele Personen sind:

Willkommen im Club Flamingo

---

Einlass möglich für maximal 4 Personen auf einmal. Wie viele Personen sind Sie:  
5 Sie sind leider zu viele Personen.

Reaktion des Programms, wenn das Alter kleiner wie 18 ist.

Willkommen im Club Flamingo

---

Einlass möglich für maximal 4 Personen auf einmal. Wie viele Personen sind Sie:  
4 Wir benötigen die Personalien von einer Person. Vorname: Hans Nachname: Muster  
Strasse: Musterweg 1 Plz: 4000 Ort: Mustersta  
Telefon: 079 999 99 99 SSind Sie schon 18 Jahre alt? false Sie sind leider zu  
jung und dürfen nicht in den Club.ind Sie schon 18 Jahre alt?

## 4 - Methoden und Fehleranalyse (Debugging)

Sie erhalten einen **ersten Einblick in Methoden**, die für grössere Programme zur Strukturierung unerlässlich sind. Zusätzlich wird Ihnen das **Verfahren des Debuggings** vorgestellt, da es besonders hilfreich sein kann, um Methoden besser zu verstehen und Ihnen auch künftig hilft, Fehler zu finden.

### ⌚ Ziele

- Sie können einfache Methoden erklären und anwenden.
- Sie können mittels Debugging Ihr Programm schrittweise durchlaufen, dabei die Werte von Variablen beobachten und so Abläufe verstehen und Fehlerquellen entdecken.
- Sie kennen die Methode `main` inzwischen und können nun begründen, warum ein Programm mit Methoden besser strukturiert werden kann.
- Sie können Methoden aufrufen, diesen Parameter-Werte übermitteln und Rückgabewerte verwenden.
- Sie kennen den Geltungsbereich von Variablen und können erklären, ob eine Variable aus der Main-Methode auch in anderen Methoden zur Verfügung steht.
- Sie können bei Laufzeitfehlern (das Programm stürzt ab) den Stacktrace erkennen und von dort angezeigte Fehlerstellen in Ihrem Programm anspringen.

### Was ist eine Methode?

Die bisher in diesem Modul entwickelten Programme sind alle mit einer Methode, der Methode `main` ausgekommen. In der Praxis sind Programme jedoch wesentlich komplexer. Daher werden sie auf verschiedene Art strukturiert und aufgeteilt. Einen ersten Ansatz, um Programme in diesem Sinne zu gliedern, stellen Methoden dar.

Eine Methode kann vereinfacht als **Gruppierung von Befehlen** angesehen werden, welche in Kombination eine **spezifische Funktionalität** unter einem **aussagekräftigem Namen** beschreiben.

Eine mathematische Funktion beschreibt z.B. der Satz des Pythagoras  $a^2+b^2 = c^2$

Satz des Pythagoras

```
public double pythagoras(double a, double b) {  
    return (a*a) + (b*b);  
}
```

Im Programmieren kann man jedoch nicht nur Rechnen, sondern z.B. auch einfach etwas auf die Konsole ausgeben. Die folgende Methode gibt das aktuelle Datum aus:

printCurrentDate() gibt das aktuelle Datum aus

```
public static void printCurrentDate() {  
    System.out.println("Current date is: " + LocalDate.now());  
}
```

Methoden dienen also dazu spezifische Funktionalitäten unter einem aussagekräftigem Namen zu Gruppieren, damit diese

1. nur einmal definiert werden müssen
2. an verschiedenen Orte ausgeführt werden können
3. das Programm lesbarer machen, da der Name beschreibt was sie macht

#### **METHODE ODER FUNKTION?**

Grundsätzlich ist der Name Methode ein Synonym zu Funktion. In objektorientierten Programmiersprachen wie Java spricht man jedoch von Methoden. Eine "Daumenregel" ist:

- Eine Funktion in einer Klasse nennt man Methode.
  - Eine Methode besitzt immer einen Zeiger auf das Objekt (`this` in Java, `self` in Python und Ruby, ...)
- Eine Funktion ausserhalb einer Klasse, nennt man Funktion.

 **In Java gibt es nur Funktionen in Klassen, daher wird immer von Methoden gesprochen!**

## Was ist Debugging?

Debugging ist eine Arbeitstechnik in der Programmierung, welche es erlaubt, den Ablauf eines Programms besser nachzuvollziehen. Dabei kann das Programm in einem speziellen Modus ausgeführt werden, der es erlaubt, das Programm auf jeder ausführbaren Codezeile anzuhalten, die in den Variablen gespeicherten Werte einzusehen, zu ändern, etc.

Dies hilft um zu verstehen, wie sich das Programm genau verhält und unterstützt somit die Fehlersuche von Laufzeitfehlern.

## Erklärvideos von Studyflix

- Java Methoden einfach erklärt
- Parameter von Methoden einfach erklärt

# Einfacher Methodenaufruf

Das folgende Beispiel zeigt einen einfachen Methodenaufruf, ohne Parameter.

Die hervorgehobenen Linien zeigen den grundsätzlichen Ablauf des Programms. Wie bisher bekannt, wird das Programm in der `main`-Methode ausgeführt, und die Anweisung welche dort steht, ruft die Methode `printCurrentDate` auf, die ihre Aufgabe, das aktuelle Datum auszugeben, ausführt.

```
import java.time.LocalDate ;  
  
public class OutputMethodExample {  
    public static void main(String[] args) {  
        printCurrentDate();  
    }  
  
    public static void printCurrentDate() {  
        System.out.println("Current date is: " + LocalDate.now());  
    }  
}
```

- Das Programm startet auf **Zeile 5**
- und springt sogleich durch den Methodenaufruf `printCurrentDate()` zur **Zeile 9**, in den Methodenkörper von `printCurrentDate`.

## Verwendete Schlüsselwörter

### `public`: Sichtbarkeit

An erster Stelle wird die Sichtbarkeit definiert. Die Methode wird `public` definiert und ist somit von überall aus sichtbar.

- `public` für alle sichtbar
- `private` nur für die entsprechende Klasse selbst sichtbar
- `nicht-angegeben` im eigenen Modul sichtbar (**nie leer lassen bitte!**)

**⚠ VORDERHAND GENÜGT ES, WENN ALLE METHODEN PUBLIC SIND.**

### `static`: Art (Optional)

An zweiter Stelle steht `static`. Dies muss so sein, da die Methode von der ebenfalls statischen `main`-Methode aus aufgerufen wird.

- Das Schlüsselwort `static` ist **Optional!** Wenn nicht vorhanden ist die Methode immer dynamisch.

- Dieses Schlüsselwort kennzeichnet Variablen und Methoden, welche keine dynamischen Werte verwendet. Methoden welche `static` sind, dürfen nur Konstanten Verwenden. Konstanten sind Variablen welche mit `static` gekennzeichnet sind.
- Will man aus einer statischen Methode (wie z.B. der `main`-Methode) eine andere Methode aufrufen, muss diese auch statisch sein.

## void: Rückgabewert

An dritter Stelle wird durch `void` angegeben, dass diese Methode keinen Rückgabewert besitzt. Dadurch braucht die Methode auch kein `return` Befehl.

### NOTE

Wäre `static` weggelassen würde `void` an zweiter Stelle stehen.

- `void` bedeutet, dass die Methode keinen Rückgabewert besitzt
- Steht ein Datentyp wie z.B. `int`, `String`, muss die Methode diesen Datentyp durch das Schlüsselwort `return` zurückgeben.
- **Sie lernen Parameter und Rückgabewerte später im Detail**

# Debugging / Fehleranalyse

Debugging ist eine Arbeitstechnik in der Programmierung, welche es erlaubt, den Ablauf eines Programms besser nachzuvollziehen. Dabei kann das Programm in einem speziellen Modus ausgeführt werden, der es erlaubt, das Programm auf jeder beliebigen Codezeile anzuhalten, die in den Variablen gespeicherten Werte einzusehen, zu ändern, etc.

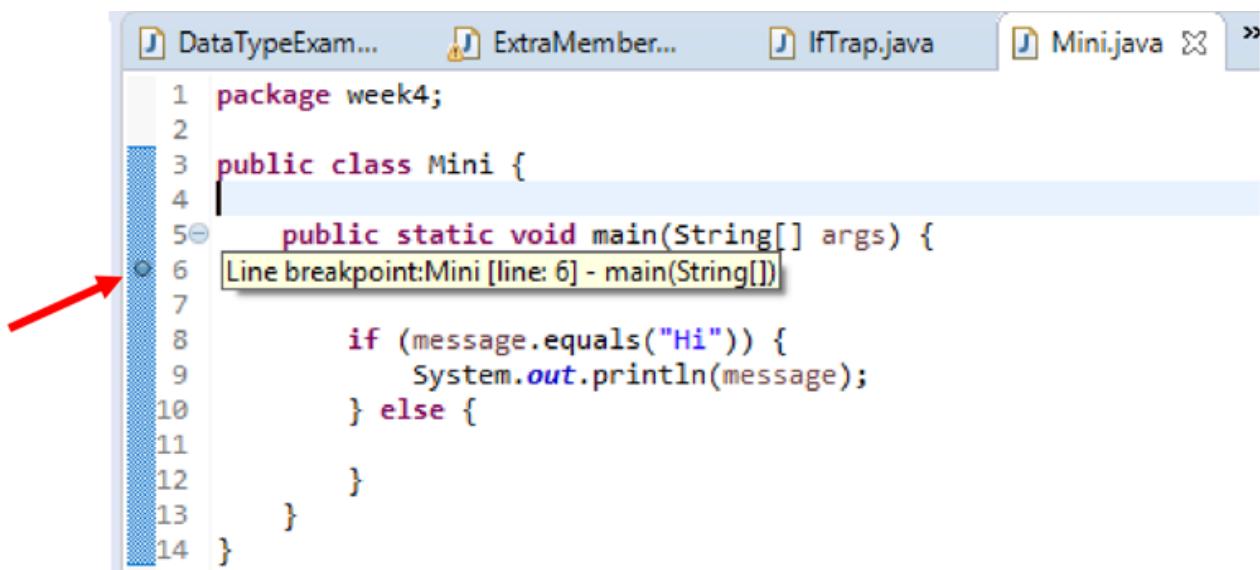
## Eine Debugging-Session Schritt-für-Schritt

Folgende Schritte sind nötig, um eine Debugging-Session für folgenden Programmcode durchzuführen:

Das verwendete Code-Beispiel

```
public class Mini {  
    public static void main(String[] args) {  
        String message = "Hi";  
  
        if (message.equals("Hi")) {  
            System.out.println(message);  
        } else {  
            // nix tun  
        }  
    }  
}
```

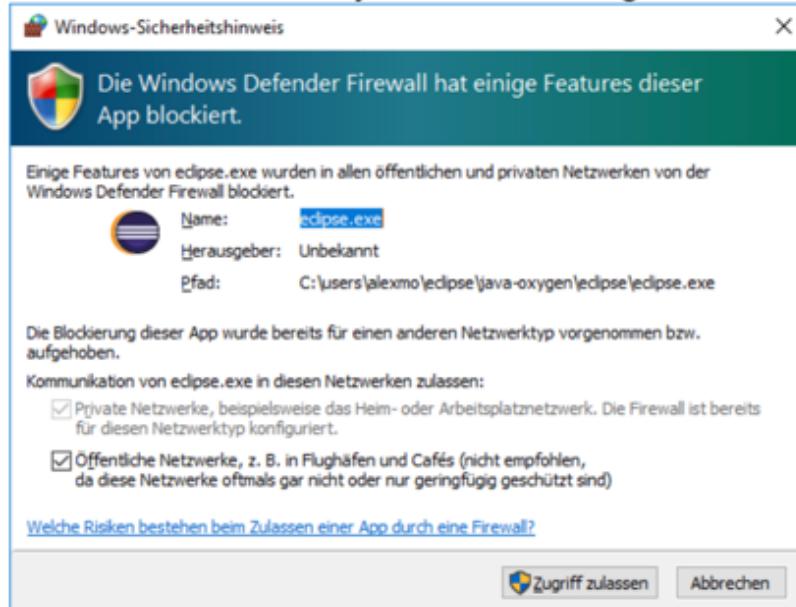
1. Einen (oder mehr) Breakpoint (Haltepunkt) setzen



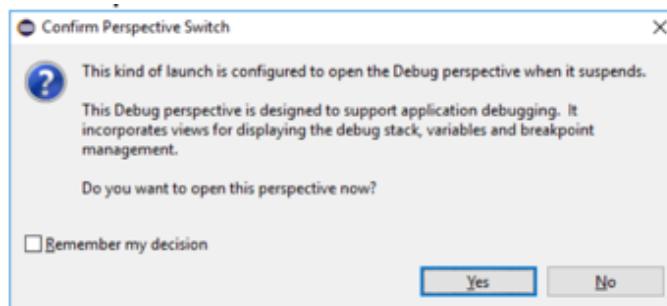
! DAZU AUF DAS BLAUE BAND LINKS VOM CODE DOPPELKICKEN

2. Das Debugging durch den „Käfer-Button“ starten

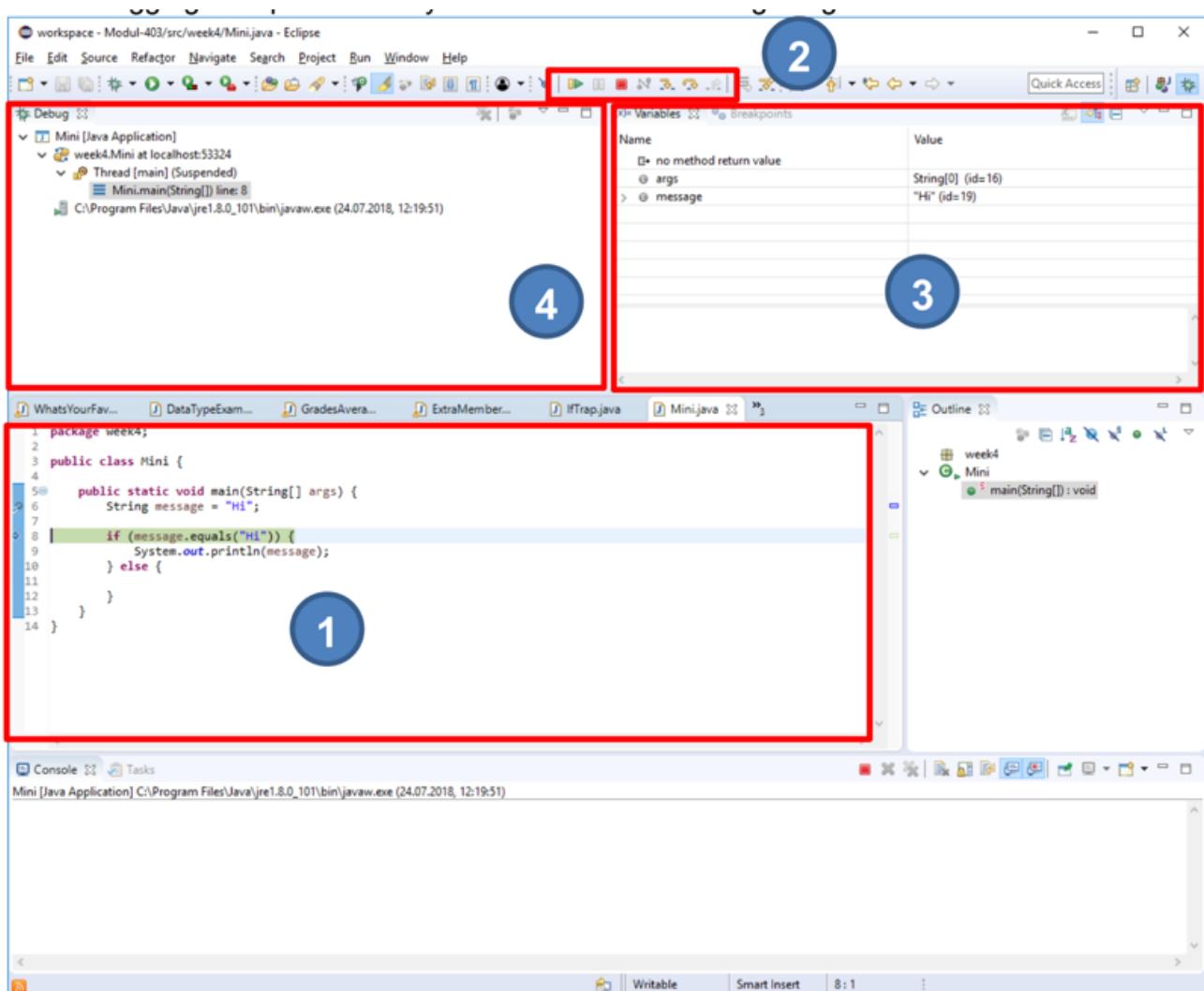
3. Auf Ebene Betriebssystem bei allfälliger Nachfrage „Zugriff zulassen“ wählen



4. In Eclipse den Wechsel in eine andere Perspektive („Ansichtsart“) bestätigen



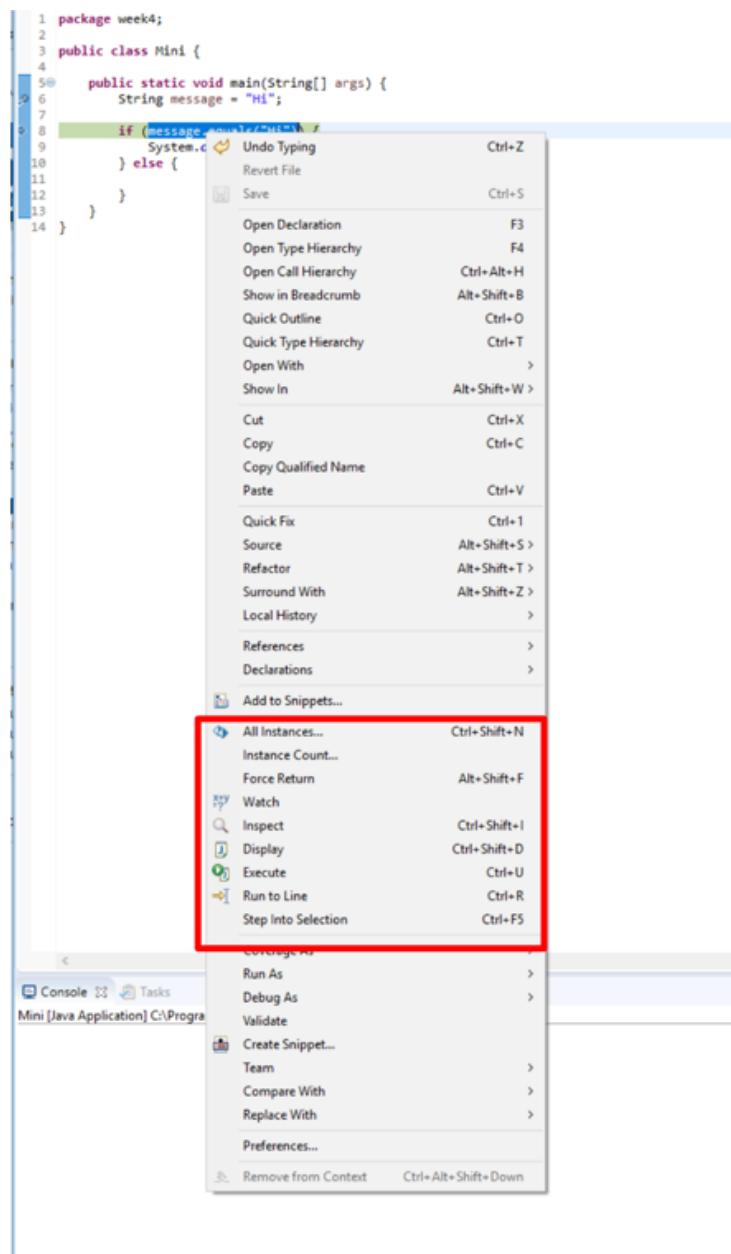
5. Die Debugging-Perspektive analysieren



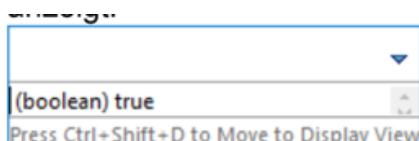
Was wird hier angezeigt?

- **(1)** Code-Fenster: die Zeile mit dem Breakpoint, grün unterlegt ist der Code, der beim nächsten Schritt ausgeführt wird
- **(2)** Buttons zur Steuerung:
  - **Resume:** Programm weiter laufen lassen, bis zum nächsten Breakpoint
  - Pause: Kann ignoriert werden
  - **Stop:** Debugging Session wird beendet
  - **Step-Into:** Springt zur **nächsten Linie welche ausgeführt wird**. Also auch weiter zu einer möglichen Methode. So kann manuell durch den gesamten Programmablauf navigiert werden.
  - **Step-Over:** Springt zur **nächsten Linie in der aktuellen Methode**. Überspringt also den Aufruf einer möglichen Methode auf der aktuellen Linie.
- **(3)** Variablen-Fenster: hier sind alle bereits initialisierten Variablen und deren Werte einsehbar (das Programm ist bereits einen Schritt weiter als der Breakpoint)
- **(4)** Debug-Fenster: alle laufenden Debug-Prozesse (mit Doppel-x am oberen Rand aufräumen)

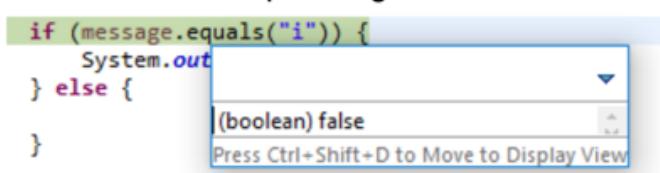
## 6. Einzelne Ausdrücke genauer untersuchen (Rechtsklick auf Linie)



Als Beispiel wurde **Display** ausgewählt, welches den Ausdruck ausführt, und das Ergebnis anzeigen:



Nun kann man **Anpassungen am Code vornehmen**, und sofort die **Folgen beobachten**. Wenn man z.B. "Hi" nach "i" ändert, wird der Ausdruck `false`, da `String message = "Hi"` ist und nicht "i"



7. Das Debugging wird über den Stopp-Button beendet. Anschliessend können Sie in der Entwicklungsumgebung oben rechts wieder zur Java-Perspektive zurück wechseln. Sobald Debugging einmal ausgeführt wurde, wird oben rechts einerseits ein Symbol für die Java-Perspektive (J), und andererseits ein Symbol für die Debugging-Perspektive (Käfer) angezeigt.

## Aufgaben

### Auftrag 1

Kopieren Sie das Code-Beispiel und debuggen Sie ihn, wie in der **Schritt für Schritt Anleitung** gezeigt.

Das verwendete Code-Beispiel

```
public class Mini {  
    public static void main(String[] args) {  
        String message = "Hi";  
  
        if (message.equals("Hi")) {  
            System.out.println(message);  
        } else {  
            // nix tun  
        }  
    }  
}
```

### Auftrag 2

Kopieren Sie folgendes Beispiel mit einfachem Methodenaufruf in eine Klasse OutputMethodExample in Eclipse:

```
1 import java.time.LocalDate;  
2 public class OutputMethodExample {  
3     public static void main(String[] args) {  
4         printCurrentDate();  
5     }  
6     public static void printCurrentDate() {  
7         System.out.println("Current date is: " + LocalDate.now());  
8     }  
9 }  
10
```

- Untersuchen Sie den Ablauf mittels Debugging.
- Ein **Breakpoint** muss dabei mindestens auf die **Zeile 4**, mit dem Aufruf der Methode `printCurrentDate`, gesetzt werden.
- Sobald der Debugger dort anhält, muss:
  - **Step Into** gewählt werden, damit der Debugger in die Methode verzweigt.

- Diese Debugging-Session soll aufzeigen, wie der Aufruf einer Methode erfolgt.
- Spielen Sie mit den Möglichkeiten

# Methoden Vertiefung

Jetzt lernen Sie, wie Sie Methoden erstellen können, **denen Sie bestimmte Werte übermitteln** und die ein berechnetes **Ergebnis zurückgeben** können. Zusätzlich werden verschiedene Ansätze betrachtet, wie man sonst noch mit Fehlern umgehen kann.

## Anatomie einer Methode

### Signatur / Kopf `public ...`

Die unten hervorgehobene Zeile einer Methode, in der die Sichtbarkeit, Name, Parameter und der Datentyp des Rückgabewertes deklariert werden, heisst **Kopf** (oder auch **Signatur**) der Methode. In der Signatur wird definiert **wie** die Methode aufgerufen werden muss. Nicht aber, was sie genau macht.

Signatur einer Methode

```
public static int methodenName(int parameter1, String parameter2) {  
    // Methodenkörper  
    return 0;  
}
```

- **Sichtbarkeit:** `public`, `private`, `protected`
- **Art:** `static` oder dynamisch, wenn die Art fehlt
- **Rückgabewert:** `void`, `int`, beliebiger Datentyp
  - `void` besitzt keinen Rückgabewert und braucht kein `return`
  - Wenn ein Rückgabewert angegeben wird, ist `return` am Ende Pflicht!
- **methodenName:** beliebig, sollte in camelCase geschrieben sein
- **Kommaseparierte Parameterliste in Klammern:** `(Datentyp parameter1, Datentyp2 parameter2)`

### Methodenkörper `{ ... }`

Nach dem Kopf/Signatur folgt **zwischen geschweiften Klammern** `{}` der Körper der Methode. Hier wird nun definiert, **was** gemacht werden soll, wenn die Methode aufgerufen wird.

 Im Methodenkörper befindet sich somit **der ausführbare Code** in Java

Methodenkörper, ausführbarer Code

```
public static int methodenName(int parameter1, String parameter2) {  
    int lokaleVariable = parameter1 + 2;  
    return lokaleVariable + " " + parameter2;  
}
```

## Beispiel Methode mit Parameter und Rückgabewert

Nun wollen wir an einem expliziten Beispiel eine Methode analysieren welche über Parameter und Rückgabewert verfügt.

- Wir erstellen eine Methode mit dem Namen `readNumberInRange(int min, int max)` welche vom Benutzer eine Nummer erwartet.
- Die Nummer muss sich in einem gewählten Bereich befinden.
- Wenn eine Nummer eingegeben wird, welche nicht im Bereich ist, soll nochmals nachgefragt werden.
- Die korrekt eingegebene Nummer soll in der Konsole ausgegeben werden.

Explizites Beispiel

```
1 import mytools.StdInput;  
2  
3 public class Beispiel {  
4  
5     public static void main(String[] args) {  
6         int userInput = readNumberInRange(40, 60); // Verwendung  
7         System.out.println("Sie haben " + userInput + " eingegeben!");  
8     }  
9  
10    private static int readNumberInRange(int min, int max) // Kopf / Signatur  
11    {  
12        // Methodenkörper / Methoden-Body  
13        int userInput;  
14        do {  
15            System.out.print("Geben Sie eine Zahl zwischen " + min + " und " + max +  
" ein: ");  
16            userInput = StdInput.readInt();  
17        } while (min > userInput || userInput > max);  
18        return userInput;  
19    }  
20  
21 }
```

### ⌚ Methoden Verwendung - gelb, Linie 6

Die Verwendung befindet sich **immer in einem Methoden-Body**. Dies ist ausführbarer Code.

- In der `main`-Methode auf Linie 6 wird die Methode `readNumberInRange(40, 60);` verwendet/aufgerufen.
- Es wird explizit nach einer Nummer im Bereich zwischen 40 und 60 verlangt.
- die Methode `readNumberInRange` selbst wird auf Zeile 10 definiert.

### Methoden Kopf / Signatur - blau auf Zeile 10

Es wird definiert **wie** eine Methode genau aufgerufen werden muss. Der Methoden-Kopf ist nicht ausführbarer Code, sondern gehört zur Struktur. Im Beispiel werden folgende Merkmale spezifiziert:

- **Sichtbarkeit:** `private`

Da sie ausschliesslich in der Klasse Beispiel verwendet wird, in welcher sie auch implementiert ist, muss sie nicht `public` sein.

- **Art:** `static`

Da die Methode direkt in der statischen Methode `main` aufgerufen wird, muss diese Methode ebenfalls statisch sein.

- **Rückgabewert:** `int`

Da wir einen Integer `int` einlesen wollen, muss die Methode auch einen `int` zurück geben.

- **methodName:** `readNumberInRange`

Der Name "readNumberInRange" fängt klein an und ist sprechend. Er besagt dass eine Nummer in einem bestimmten Bereich eingelesen werden soll.

- **Kommaseparierte Parameterliste in Klammern:** `(int min, int max)`

Mit den Parametern `int min` und `int max` wird der Bereich mitgegeben, in dem sich die einzulesende Nummer befinden muss.

### Methodenkörper - grün, Zeilen 12 - 18

Es wird definiert **was** genau geschehen soll, wenn die Methode aufgerufen wird. Im Methoden-Body befindet sich der ausführbare Code.

- **Linie 13:** Hier wird die lokale Variable `int userInput` initialisiert. In diese solle der eingelesene Wert gespeichert werden.
- **Linie 14 - 17:** in der `do..while` Schleife, wird die Einleselogik definiert. Es wird so lange nachgefragt, bis eine korrekte Eingabe gemacht wurde. Die Schlaufe wird **mindestens ein Mal** ausgeführt.
- **Linie 15:** Hier wird dem Benutzer mitgeteilt, dass er eine Zahl eingeben muss. Dabei wird `min` und `max` durch die Parameter dynamisch bestimmt.
- **Linie 16:** Hier wird mit Hilfe der Methode `mytools.StdInput.readInt` ein Integer von der Konsole eingelesen und in die Variable `userInput` geschrieben.
- **Linie 17:** Am Ende der `do..while` Schlaufe wird die Bedingung definiert, bei welcher die sie nochmals ausgeführt werden soll. Die Bedingung `min > userInput || userInput > max` besagt, wenn der `userInput` kleiner als `min` ODER grösser als `max` ist, die Schlaufe nochmals ausgeführt werden soll.
- **Linie 18:** durch `return` wird der `userInput` von der Methode zurück gegeben.

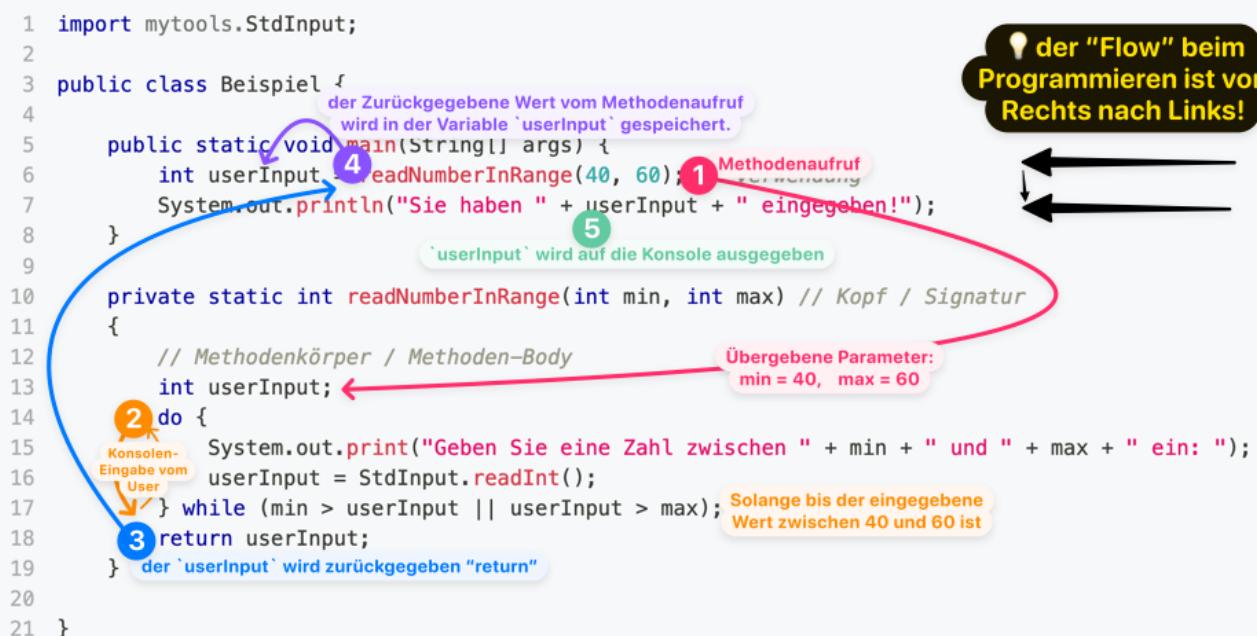
### SORTIERUNG VON METHODEN IN KLASSEN IST EGAL!

Die `main`-Methode ist im Beispiel vor der `readNumberInRange` Methode definiert worden. Die Reihenfolge spielt keine Rolle. Diese könnte auch am Ende definiert werden.

- Die Reihenfolge ist daher egal, da es sich hier um Struktur und nicht ausführbarem Code handelt.
-  **Achtung!** die Reihenfolge vom ausführbaren Code im Methodenkörper spielt natürlich eine Rolle!

## Ablauf eines Programms "Flow"

Hier wird nochmals das selbe Programm dargestellt. Es wird nun aufgezeigt wie das Programm genau durchlaufen wird. In allen Programmiersprachen wird ein Programm immer von **Rechts nach Links, nach Unten**  durchlaufen.



```
1 import mytools.StdInput;
2
3 public class Beispiel {
4     // Zurückgegebene Wert vom Methodenaufruf
4     // wird in der Variable 'userInput' gespeichert.
5     public static void main(String[] args) {
6         int userInput = readNumberInRange(40, 60); 1 Methodenaufruf
7         System.out.println("Sie haben " + userInput + " eingegeben!");
8     }
9
10    private static int readNumberInRange(int min, int max) // Kopf / Signatur
11    {
12        // Methodenkörper / Methoden-Body
13        int userInput; 2 Übergebene Parameter:
14        do {  min = 40, max = 60
15            Konsolen-  System.out.print("Geben Sie eine Zahl zwischen " + min + " und " + max + " ein: ");
16            Eingabe vom  userInput = StdInput.readInt(); 3 Solange bis der eingegebene
17            User       } while (min > userInput || userInput > max); Wert zwischen 40 und 60 ist
18
19        return userInput; 4 der 'userInput' wird zurückgegeben "return"
20    }
21 }
```

**Das Programm startet im Körper der Methode `main` auf Zeile 6:**

1. Da es von Rechts nach Links abläuft wird zuerst die Methode `readNumberInRange(40, 60)` auf **Zeile 6** ausgeführt. Die Methode `readNumberInRange` wird mit den **Parameter min=40 und max=60** ausgeführt.
2. Nun springt das Programm in den Körper der Methode `readNumberInRange`.
  - Zuerst wird auf **Zeile 13** die lokale Variable `int userInput;` deklariert. Sie wird nicht initialisiert, da der Wert direkt vom Benutzer eingelesen wird.
  - dann wird in der `do...while` Schlaufe mit Hilfe von `mytools.StdInput.readInt()` auf **Zeile 16** ein Integer eingelesen und in der Variable `userInput` gespeichert.
  - Auf **Zeile 17** wird geprüft ob `userInput` zwischen `min` und `max` liegt. Hier also zwischen 40 und 60.

- Wurde keine korrekte Zahl eingelesen Springt das Programm wieder zur **Zeile 15**
- 3. Wurde eine korrekte Zahl eingelesen, wird sie nun auf **Zeile 18** zurückgegeben.
- 4. Nun springt das Programm wieder zurück zur **Zeile 6**. Der zurückgegebene Wert wird nun durch den Zuweisungsoperator `=` in die lokale Variable `int userInput` der `main`-Methode gespeichert.
- 5. Als weiteres wird nun die Variable `userInput` auf **Zeile 7** mit einem String verknüpft und durch die Methode `System.out.println` auf die Konsole ausgegeben. Es wird bei einem Methodenaufruf also immer zuerst der Code innerhalb der Klammer `()` ausgeführt.

### ⓘ ZEILEN-FLOW

6, 13, (14, 15, 16, 17), 18, 6, 7

- Wobei die Zeilen (14, 15, 16, 17) mehrmals ausgeführt werden können.
- Die Zeilen 1-5, 8-11, 19-21 werden nie ausgeführt. Sie beinhalten Struktur Informationen, nicht aber ausführbarer Code.

## Geltungsbereich von lokalen Variablen

Lokale Variablen können nur im selben Code-Block (geschweifte Klammern `{ }` ) indem Sie auch deklariert wurden, verwendet werden.

Im oberen Beispiel wird die Variable `int userInput` zwei Mal deklariert.

- Einmal in der Methode `main` auf Zeile 6
- Einmal in der Methode `readNumberInRange` auf Zeile 13.

Auch wenn diese gleich heissen und vom gleichen Typ sind, sind es **zwei eigenständige Variablen**. Sie könnten auch anders heissen.

### ⓘ LOKALE VARIABLE?

Wenn eine Variable **innerhalb einer Methode** initialisiert wird, spricht man von lokalen Variablen. Lokal, da sie nur innerhalb dieser Methode ab der Initialisierung verwendet werden kann (gültig ist).

☞ Es gibt noch Instanz-Variablen, die innerhalb einer gesamten Klasse sichtbar sind. Instanz-Variablen sind in diesem Modul noch nicht relevant!

## Explizites Beispiel

Im unteren Beispiel sind die Variablen `userInput` in `userInputMain` und `userInputRange` umbenannt.

Ebenfalls führen wir noch die Variable `userInputRangeNested` ein. Diese braucht es für die Logik nicht, zeigt aber die Sichtbarkeit innerhalb von geschachtelten Code-Blocks in Methoden.

- Grün wird dargestellt wenn eine Variable sichtbar, also deklariert wird
- Gelb zeigt an wenn die Sichtbarkeit einer Variable endet

- Rot sind Fehlerbeispiele, an diesen Stellen wird versucht auf eine Variable zuzugreifen, welche an dem Punkt nicht sichtbar ist.

### Geltungsbereich von lokalen Variablen

```
1 import mytools.StdInput;
2
3 public class Beispiel {
4
5     public static void main(String[] args) {
6         // ERROR: userInputMain wird erst auf Zeile 9 definiert
7         System.out.println("Hier kann " + userInputMain + " noch nicht verwendet
8             werden!");
9
10        int userInputMain = readNumberInRange(40, 60); // ab hier ist userInputMain
11            sichtbar
12
13        // userInputMain ist sichtbar
14        System.out.println("Sie haben " + userInputMain + " eingegeben!");
15
16        // ERROR: userInputRange ist hier nicht sichtbar!
17        System.out.println("Hier kann " + userInputRange + " nicht verwendet
18            werden!");
19        // hier endet die Sichtbarkeit von userInputMain
20    }
21
22    private static int readNumberInRange(int min, int max)
23    {   // ab hier ist min und max sichtbar
24        int userInputRange; // ab hier ist userInputRange sichtbar
25        do {
26            int userInputRangeNested; // ab hier ist userInputRangeNested sichtbar
27            System.out.print("Geben Sie eine Zahl zwischen " + min + " und " + max +
28                " ein: ");
29            userInputRangeNested = StdInput.readInt(); // userInputRangeNested ist
30            sichtbar
31            userInputRange = userInputRangeNested; // userInputRange ist sichtbar
32            // hier endet die Sichtbarkeit von userInputRangeNested
33        } while (min > userInputRange || userInputRange > max);
34        return userInputRangeNested; // ERROR: userInputRangeNested ist nicht
35            sichtbar!
36        return userInputRange; // userInputRange ist hier sichtbar
37        // hier endet die Sichtbarkeit von userInputRange, min und max
38    }
39
40 }
```

### **GESCHWEIFTE KLAMMERN ZÄHLEN!**

Beim Programmieren muss man immer ein gutes Auge auf die geschweiften Klammern halten.  
Diese sind sehr oft das Problem wenn Eclipse errors anzeigt.

 Es ist guter Stil, wenn die Verschachtelung nicht mehr als 3 Klammern übersteigt.

## **Erklärvideos von Studyflix**

- Java Methoden einfach erklärt
- Parameter von Methoden einfach erklärt

# Fehleranalyse und Fehlersuchen

Um Herausforderungen beim Programmieren zu lösen, müssen Sie erkennen können, welche Meldungen von Eclipse nur Warnings sind, die Sie mindestens zuerst mal nicht beachten müssen.

Dann bleiben **drei Fehlerarten**:

1. solche, die bereits beim eintippen angezeigt werden,
2. Fehler, die beim kompilieren angezeigt werden und anhand des angezeigten Fehlers analysiert werden können
3. Fehler die zur Laufzeit eine "Exception" ergeben, für die ein "Stacktrace" besteht, der einen Link zu einer möglichen Fehlerquelle enthält.

## Warnings (Gelb unterwellt)

Nicht alles was Eclipse anzeigt, sind Fehler. So zeigt Eclipse auch verschiedene Warnings an. Dabei kann ein **gelbes Dreieck mit einem Ausrufezeichen** bei den Zeilennummern angezeigt werden.

Meist ist dann ein Ausdruck mit einer **gelben Schlangenlinie unterstrichen**. Wenn man mit der Maus darüberfährt, wird ein Dialog angezeigt, der es ermöglicht, die Ursache der Warnung zu entfernen.

The screenshot shows a code editor with Java code. Line 7 contains the code `//ln`. A yellow callout box appears over this line, containing the message: "The value of the local variable someValue is not used". Below this, it says "5 quick fixes available:" followed by three options: "Remove 'someValue' and all assignments", "Remove 'someValue', keep assignments with side effects", "@ Add @SuppressWarnings 'unused' to 'someValue'", and "@ Add @SuppressWarnings 'unused' to 'main()'".

```
1
2 public class BugHunt {
3
4     public static void main(String[] args) {
5         int someValue;
6
7         //ln
8
9     }
10
11
12     public s
13 }
```

## Direkt angezeigte Fehler

Bei verschiedenen Fehlern kann Eclipse direkt erkennen, dass ein Fehler vorliegt. So wie hier im Beispiel, wo versucht wird, einen String-Wert in einer int-Variablen zu speichern. Die Fehlerquelle ist mit einer **roten Schlangenlinie** unterstrichen und rechts erscheint neben der Zeilennummer ein **rotes Quadrat mit einem weissen x**. Ob die vorgeschlagene Lösung sinnvoll ist, muss im Einzelfall geprüft werden.

```
1
2 public class BugHunt {
3
4     public static void main(String[] args) {
5         int someValue;
6
7         int anotherValue = "abc"; Type mismatch: cannot convert from String to int
8
9     }
10
11     public static void doSom1 quick fix available:
12         System.out.println("did something..."); Change type of 'anotherValue' to 'String'
13
14     }
15
16 }
17 } Press 'F2' for focus
```

## Kompilierfehler

Werden direkt angezeigte Fehler ignoriert und das Programm wird kompiliert, zeigt Eclipse einen Kompilierfehler an. Die Fehlerursache kann im sogenannten "Stacktrace", der Ausgabe der Fehlermeldung, analysiert werden.

Kompilierfehler sind meistens **Rechtschreibfehler**. Im Beispiel wurde die Variable `anotherValue` falsch geschrieben `anotheValue`. Wird dieser Code nun Kompiliert erscheint folgender Fehler.

- In der ersten Zeile steht «unresolved compilation problem».
- In der vierten Zeile ist die Fehlerquelle angegeben, der Klassenname und die Fehlerzeile.
- Diese Fehlerquelle kann man anklicken, dann springt der Cursor direkt an diese Position.
- Der Fehler wird ebenfalls direkt beim Code (Zeile 9) rot markiert.

The screenshot shows the Eclipse IDE interface. In the top left, there's a code editor window displaying a Java file named 'BugHunt.java'. The code contains several syntax errors, indicated by red squiggly lines under words like 'anotheValue' and 'anotherValue'. Lines 9 and 12 are highlighted in blue. The bottom part of the screenshot shows the 'Problems' view, which displays the following error message:

```
<terminated> BugHunt [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (31.05.2021, 20:38:02 – 20:38:05)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    anotheValue cannot be resolved to a variable
        at BugHunt.main(BugHunt.java:9)
```

## Fehler zur Laufzeit

Laufzeitfehler sind die gängigsten und auch die am schwierigsten zu finden! Fehler zur Laufzeit sind **Inhaltsfehler**. Sie passieren, wenn die Eingabe vom Benutzer nicht korrekt geprüft wurde. Der häufigste Laufzeitfehler ist die `NullPointerException`. Dieser tritt auf, wenn ein Objekt noch nicht initialisiert wurde und somit `null` ist. Wird dann versucht eine Methode auf dem Objekt auszuführen gibt es einen Fehler.

Im Beispiel ist das Problem eine solche `NullPointerException`. Es wird versucht auf Zeile 12 die Länge vom String-Objekt name herauszufinden. Nur wurde die Variable "name" auf der Zeile 10 mit `null` überschrieben. Es wird also wieder ein "Stacktrace" angezeigt die Hinweise zum Fehler ausgeben. Die letzte Zeile der Meldung ist wiederum Klassenname und die Zeile. Es ist wieder ein Link, den man anklicken kann, damit der Cursor direkt an der Fehlerquelle steht.

Diesmal wird jedoch kein Fehler direkt bei der Zeile 12 dargestellt. Eclipse zeigt nur Rechtschreibfehler direkt bei der Zeile an. Inhaltsfehler werden ausschliesslich durch den "Stacktrace" beschrieben.

The screenshot shows a Java code editor with the following code:

```
1 public class BugHunt {
2     public static void main(String[] args) {
3         int someValue;
4         String name = "abc";
5
6         int anotherValue = 456;
7
8         if(true) name = null;
9
10        System.out.println(name.length());
11    }
12
13    public static void doSomethingMethod() {
14        System.out.println("did something...");
15    }
16
17 }
```

The line `System.out.println(name.length());` is highlighted in blue. The code editor has a vertical scrollbar on the right.

Below the code editor is a terminal window showing the execution results:

```
Problems @ Javadoc Declaration Console <terminated> BugHunt [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (31.05.2021, 20:40:16 – 20:40:16)
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "name" is null
at BugHunt.main(BugHunt.java:12)
```

# Aufgaben

## ! UNTERSUCHEN SIE ALLE AUFGABE AUCH MITTELS DEBUGGING

🔍 Verfolgen Sie den Programmablauf und die Werte der Variablen.

👉 Schauen Sie auf welchen Zeilen überhaupt ein "Breakpoint" gesetzt werden kann. Sie werden sehen, dies geht nur auf Zeilen die auch wirklich ausgeführt werden. Also auf Zeilen im Methodenkörper.

## Aufgabe 1 - «Grössere Zahl»

Erstellen Sie ein Programm, das zwei Zahlen von der Konsole einliest.

- Diese beiden Zahlen sollen an eine Methode als Parameter übergeben werden können.
- Die Methode soll die grösste der beiden Zahlen ermitteln und diese als Rückgabewert zurückgeben.

## Auftrag 2 - «Intervall»

Erstellen Sie ein Programm, in dem eine Methode prüft, ob eine Zahl innerhalb eines Intervalls liegt.

- Der Intervall (unterer und oberer Grenzwert) und die Zahl werden als Parameter an die Methode übermittelt.
- Die Methode gibt true zurück, falls die Zahl im Intervall liegt, sonst false.
- Das Ergebnis soll in der `main`-Methode auf die Konsole ausgegeben werden.

## Auftrag 3 «Zinsrechner»

Erstellen Sie ein Programm, wobei Sie einen Sparbetrag eingeben können, sowie einen Zins in %.

- In einer Methode soll entsprechend der Zinsbetrag ausgerechnet und zurückgegeben werden.
- Dieser Zinsbetrag soll dann in der `main`-Methode auf die Konsole ausgegeben werden.

## Auftrag 4 «Einfacher Rechner»

Dieses Programm erwartet die Eingabe zweier Zahlen `a` und `b` durch den Benutzer, sowie den Rechenoperator als String `+` oder `*`.

- Für die Operatoren `+` und `*` sollen zwei Methoden (z.B. `add`, `multiply`) existieren welche die Parameter `a` und `b` besitzen und das Resultat zurückgeben.
- Je nach Operator werden die beiden Zahlen `a` und `b` an die zuständige Methode als Parameter übergeben

- Das Ergebnis wird berechnet und als Rückgabewert an die `main`-Methode zurückgegeben.
- Das Ergebnis soll in die Konsole ausgegeben werden.

## Auftrag 5 «Eigenes Beispiel»

Adaptieren Sie ein Beispiel mit Methoden inkl. Parameter(n) und Rückgabewert auf eine eigene Situation.

## 5 - Schleifen und Wiederholungen

### 🎯 Ziele

- Sie können mit dem while durch eine Bedingung gesteuerte Wiederholungen programmieren.

### 📺 Erklärvideos von Studyflix

- [While Schleife einfach erklärt](#)
- [For Schleife einfach erklärt](#)

## while

Das `while` ist dem `if` ähnlich. Es hat auch eine Bedingung, aber nur einen Code-Block, der solange ausgeführt wird, bis die Bedingung den boolean Wert `false` ergibt.

Das `while` ermöglicht es nun grundsätzlich, dass Programme geschrieben werden können, die nicht nur einmal durchlaufen, sondern Abschnitte resp. Blöcke enthalten, die mehrfach ausgeführt werden können.

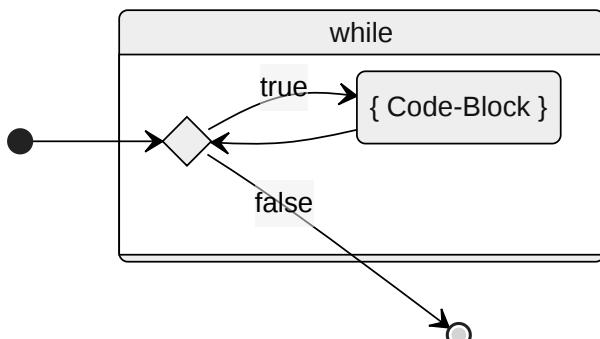
Beim `while` wird das Wissen über den boolean Datentyp, wie es beim `if` angewandt wurde, ebenfalls benötigt. Zentral ist also wieder, dass eine Bedingung formuliert werden kann.

Das grundsätzliche Muster der `while`-Kontrollstruktur sieht folgendermassen aus:

While Code-Beispiel

```
1 int i = 10;
2 while (i > 0) {
3     System.out.println("value of i: " +
4         i);
5     i = i - 1;
```

### Ablaufdiagramm



Der Aufbau der Anweisung ist also so:

1. Es besteht ein Anfangsstatus.
  - Hier eine Variable `int i = 10;`.
2. Es folgt das Schlüsselwort `while` mit einer **booleschen Bedingung** im runden Klammerpaar `()`.
  - Hier `i` grösser als `0` also `(i > 0)`
3. dann folgt zwischen den beiden geschweiften Klammern `{}` der Code- Block, dessen Ausführung durch das `while` kontrolliert wird.
  - Er wird solange ausgeführt bis die Bedingung `false` ergibt.

### Und wie oft wird also obige Anweisung ausgeführt?

- Beim ersten Durchlauf hat `i` den Wert 10 wie auf **Zeile 1** initialisiert.
- Dann wird dieser Wert auf **Zeile 4 um 1 verringert**
- Sobald `i` von 1 nach 0 verringert wird ist die Bedingung nicht mehr erfüllt, da die Bedingung verlangt, dass der Wert von `i` grösser als 0 sein muss.
- Die Schlaufe wird somit 10 Mal durchlaufen.

### ① WHILE

- **Solange die Bedingung zutrifft** wird ein Code-Block ausgeführt.
- Trifft die Bedingung von Anfang an nicht zu, wird der Code-Block auch nie ausgeführt.

## 📝 Aufgaben

### Wiederholte Ausgabe

Schreiben Sie ein Programm, welches 10-mal „Hopp Schwiiiz“ ausgibt.

- Schreiben Sie das Programm **zuerst als Sequenz von 10 Zeilen** mit `System.out.println`.
- Schreiben Sie dann das Programm kürzer und einfacher mit einer `while`-Schleife.

#### Zusatzaufgabe:

Schreiben Sie das Programm mit einer `while`-Schleife und einer Variablen, welche diese steuert, wobei Sie die Variable wie folgt deklarieren:

```
int zaehler = -5;
```

Bis zu welchem Wert muss diese Variable laufen?

### Multiplikationstafel ausgeben

Schreiben Sie ein Programm, welches eine Multiplikationstafel ausgibt.

- Der Benutzer gibt an, für welche Zahl die Multiplikationstafel erstellt werden soll.

Folgende Ausgabe ist erwünscht, falls der Benutzer z.B. 2 eingibt:

```
1 x 2 = 2
2 X 2 = 4
3 X 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12
7 x 2 = 14
8 x 2 = 16
9 x 2 = 18
10 x 2 = 20
```

## 📺 Erklärvideos von Studyflix

- [while-Schleife einfach erklärt](#)

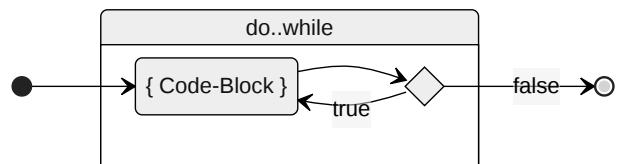
## do { } while

Zusätzlich existiert eine sogenannte "flussgesteuerte" Variante von `while`. Diese Variante **beginnt mit dem Schlüsselwort do**, und das `while` rutscht an den Schluss.

### do { } while Code-Beispiel

```
int x = 10;
do {
    System.out.println("value of x: " +
x);
    x = x-1;
} while(x > 0);
```

### Ablaufdiagramm



Der `do`-Block in den gescheiteten Klammern `{}` wird **immer einmal ausgeführt**, bevor die Bedingung überprüft wird.

Ist die Bedingung erfüllt, wird der do-Block erneut ausgeführt, solange bis die Bedingung `x > 0` nicht mehr erfüllt ist. Die `do..while` Schlaufe ist dadurch unerscheidlich zur `while` Schlaufe, dass der Code-Block immer **mindestens ein Mal ausgeführt wird**.

### ! DO..WHILE

- Der Code-Block wird **immer zuerst einmal ausgeführt**
- Solange die Bedingung zutrifft** wird der Code-Block wiederholt ausgeführt.

## Aufgaben

### Fahrenheit Umrechner

Schreiben Sie ein Programm, welches eine Auflistung der Celsiuswerte von -20 bis 100 Grad in 5er Schritten in Fahrenheit ausgibt.

Die Formel dazu lautet:

```
fahrenheit = (9.0/5.0) * celsius + 32
```

### Beispiel Werte zur Prüfung

	Celsius	Fahrenheit
Gefrierpunkt	0	32

	Celsius	Fahrenheit
Kochpunkt	100	212

## Guthaben aufbrauchen

Schreiben Sie ein Programm, welches zuerst ein Startguthaben von 100.00 Franken in einer Variablen festlegt.

- Nun soll jeweils wiederholt der Benutzer über sein aktuelles Guthaben informiert und nach einem Betrag gefragt werden, den man abheben möchte.
- Dieser Betrag wird so lange abgezogen, bis das Guthaben 0 oder weniger beträgt.
- Am Schluss soll «Sorry, your balance now is zero or below» auf die Konsole ausgegeben werden.

## Passwort-Prüfung

Erstellen Sie ein Programm, welches mittels `do..while` nach dem Passwort "abrakadabra" fragt.

- Solange das Passwort falsch ist, soll die Anforderung wiederholt werden.
- Sobald das Passwort "abrakadabra" richtig eingegeben wurde, soll dem Benutzer die Bestätigung „Logged in!“ angezeigt werden.

### ⓘ INFO

Falls es nicht klappt → if-Blatt lesen → Regeln für String!

# for

Neben `while` und `do..while` existiert noch eine dritte Schleifenart, die `for`-Schleife.

Die `for`-Schleife ist eine Erweiterung der `while` Schleife, sodass zwischen den runden Klammern `()` nicht nur eine `<Bedingung>` definiert wird, sondern auch eine `<Zählervariable>` sowie eine `<Schrittweite>`.

- 💡 Die drei Teile werden mit einem `;` Semikolon getrennt.
- 💡 Für die `for`-Schleife existiert **keine** `do...for` Variante.

## Grundform

Die for Kontrollstruktur ist zuerst einmal gewöhnungsbedürftig. Das Muster sieht so aus:

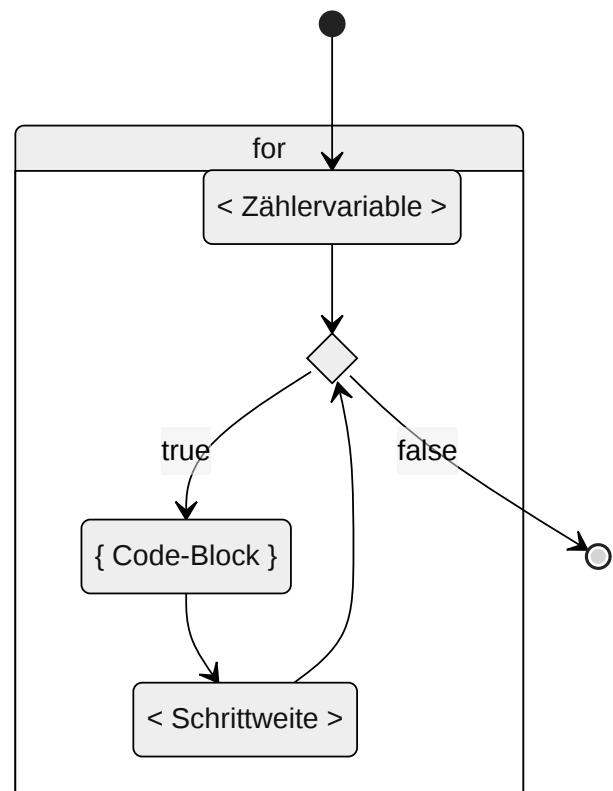
### for - Muster

```
for (
    <Zählervariable>;
    <Bedingung>;
    <Schrittweite>
) {
    <Anweisungen, Sequenz>
}
```

#### ⚠ SEMIKOLON

Die 3 Teile in den Runden Klammern `()` sind mit einem Semikolon `;` getrennt! (und stehen meist auf einer Linie nebeneinander)

### for - Ablaufdiagramm



`<Zählervariable>`

**Ganz am Anfang** wird die `<Zählervariable>` initialisiert und ist im Code-Block sichtbar.

`<Bedingung>`

**Vor jeder Sequenz** wird die Bedingung auf Wahrheit geprüft (`true`), ob die die `<Anweisungen, Sequenz>` ausgeführt werden sollen oder ob die Schlaufe beendet wird.

`<Schrittweite>`

**Am Schluss einer Sequenz**, vor der `<Bedingung>`, wird die `<Zählervariable>` hoch oder runter gezählt.

💡 `i++` Zählt z.B. die Variable `i` mit der Schrittweite 1 hoch.

💡 Häufig dient die `<Zählervariable>` als Indexnummer und heisst dadurch `i`.

💡 Meistens betrifft die `<Bedingung>` die `<Zählervariable>`.

#### `<Anweisungen, Sequenz>`

Die `<Anweisungen>` im Code-Block werden ausgeführt, solange die `<Bedingung>` wahr (`true`) ist.

## Grundform am Beispiel

Damit es weniger abstrakt ist, hier ein Beispiel als `for` sowie `while` Schleife, welches die Zahlen 0 bis 4 ausgibt. Dies soll veranschaulichen, wieso die `for`-Schleife zusätzlich zur `while`-Schlaufe in fast jeder Programmiersprache existiert.

### for

for: Wiederhole 5 mal fix

```
public class FixeWiederholung {  
  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

Es wird als Erstes die Variable `int i = 0;` initialisiert. Sobald die Anweisungen innerhalb des Code-Blocks abgearbeitet sind (`System.out.println(i);`), springt die Programmausführung bei der schliessenden geschweiften Klammer wieder zurück zum `for`, verändert die Variable um die Schrittweite (`i++`) und prüft, ob die Bedingung (`i < 5`) noch erfüllt ist. Wenn ja, wird die Schleife erneut ausgeführt.

### while

while: Wiederhole 5 mal fix

```
public class FixeWiederholungWhile {  
  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 5) {  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

Die `while` Variante verhält sie gleich wie die `for` Variante. Sie braucht jedoch mehr Zeilen.

### ⚠️ WIESO WIRD `for` HIER BEVORZUGT?

`int i = 0;` ist außerhalb der Schleife initialisiert!

Dadurch ist diese also in der ganzen äusseren Methode sichtbar und nicht nur im Code-Block der Schleife und existiert auch nach dem Beenden der Schleife weiter.

#### `<Zählervariable>`

#### `<Bedingung>`

#### `<Schrittweite>`

int i = 0: Eine Variable *i* mit  
Datentyp int wird mit 0  
initialisiert

i < 5: Solange *i* kleiner als 5  
ist, wird die for-Schleife  
weiterhin ausgeführt

i++: Die Variable *i* wird bei  
jedem Durchlauf um 1 erhöht

### <Anweisungen, Sequenz>

Mit der Anweisung `System.out.println(i);` wird bei jeder Sequenz/Iteration bei der die <Bedingung> wahr ist, die Variable *i* auf die Konsole ausgegeben.

## Programmablauf

Eine for-Schleife wird so durchlaufen, wie folgend dargestellt. Wenn die Bedingung nicht mehr erfüllt ist, wird die Schleife beendet und es wird die erste Anweisung unterhalb der Schleife ausgeführt (roter Pfeil).

```
public class FixeWiederholung {  
  
    public static void main(String[] args) {  
  
        1 for (int i = 0; i < 5; i++) {  
            2 true 3 System.out.println(i); 4 false  
            5 }  
    }  
}
```

1. Die Bearbeitung der for-Schleife beginnt beim roten Pfeil. Es wird im ersten Teil die <Zählervariable> *i* mit 0 initialisiert.
2. Im zweiten Teil, der <Bedingung>, wird geprüft ob die Zählervariable *i* kleiner als 5 ist.
3. Da dies der Fall ist, wird nun die <Anweisungen> (`System.out.println(i);`) im Code-Block `{}` ausgeführt.
4. Danach wird "nach oben" zum letzten Teil, der <Schrittweite> gesprungen. Hier wird durch `i++` die <Zählervariable> *i* um 1 hochgezählt.
5. Nun wird wieder zur <Bedingung> gesprungen.
6. Wenn die Variable *i* immer noch kleiner als 5 ist, wird die <Anweisung> wiederholt ausgeführt.

Die Punkte 3-6 (grün), werden solange wiederholt, bis die <Bedingung> falsch (`false`) und somit die Variable *i* mindestens 5 ist. Im oberen Beispiel wird das nach der 5. Wiederholung der Fall sein.

### Ausgabe vom oberen Beispiel

```
0  
1  
2  
3  
4
```

## Aufgaben

### Auftrag 1

1. Geben Sie die Zahlen von 1 bis 10 aus.
2. Geben Sie die Zahlen von 37 bis 55 aus.
3. Geben Sie jede zweite Zahl von 18 bis 96 aus.
4. Fragen Sie den Benutzer nach Startwert, Endwert und Schrittweite und geben Sie die entsprechenden Zahlen aus.

### Auftrag 2a

- Schreiben Sie ein Programm, das eine Zeile mit 10 Sternen ausgibt.
- Die Sterne sollen einzeln in einer for-Schleife ausgegeben werden (also bei jedem Durchlauf der Schleife wird ein Stern der Zeile hinzugefügt).
- Verwenden Sie dazu die Methode `System.out.print` (und nicht `System.out.println`). Die Ausgabe sieht also so aus:

Ausgabe

\*\*\*\*\*

### Auftrag 2b

Erweitern Sie das obige Programm so, dass 5 Zeilen mit jeweils 10 Sternen ausgegeben werden. Die 5 Zeilen müssen auch in einer for-Schleife erstellt werden. Die Ausgabe sollte also so aussehen :

Ausgabe

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*

### Auftrag 3

Schreiben Sie ein Programm Flaggen, das folgende Anforderungen erfüllt:

- die Anzahl Zeilen für die Flaggen werden vom Benutzer angegeben
- Zeile 1 enthält 1 Stern, Zeile n enthält n Sterne Die Ausgabe soll so aussehen:

### Ausgabe

```
Flaggen 1
Anzahl Zeilen: 5
*
**
***
****
*****
*****
```

## Auftrag 4

Erweitern Sie das Programm so, dass der Benutzer das Zeichen, mit dem die Flagge gezeichnet wird, selbst festlegen kann.

Die Ausgabe soll so aussehen:

### Ausgabe

```
Flaggen 2
Anzahl Zeilen: 5
Zeichen: x
x
xx
xxx
xxxx
xxxxx
```

## Auftrag 5

Erweitern Sie das Programm so, dass der Benutzer eingeben kann, wie viele Flaggen gezeichnet werden.

Die Ausgabe soll so aussehen:

## Ausgabe

Flaggen 3

Anzahl Zeilen: 5

Anzahl Flaggen: 2

Zeichen: @

Flagge 1

```
@  
@@  
@@@  
@@@@  
@@@@@
```

Flagge 2

```
@  
@@  
@@@  
@@@@  
@@@@@
```

## 📺 Erklärvideos von Studiflix

- [for-Schleife einfach erklärt](#)

## 6 - Spiele

Spiele brauchen immer den Zufall. Ohne den Zufall wären Sie langweilig. In diesem Themenblock lernen Sie, wie Sie zufällige Zahlen erzeugen, und damit ein einfaches Spiel programmieren können.

### ⌚ Ziele

- Sie können in Java zufällige Zahlenwerte in einem vorgegebenen Bereich erzeugen.
- Sie kennen den Grundmechanismus für ein Game und können damit einfach Spiele programmieren.

### 🎲 Erste einfache Spiele

Sie können in Java-Programmen nun mit Bedingungen umgehen und Wiederholungen ausführen. Damit kann man schon einiges machen. Beispielsweise lassen sich damit kleine Ratespiele programmieren, wie im nachfolgenden Auftrag.

### 📺 Erklärvideos vom Studiflix

- <https://studyflix.de/informatik/zufallszahlen-237>

# Ratespiel

Nun erstellen wir ein Ratespiel. Um einfache Rate-Games zu erstellen, ist es nötig, dass zufällige Zahlen zur Verfügung stehen. Daher werden wir zuerst erkunden wie dies genau geht.

## Zufall erzeugen

In Java gibt es eine Klasse Math. Auf dieser kann man beispielsweise den Befehl `Math.random()` aufrufen, dann bekommt man eine Zahl zwischen 0 inklusive und 1 (exklusive) zurück. Wenn ein Programm simulieren soll, dass ein Würfel geworfen wurde, und folglich die Zahlen 1 inklusive bis 6 inklusive zufällig ausgewählt werden sollen, kann dies mit der folgenden Zeile erfolgen:

6er Würfel

```
int diceValue = (int) (Math.random() * 6) + 1;
```

Die allgemeine Formel für eine Zufallszahl lautet:

Generelle Zufallszahl Formel

```
public static int randomNumberInRange(int min, int max) {  
    return (int)(Math.random() * (max - min + 1)) + min;  
}
```

**A ACHTEN SIE DARAUF, DIE KLAMMERN KORREKT ZU SETZEN.**

## Auftrag

Erstellen Sie das Programm namens `GuessGame`.

Dieses Programm:

- erzeugt eine zufällige Zahl zwischen 1 und 100 (wie das gemacht wird, ist oben beschrieben)
  - das ist die zu erratende geheime Zahl
- fordert den Benutzer auf, die Zahl zu erraten, liest die Eingabe des Benutzers ein und vergleicht sie mit der geheimen Zahl
  - das wird solange wiederholt, bis die geheime Zahl erraten wird
- zum Schluss wird dem Benutzer zum Erfolg gratuliert

Für den Benutzer ist das nicht einfach. Bauen Sie deshalb schrittweise die folgenden Erweiterungen in das Programm ein:

1. Geben Sie dem Benutzer einen Hinweis, ob die Zahl, die er erraten soll, kleiner oder grösser als die von ihm geratene Zahl ist.
2. Bauen Sie als Cheat die Ausgabe der geheimen Zahl gleich am Anfang ein, um das Programm rascher testen zu können.
3. Zählen Sie die Anzahl Versuche, die der Benutzer dazu braucht und geben Sie dies am Ende auf die Konsole aus.

# Reaktionsspiel

## Aktuelle Zeit

Die aktuelle Zeit kann mit folgendem Code in Millisekunden seit dem 01.01.1970 ausgelesen werden:

```
long time = System.currentTimeMillis();
```

Die Differenz zwischen zwei Zeiten ergibt somit die Zeitspanne zwischen zwei Messungen in Millisekunden.

```
long time1 = System.currentTimeMillis();
// some time
long time2 = System.currentTimeMillis();
long reaction = time2 - time1;
```

## Auftrag

Erstellen Sie ein Programm namens `ReactionGame`.

Dieses Programm:

- Begrüßt den Spieler
- Definiert dabei, welche Zahl, welchem Buchstaben entspricht:

```
Welcome to the reaction game!
Hit as fast as possible according to the following map:
```

```
0 => a
1 => s
2 => d
3 => f
```

- Danach wird zufällig eine Zahl zwischen 0 und 3 generiert und ausgegeben:

```
Please immediately response to:
```

```
2
```

- Der Benutzer muss nun so schnell wie möglich eine `d` eingeben und `Enter` drücken
- Dies wird 10x wiederholt
- Zum Schluss erhält der Spieler seine Reaktionszeit in Sekunden auf der Konsole

Your score: 1.6265 seconds

- Gibt der Benutzer einen falschen Buchstaben ein, zählt dies als Versuch, wird aber in der Berechnung der Reaktionszeit ignoriert.

### Komplette Beispieldausgabe:

Welcome to the reaction game!

Hit as fast as possible according to the following map:

```
0 => a  
1 => s  
2 => d  
3 => f
```

Please immediately response to: 1

s

Please immediately response to: 2

d

Please immediately response to: 1

s

Please immediately response to: 2

d

Please immediately response to: 1

s

Please immediately response to: 1

s

Please immediately response to: 2

d

Please immediately response to: 1

s

Please immediately response to: 3

f

Please immediately response to: 2

d

Your score: 1.6265

# Würfelspiel

## 🧠 Zufall erzeugen

Die allgemeine Formel für eine Zufallszahl lautet:

Generelle Zufallszahl Formel

```
public int randomNumberInRange(int min, int max) {  
    return (int)(Math.random() * (max - min + 1)) + min;  
}
```

## 📝 Einspieler Variante

Erstellen Sie das Programm namens `DiceGame`. Dieses Programm:

- Wird zuerst nur für einen Spieler implementiert.
- Der Benutzer gibt zu Beginn ein Maximum an Punkten an
- Danach wird für den Spieler so lange gewürfelt, bis er das Maximum erreicht oder über-schritten hat. Das Würfeln an sich soll in eine Methode ausgelagert werden. Der Rückgabewert ist die gewürfelte Zahl.
- Die einzelnen Würfe sollen auf der Konsole dargestellt werden.
- Zum Schluss soll auch ausgegeben werden, wie viele Würfe benötigt wurden, um auf das Maximum zu kommen

### Beispieldaten:

```
Enter the maximum of points: 100  
You rolled: 6 3 4 6 1 2 4 1 6 3 1 5 4 4 4 6 5 6 2 6 4 6 4 5 5  
Total points: 103, number of dice: 25
```

## 📝 Zweispieler Variante

Sie haben nun das Programm für einen Spieler erstellt. Kopieren Sie nun die Klasse `DiceGame` und benennen Sie die Kopie `DiceGame2Player`.

Programmieren Sie es nun so um, dass:

- zwei Spieler daran teilnehmen können.
- Sie treten nacheinander an.
- Zuerst versucht also Spieler 1 auf das Maximum zu kommen, danach Spieler 2.
- Lagern Sie den Spielmechanismus in eine Methode aus.

- Die Methode erhält als Parameter das Punktemaximum und liefert als Rückgabewert die benötigte Anzahl Würfe, um das Maximum zu erreichen.
- Es gewinnt derjenige, der das Maximum mit weniger Würfen erreichen konnte.

### Beispieldausgabe:

```
Enter the maximum of points: 100
1 3 2 1 6 4 3 3 1 5 2 5 4 6 5 4 3 1 4 2 4 3 3 1 1 3 2 1 3 3 2 3 4 2
1 3 4 6 6 4 5 6 6 4 2 6 4 5 5 3 2 3 1 4 1 4 1 1 6 1 2 2 6
Number of dice Player 1: 34
Number of dice Player 2: 29
Player 2 wins.
```

## Zusatz Variante

Machen Sie wiederum eine Kopie von `DiceGame2Player` und benennen Sie die Kopie `DiceGameVariante`.

Der Benutzer gibt zu Beginn z.B. nicht ein Punktemaximum an, sondern **definiert die Anzahl Würfe**, welche die Spieler würfeln sollen. Derjenigen mit der höheren Punktzahl gewinnt zum Schluss das Spiel.

## 7 - Arrays und for(each)

### ⌚ Ziele

- Sie können mittels Arrays effizient mit mehreren Werte des gleichen Datentyps programmieren.
- Sie können die for Kontrollstruktur nutzen, die besonders für das Durchlaufen von Arrays hilfreich ist.

### Einleitung

Wir haben Datentypen kennengelernt und wissen, wie wir Variablen anlegen können. Damit können wir Werte zwischenspeichern und später wieder darauf zurückgreifen.

Stellen Sie sich nun einmal vor, Sie schreiben ein Programm, mit dem Sie Messwerte erfassen. Alle Messwerte haben den gleichen Datentyp. Wenn wir annehmen, dass Sie 100 solche Messwerte speichern möchten, dann könnten Sie das so machen:

```
int wert1;  
int wert2;  
int wert3;  
// immer weiter so  
int wert100;
```

Das wäre sicher möglich, aber sehr aufwendig und nicht gut umsetzbar. Möchten Sie jetzt die Messwerte auslesen (über eine **fiktive Messmethode** `mess_methode`), dann könnte das so aussehen:

```
wert1 = mess_methode();  
wert2 = mess_methode();  
wert3 = mess_methode();  
// und so weiter  
wert100 = mess_methode();
```

Wenn Sie jetzt zum Beispiel den Durchschnitt der Messwerte ausrechnen möchten, dann schreiben Sie einen Code wie diesen:

```
int summe = wert1 + wert2 + wert3 + /* ... */ + wert100;  
int durchschnitt = summe / 100;
```

Soll das Programm jetzt noch so erweitert werden, dass Sie mehr Messwerte erfassen können, zum Beispiel 1000, dann läuft es endgültig aus dem Ruder.

Wie man das Program trotzdem zähmen kann, erfahren Sie im nächsten Kapitel.

## 📺 Erklärvideos von Studyflix

- [Array einfach erklärt](#)
- [for-Schleife einfach erklärt](#)
- [foreach-Schleife einfach erklärt](#)

# Arrays / Listen

Programmierer versuchen immer so wenig wie möglich Code schreiben zu müssen - ja gute Programmierer sind faul 😊. Daher gibt es genau für das in der Einleitung beschriebene Problem das Konzept der Arrays, also Auflistungen.

## ⚠️ ARRAY -> AUFLISTUNG

**Eine Variable mit einer Auflistung von Werten des gleichen Typs.**

- Wenn wir viele Daten erfassen möchten, die alle den **gleichen Datentyp** besitzen und auch zusammengehören, dann können wir mit Arrays arbeiten.
- Arrays besitzen eine **vordefinierte Länge**. Diese kann nicht einfach so geändert werden.
  - Es gibt in weiteren Modulen Listen mit dynamischer Länge. Dies ist in diesem Modul jedoch noch nicht relevant!

## Array initialisieren

Eine Array-Variable kann auf folgende Weise angelegt werden:

Array initialisieren

```
int[] werte = new int[100];
//           ^^^ Array-Länge      (hier 100)
//           ^^^ ^^^^^^ Initialisierung (immer mit new)
// ^^^^^^ ^^^ Definition int-Array      (eckige Klammern)
```

Zuerst wird durch `int[] werte` eine Variable vom Datentyp `int` definiert welche als Array mehrere Werte fassen kann. Danach wird durch `= new int[100]` der Array initialisiert. Beim initialisieren wird festgelegt, dass der Array 100 Werte fassen kann. Diese Länge ist fest und kann nicht mehr geändert werden.

## ⚠️ CAUTION

Alle weiteren Code-Beispiele beziehen sich immer auf oberen Array `werte`

## Array Werte-Zuweisung

Der Zugriff auf eine Wert im Array geschieht wie bei jeder Variable über den Variablenamen. Da ein Array nun mehrere Werte gespeichert hat, benötigt es zusätzlich die Stelle an der der Wert in der Liste steht. Die Zuweisung geschieht nun über den Variablenamen und danach, in eckigen Klammern, die Stelle/Index des Wertes, den man ansprechen möchte.

Und die sieht wie folgt aus:

Array-Zuweisung durch index in eckiger Klammer

```
werte[0] = 3;  
//      ^^^ Durch eckige Klammern und einem index wird auf  
//      einen beliebigen Wert im Array zugegriffen.  
//      (hier 0, also der erste Wert)
```

Nun könnte man die Messwerte analog der [Einleitung](#) folgendermassen aufnehmen:

Array manuell initialisieren

```
werte[0] = mess_methode();  
werte[1] = mess_methode();  
werte[2] = mess_methode();  
// und so weiter  
werte[99] = mess_methode();
```

Diese Variante ist nun jedoch nicht weniger Schreibintensiv als mit die Version mit den 100 eigenen Variablen der Einleitung. Der Vorteil liegt nun aber darin, dass nur noch eine Variable definiert werden muss.

### **for-SCHLEIFEN**

Schreibaule Programmierer könnten nun auf die Idee kommen, dass Zahlen von 0-99 durch die bereits bekannte **for**-Schlaufe automatisiert erstellt werden könnten!

Und Sie haben recht! Dies geht nun folgendermassen: ✎ **Akra-kadabara-simsalabim aus 100 Zeilen, da mach 3!**

Abfüllen eines Arrays mit Hilfe einer for-Schleife

```
1 for (int index = 0; index < 100; i++) { // 100 ist die Array Länge  
2     werte[index] = mess_methode(); // die Variable index von 0-99  
3 }
```

In diesem Beispiel wird durch eine **for**-Schleife 100 Mal den Code-Block auf Zeile 3 ausgeführt. Dabei wird der Variable **index** im Einerschritt (**i++**) die Werte 0 bis 99 zugewiesen (**index < 100 ist maximal 99**). Die Methode **mess\_methode()** wird somit 100 Mal ausgeführt und der Rückgabewert jedes Mal einer anderen Stelle des Arrays **werte** zugewiesen.

### ⚠ ARRAYS STARTEN BEI 0

Beachten Sie, dass das erste Element des Arrays den Index 0 hat. Entsprechend hat das letzte Element als Index eine Zahl, die um 1 kleiner ist als die Anzahl Elemente.

## Array Zugriff

Der Zugriff auf Werte eines Arrays funktioniert analog der Zuweisung. Nur braucht es dafür keinen Zuweisungsoperator `=`. Dieser erfolgt folgendermassen:

### Array-Zugriff

```
System.out.println(werte[0]); // Gibt den ersten Wert auf die Konsole aus
```

Wenn wir jetzt den Durchschnitt aller Messwerte berechnen möchten, können wir nun nochmals die `for`-Schleife zur Hilfe nehmen:

### Summe aller Werte des int-Arrays werte

```
int summe = 0;
for (int i = 0; i < 100; i++) {
    summe += werte[i];
}
int durchschnitt = summe / 100;
```

Jeder Array besitzt das Attribut `.length` in welchem seine genaue Länge abgerufen werden kann. Dies hilft um nicht immer die Länge nachzuschauen zu müssen. Auch ist es besser lesbar. Dies können wir uns zu Nutzen machen um den Code noch genereller zu schreiben:

### Summe aller Werte optimiert mit `.length`

```
int summe = 0;
for (int i = 0; i < werte.length; i++) {
    summe += werte[i];
}
int durchschnitt = summe / werte.length;
```

### ⚠ MAGISCHE NUMMERN VERMEIDEN

Zahlen, die nicht klar ersichtlich sind, wofür sie stehen, werden auch magische Nummern genannt und sollen vermieden werden. Sie machen den Code unübersichtlich und führen zu Fehlern.

- Hier ist die Zahl 100 eine Magische Nummer ✎
- `werte.length` sagt klar aus, dass es sich um die Länge vom Array `werte` handelt. 100 kann vieles sein. z.B. 100 🤔

## Zugriff mit `for(each)`

Wenn man von einem Array lediglich alle Werte auslesen möchte, nicht aber schreiben, gibt es eine spezielle Variante der `for`-Schleife. Die sogenannte `foreach`-Schleife (Für jedes Element). In dieser Variante existiert keine Zählervariable, stattdessen wird bei jeder Iteration der nächste Wert direkt in eine Variable geschrieben. Dies macht das Auslesen eines gesamten Arrays um einiges einfacher.

Summe aller Werte optimiert mit `foreach`

```
int summe = 0;
for (int wert : werte) {
    //           ^^^^^^ Array welcher ausgelesen werden soll
    //           ^ Ein Doppelpunkt trennt die Variable pro Iteration und der Array
    //           ^^^^^^^^^ Die Variable welche pro Iteration den nächsten Wert beinhaltet
    summe += wert;
}
int durchschnitt = summe / werte.length;
```

Mit dieser Variante kann also nicht auf ein individuelles Element zugegriffen werden. Auch kann nicht ein Element neu zugewiesen werden. Dafür benötigt es viel weniger Zeichen und ist somit weniger Komplex und lesbarer.

### ❗ ECKIGE KLAMMERN EIN-MAL-EINS

- `datentyp[]` : Definition eines Arrays
- `new datentyp[100]` : Initialisierung und Definition der Array-Länge
- `variablenamen[n]` : Zugriff auf den n-ten Wert im Array.
  - `variablenamen[0]` : Zugriff auf den 1ten Wert im Array.
  - Ein Array beginnt immer bei 0!

## Generische Durchschnitt-Methode

Mit Hilfe des Attributes `.length` könnten wir nun eine generelle Methode definieren, welche den Durchschnitt aller Einträge eines beliebigen `int`-Arrays berechnet.

### Generische Summen-Methode

```
public static int durchschnitt(int[] liste) {  
    int summe = 0;  
    for (int eintrag : liste) {  
        summe += eintrag;  
    }  
    return summe / liste.length;  
}
```

Methode durchschnitt verwendet mit dem Array werte

```
int durchschnitt = durchschnitt(werte);
```

#### GENERALISIERUNG

- Tritt etwas **häuft** vor, ist es elegant eine generalisierte Methode dafür zu finden.
- Tritt etwas aber **nur 1 Mal** vor ist dies nicht immer nötig!
  - Zu frühes Generalisieren kann zu komplizierterem Code führen ⚽
  - Hier ist das obere Beispiel mit der generischen Durchschnitt-Methode komplexer (mehr Code). Hätten wir ein Programm das mehrfach int-Arrays summieren muss, wäre es weniger komplex.
- **Aber Achtung**, wird eine Methode zu gross ist es übersichtlicher sie aufzusplitten (Programmieren braucht Erfahrung, die erhält man nur beim machen)

## Eine Variante für das Anlegen und die Initialisierung

Es ist auch möglich, einen Array anzulegen und gleich mit Werten abzufüllen. Diese Variante funktioniert so:

```
int[] zahlen = { 1, 8, 9, 3 };
```

Sie sehen, dass es kein new gibt, sondern nur gleich eine Aufzählung der Werte in dem Array. Die Grösse wird auch nicht explizit angegeben. Sie ergibt sich ganz einfach aus der Anzahl Elemente, die angegeben werden.

Diese Variante wird eher für kleinere Arrays verwendet, was sicher noch nachvollziehbar ist. Es gibt aber keine Limitierung von der Sprachdefinition her.

Hier ein kleines Beispiel, wie so ein Array angelegt und dann ausgegeben wird.

```
int[] zahlen = { 1, 8, 9, 3 };
for(int i = 0; i < zahlen.length; i++){
    System.out.println(i + ": " + zahlen[i]);
}
```

## Array sortieren

Wenn wir die Daten in einem Array sortieren möchten, dann müssen wir das nicht selbst programmieren. Es gibt eine Methode, die das für uns übernimmt. Sie heisst (wenig überraschend) sort.

```
int[] zahlen = { 1, 8, 9, 3 };
Arrays.sort(zahlen);
for(int i = 0; i < zahlen.length; i++){
    System.out.println(i + ": " + zahlen[i]);
}
```

Der Datentyp der Werte im Array spielt dabei keine Rolle. Der Aufruf funktioniert auch bei einem String-Array.

```
String[] worte = { "eins", "zwei", "drei" };
for(int i = 0; i < worte.length; i++){
    System.out.println(i + ": " + worte[i]);
}
Arrays.sort(worte);
for(int i = 0; i < worte.length; i++){
    System.out.println(i + ": " + worte[i]);
}
```

## Aufgaben

### Array Chars

Verwenden Sie eine for-Schleife, um den folgenden Array auszugeben:

```
char[] text = {'h','a','l','l','o',' ', 'w','e','l','t'};
```

- Geben Sie danach den Array in **umgekehrter Reihenfolge** aus, also von hinten nach vorne.
- Verwenden Sie dafür auch wieder eine for-Schleife.

### String sortieren

1. Kopieren Sie den untenstehenden Code in ein Eclipse-Projekt und lassen Sie ihn laufen.
2. Erweitern Sie das Programm so, dass der Array sortiert und danach wieder ausgegeben wird.

```
public class SortString {  
  
    public static void main (String[] args) {  
        String beruf[] = {  
            "Wurstmacher",  
            "Holzbearbeiter",  
            "Laufbursche",  
            "Taxifahrer",  
            "Velokurier",  
            "Wasserfahrer",  
            "Zitronenfalter",  
            "Fensterputzer",  
            "Halsabschneider",  
        }  
        //Ausgabe aller Werte des Arrays  
        for (int i = 0; i < beruf.length; i++){  
            System.out.println(i + ". " + beruf[i]);  
        }  
        // Erweitern Sie hier das Programm.  
    }  
}
```

## Array negativ

Gegeben ist Array `a`. Geben Sie alle Elemente mit einem negativen Wert untereinander auf der Konsole aus. Verwenden Sie auch wieder eine for-Schleife.

```
int[] a = { 1, -2, -25, 6, -3, 5 };
```

## Add Arrays

Gegeben sind drei Arrays `a`, `b` und `c`. Verändern Sie durch Java-Code den Array `c` so, dass im Element `c[0]` die Summe `a[0]` plus `b[0]` steht und entsprechend für die Elemente 1, 2, 3.

```
int[] a = { 1, 2, 25, 6 }; // { 1, 2, 25, 6 }  
int[] b = { 9, 18, 5, 34 }; // + { 9, 18, 5, 34 }  
int[] c = new int[4]; // = { 10, 20, 30, 40 }
```

## Combine Arrays

Gegeben sind zwei Arrays `a` und `b`. Erzeugen Sie einen neuen Array `c`, der so lang ist wie `a` und `b` zusammengenommen und auch die Werte von `a` und `b` (in dieser Reihenfolge) enthält.

```
int[] a = { 1, 2, 25 };  
int[] b = { 9, 18 };
```

## Combine Arrays with Methods

Verwende den Code von [Combine Arrays](#).

Nun versuche die Aufgabe so zu lösen, dass das Kopieren und Ausgeben in zwei Methoden gemacht wird. Die Signatur der Methoden sind folgende:

- `public static void copyArray(int[] source, int[] destination, int destination_index)`
- `public static void showArray(String name, int[] array)`

### Noten

Erstellen Sie ein Programm, welches Noten einlesen kann und in einem Array speichert.

- Zuerst soll es nach der Anzahl Noten fragen...
  - ... und dann mit diesem Wert einen neuen Array anlegen.
- Dann sollen alle Werte vom Benutzer eingegeben werden, wobei...
  - ... nach jedem Wert der Durchschnitt der bisher eingegebenen Noten berechnet und angezeigt wird.



Die Arbeit mit Methoden kann Ihr Programm übersichtlicher machen.

## Dice

1. Erstellen Sie einen Array mit sechs 0en.
2. Würfeln Sie 100 mal mit der Methode `rollDice()`
  - Bei jedem Wurf zählen Sie für die gewürfelte Zahl der entsprechende index im Array hoch.
3. Geben Sie aus wie häufig die jeweiligen Zahlen gewürfelt wurden.

```
public class Dice {  
  
    public static void main(String[] args) {  
        // die Aufgabe hier implementieren  
    }  
  
    public static int rollDice() {  
        return (int)(Math.random() * (6 - 1 + 1)) + 1;  
    }  
}
```

## 📺 Erklärvideos von Studyflix

- <https://studyflix.de/informatik/java-array-1898>

# foreach

Arrays, bzw. allgemein Collections, können auch mit der `foreach` Schleife durchlaufen werden. Die Schleife hat **keinen Zähler**, wie die klassische `for`-Schleife.

Ist man jedoch hauptsächlich am **Inhalt der Auflistung** interessiert, kann so sehr einfach darauf zugegriffen werden. Mit dieser Variante kann jedoch **keine Änderung des Arrays** gemacht werden.

Schema

```
for(Datentyp variable : Array) {  
    // code block  
}
```

Code-Beispiel

```
String[] worte = { "hallo", "liebe",  
                  "Welt" }  
for(String wort : worte) {  
    System.out.println(wort);  
}
```

## 📺 Video über Foreach mit Array

foreach

```
for(int Zahl : Zahlen){  
    System.out.println(Zahl);  
}
```

int[] Zahlen = new int[5];

0	38
1	14
2	8
3	23
4	2

## 📺 Erklärvideos von Studyflix

- foreach-Schleife einfach erklärt

## 8 - Repetition und Übungen

Auf dieser Seite finden Sie diverse Aufträge, welche Repetitionen des erarbeiteten und erlernten Stoffes anbieten, sowie Übungen dazu bieten. Entsprechend sind in den unterschiedlichen Aufträgen jeweils verschiedene Wissenselemente kombiniert.

Für die Übung mit den Puzzles finden Sie sowohl die Aufgaben wie auch Lösungen am Schluss der Seite.

# Puzzles

Unter dem Namen "Puzzles" sind viele kleine Aufgaben zusammengestellt, wobei die **Aufgabe direkt im Kommentar der Klasse** gestellt wird.

## ! HIER MÜSSEN SIE ALSO IMMER:

1. zuerst die Aufgabe kopieren
2. danach die Methode `puzzles` so erweitern, dass Sie die Aufgabe vom Kommentar löst.

## Beispiel

Erkennen Sie eine Lösung für die Aufgabenstellung, die im Kommentarblock (Zeilen 4-10) des folgenden Programms formuliert ist?

```
1 public class Puzzle0001 {  
2  
3     /*  
4      * find an implementation for the method `puzzle` that produces the following  
5      * outputs:  
6      *  
7      *      -459  
8      *      -3  
9      *      -17  
10     *  
11     * hint: you should only alter the puzzle method  
12     */  
13     public static void main(String[] args) {  
14         System.out.println(puzzle(459));  
15         System.out.println(puzzle(3));  
16         System.out.println(puzzle(17));  
17     }  
18  
19     public static int puzzle(int x) {  
20         return 0;  
21     }  
22 }
```

Wie Sie sehen, besteht die Aufgabenstellung darin, für die Methode `puzzle` (Zeile 18-20) eine Implementierung zu finden, so dass die vorgegebenen Werte (Zeile 6-8) ausgegeben werden.

Schwierig? Möglicherweise haben Sie Lösung bereits erkannt. In diesem Beispiel ist also die Methode `puzzle` durch folgende Programmierung zu ergänzen:

## Lösung Puzzle0001

```
public static int puzzle(int x) {  
    return -x;  
}
```

## Aufgaben

Hier finden Sie nun viele weitere ähnliche Aufgaben Lösungen.

### PACKAGE NAME!

Beachten Sie, dass Sie bei einer Übernahme/Kopie der Klassen eventuell das Package anpassen müssen.

## Puzzle0002

Puzzle0002.java

```
public class Puzzle0002 {  
  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      460  
     *      4  
     *      18  
     *  
     * hint: you should only alter the puzzle method  
     */  
    public static void main(String[] args) {  
        System.out.println(puzzle(459));  
        System.out.println(puzzle(3));  
        System.out.println(puzzle(17));  
    }  
  
    public static int puzzle(int x) {  
        return 0;  
    }  
}
```

## Puzzle0003

Puzzle0003.java

```
public class Puzzle0003 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      918
     *      6
     *      34
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(459));
        System.out.println(puzzle(3));
        System.out.println(puzzle(17));
    }

    public static int puzzle(int x) {
        return 0;
    }
}
```

## Puzzle0004

Puzzle0004.java

```
public class Puzzle0004 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      462
     *      6
     *      117
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(459,3));
        System.out.println(puzzle(3,3));
        System.out.println(puzzle(17,100));
    }

    public static int puzzle(int x, int y) {
        return 0;
    }
}
```

## Puzzle0103

Puzzle0103.java

```
public class Puzzle0103 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      121
     *      9
     *      16
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(11));
        System.out.println(puzzle(3));
        System.out.println(puzzle(4));
    }

    public static int puzzle(int x) {
        return 0;
    }
}
```

## Puzzle0104

Puzzle0104.java

```
public class Puzzle0104 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      33
     *      9
     *      12
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(11));
        System.out.println(puzzle(3));
        System.out.println(puzzle(4));
    }

    public static int puzzle(int x) {
        return 0;
    }
}
```

## Puzzle0105

Puzzle0105.java

```
public class Puzzle0105 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      4
     *      12
     *      6
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(12));
        System.out.println(puzzle(36));
        System.out.println(puzzle(18));
    }

    public static int puzzle(int x) {
        return 0;
    }
}
```

## Puzzle0106

Puzzle0106.java

```
public class Puzzle0106 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      3
     *      1
     *      2
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(12));
        System.out.println(puzzle(36));
        System.out.println(puzzle(18));
    }

    public static int puzzle(int x) {
        return 0;
    }
}
```

## Puzzle0107

Puzzle0107.java

```
public class Puzzle0107 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      456
     *      0
     *      -83
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(459,3));
        System.out.println(puzzle(3,3));
        System.out.println(puzzle(17,100));
    }

    public static int puzzle(int x, int y) {
        return 0;
    }
}
```

## Puzzle0108

Puzzle0108.java

```
public class Puzzle0108 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      465
     *      9
     *      217
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(459, 3));
        System.out.println(puzzle(3, 3));
        System.out.println(puzzle(17, 100));
    }

    public static int puzzle(int x, int y) {
        return 0;
    }
}
```

## Puzzle0109

"Puzzle0109.java

```
public class Puzzle0109 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      1377
     *      9
     *      1700
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(459,3));
        System.out.println(puzzle(3,3));
        System.out.println(puzzle(17,100));
    }

    public static int puzzle(int x, int y) {
        return 0;
    }
}
```

## Puzzle0110

Puzzle0110.java

```
public class Puzzle0110 {  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      460  
     *      4  
     *      50  
     *  
     * hint: you should only alter the puzzle method  
     */  
    public static void main(String[] args) {  
        System.out.println(puzzle(459, 3));  
        System.out.println(puzzle(3, 3));  
        System.out.println(puzzle(17, 100));  
    }  
  
    public static int puzzle(int x, int y) {  
        return 0;  
    }  
}
```

## Puzzle0112

Puzzle0112.java

```
public class Puzzle0112 {  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      0  
     *      1  
     *      2  
     *  
     * hint: you should only alter the puzzle method  
     */  
    public static void main(String[] args) {  
        System.out.println(puzzle(459));  
        System.out.println(puzzle(4));  
        System.out.println(puzzle(17));  
    }  
  
    public static int puzzle(int x) {  
        return 0;  
    }  
}
```

## Puzzle0113

Puzzle0113.java

```
public class Puzzle0113 {  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      1  
     *      1  
     *      3  
     *  
     * hint: you should only alter the puzzle method  
     */  
    public static void main(String[] args) {  
        System.out.println(puzzle(459));  
        System.out.println(puzzle(3));  
        System.out.println(puzzle(17));  
    }  
  
    public static int puzzle(int x) {  
        return 0;  
    }  
}
```

## Puzzle0114

Puzzle0114.java

```
public class Puzzle0114 {  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      10  
     *      1  
     *      10  
     *  
     * hint: you should only alter the puzzle method  
     */  
    public static void main(String[] args) {  
        System.out.println(puzzle(459));  
        System.out.println(puzzle(3));  
        System.out.println(puzzle(17));  
    }  
  
    public static int puzzle(int x) {  
        return 0;  
    }  
}
```

## Puzzle0201

Puzzle0201.java

```
public class Puzzle0201 {  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      6  
     *      33  
     *      168  
     *  
     * hint: you should only alter the puzzle method  
     */  
    public static void main(String[] args) {  
        int[] a1 = {1,2,3};  
        System.out.println(puzzle(a1));  
  
        int[] a2 = {4,17,9,3};  
        System.out.println(puzzle(a2));  
  
        int[] a3 = {4,71,93};  
        System.out.println(puzzle(a3));  
    }  
  
    public static int puzzle(int[] v) {  
        return 0;  
    }  
}
```

## Puzzle0202

Puzzle0202.java

```
public class Puzzle0202 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      91
     *      5
     *      140
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(7));
        System.out.println(puzzle(3));
        System.out.println(puzzle(8));
    }

    public static int puzzle(int n) {
        return 0;
    }
}
```

## Puzzle0203

Puzzle0203.java

```
public class Puzzle0203 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     * 91
     * 5
     * 140
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle(7));
        System.out.println(puzzle(3));
        System.out.println(puzzle(8));
    }

    public static int puzzle(int n) {
        return 0;
    }
}
```

## Puzzle0204

Puzzle0204.java

```
public class Puzzle0204 {

    /*
     * find an implementation for the puzzle method that produces the following
     * outputs:
     *
     *      7
     *      2
     *      0
     *
     * hint: you should only alter the puzzle method
     */
    public static void main(String[] args) {
        System.out.println(puzzle("Hello world, have a beautiful day"));
        System.out.println(puzzle("abcdefghijklmnopqrstuvwxyz"));
        System.out.println(puzzle("1234567890"));
    }

    public static int puzzle(String s) {
        return 0;
    }
}
```

## Puzzle0205

Puzzle0205.java

```
public class Puzzle0205 {  
    /*  
     * find an implementation for the puzzle method that produces the following  
     * outputs:  
     *  
     *      2  
     *      1  
     *      1  
     *  
     * hint: you should only alter the puzzle method  
     */  
  
    public static void main(String[] args) {  
        System.out.println(puzzle("Hello world, have a beautiful day",'o'));  
        System.out.println(puzzle("abcdefghijklmnopqrstuvwxyz",'g'));  
        System.out.println(puzzle("1234567890",'8'));  
    }  
  
    public static int puzzle(String s, char x) {  
        return 0;  
    }  
}
```

# ShoppingList

Programmierung einer einfachen ShoppingList.

## Spielregeln

- Das Programm soll es ermöglichen, dass bis zu zehn beliebige Produkte in die Liste eingetragen werden können.
- Sobald der Benutzer mit einem Leerstring („“) abbricht oder 10 Produkte eingegeben hat, folgt die Ausgabe der vom Benutzer erfassten Produkte.
- Wählen Sie geeignete Datentypen, Variablenbezeichnungen und Kontrollstrukturen.

# Mathematik Olympiade

Es soll eine "Mathematik Olympiade" implementiert werden.

## Spielregeln

Dazu gibt der Benutzer als erstes an, wie viele Aufgaben er lösen möchte und in welchem Wertebereich sich die einzelnen Zahlen einer Rechnung befinden sollen.

Danach werden die Aufgaben zufällig generiert. Es sollen die beiden Grundrechenarten Addition und Division möglich sein. Es sollen nur Ganzzahlen verwendet werden.

**Addition:** die beiden Summanden werden zufällig im angegebenen Wertebereich generiert.

**Division:** Dividend und Divisor werden zufällig generiert. Achten Sie darauf, dass der Divisor nicht 0 sein darf. Geht die Division nicht auf (also gäbe es einen Rest), sollen erneut zwei Zahlen generiert werden. Dies so lange, bis die Rechnung aufgeht.

Die generierte Rechnung wird dem Benutzer auf die Konsole ausgegeben, worauf er seine Lösung eingeben muss. Dies wird so lange wiederholt, bis die Anzahl zu lösenden Aufgaben (Eingabe des Benutzers zu Beginn) erreicht wurden.

Am Schluss erfolgt eine Ausgabe, wie viele Fehler der Benutzer gemacht hat.

## Beispielausgabe

Wie viele Aufgaben willst **du** lösen?

5

Gib den Wertebereich an!

10

1.Aufgabe

Addition

2+6

8

Richtig

2.Aufgabe

Addition

8+8

16

Richtig

3.Aufgabe

Addition

3+1

4

Richtig

4.Aufgabe

Division

2/2

0

Falsch

5.Aufgabe

Addition

0+3

2

Falsch

Fehleranzahl: 2

## IMPLEMENTATIONSHILFE

- In der main-Methode fragen Sie den Benutzer als erstes, wie viele Aufgaben er lösen möchte und in welchem Wertebereich sich die Zahlen befinden sollen
- In Abhängigkeit der Anzahl Aufgaben sollen Rechnungen generiert werden
- Als erstes soll zufällig bestimmt werden, ob eine Addition oder eine Division generiert werden soll
- **Bei einer Addition:**
  - lagern Sie das Generieren der Addition in eine **eigene Methode** dafür aus. Als Parameter erhält sie den Wertebereich, als Rückgabewert liefert sie, ob die Rechnung korrekt gelöst wurde oder nicht

- Es werden zwei Zufallszahlen im Wertebereich generiert
- Die korrekte Lösung wird im Programm berechnet
- Dem Benutzer wird die Rechnung präsentiert
- Der Benutzer muss die Lösung eingeben können
- Stimmen Eingabe und Lösung überein liefert die Methode true, ansonsten false zurück
- **Bei einer Division:**
  - lagern Sie das Generieren der Division in eine **eigene Methode** aus. Als Parameter erhält sie den Wertebereich, als Rückgabewert liefert sie, ob die Rechnung korrekt gelöst wurde oder nicht
  - Es werden zwei Zufallszahlen generiert. Achtung: der Divisor darf nicht 0 werden
  - Geht die Rechnung nicht auf, werden erneut zwei Zahlen generiert. Dies wird so lange wiederholt, bis die Rechnung aufgeht
  - Die korrekte Lösung wird im Programm berechnet
  - Dem Benutzer wird die Rechnung präsentiert
  - Der Benutzer muss die Lösung eingeben können
  - Stimmen Eingabe und Lösung überein liefert die Methode true, ansonsten false zurück
- Nachdem die Methode für die jeweilige Rechnung beendet wurde, liefert Sie zurück, ob die Rechnung korrekt gelöst wurde oder nicht. Dieser Rückgabewert soll abgefangen werden. War die Rechnung falsch, zählen Sie einen Counter für die Anzahl falschen Rechnungen hoch.
- Nachdem alle Aufgaben gelöst wurden, geben Sie die Anzahl Fehler auf die Konsole aus.

# Memory

Programmierung eines Memory-Games. (Gedächtnisspiel)

## Spielregeln

In einer ersten Version definieren Sie eine fixe Liste an Wörtern, z.B. diese:

```
Car, Pen, Star, Apple, Book, Printer, Highway, Cat, Clock, Tree
```

Der Benutzer soll nun zwischen drei Befehlen wählen können:

- **learn/lernen:** alle Wörter werden angezeigt
- **guess/raten:** der Benutzer kann ein Wort raten
- **stop:** das Programm endet

Die Funktionen `learn` und `guess` sollen solange gewählt werden können, bis der Benutzer `stop` eingibt (reicht nach Schlaufe). Dann beendet das Programm.

Wählen Sie geeignete Datentypen, Variablenbezeichnungen und Kontrollstrukturen.

## Einfache Umsetzung

- Wenn ein Benutzer ein Wort eingibt, welches in der Liste enthalten ist, dann erhöhen Sie eine entsprechende *Zählervariable* um eins.
- Sobald der Wert der Variablen der Listengröße entspricht, endet das Programm und gratuliert dem Benutzer.

## Fortgeschrittene Umsetzung

Erstellen Sie eine neue Version des Games, am besten mit einer Kopie der bisherigen Lösung.

Wählen Sie aus der folgenden Liste einzelne schwierigere Anforderungen, mit denen Sie interessante Fragen der Programmierung festigen:

- Das Programm kann sich für jedes Wort individuell merken, ob es bereits richtig geraten wurde.
  - Bei der einfachen Version kann man "cheaten" und 10 mal das gleiche Wort eingeben ;)
- Das Programm fragt direkt nach dem Wort an einer bestimmten Position.
  - z.B. wie heisst das Wort an Position 4? Korrekte Antwort: Apple
- Das Programm lässt es zu, dass der Benutzer die Worte selbst erfassen kann.

# Zeitrennen Radsport

Simulieren Sie mit einem Programm ein Zeitrennen aus dem Radsport:

## Spielregeln

- In einer definierten Startaufstellung fährt ein Radrennfahrer nach dem anderen das Rennen und erreicht dabei eine Zeit zwischen 30 Minuten und einer Stunde (1800 – 3600 Sekunden).
- Direkt nach der Zieleinfahrt erscheint der Fahrer mit der aktuell gefahrenen Zeit auf der Konsole.
- Nachdem alle Fahrer das Rennen beendet haben, wird eine Rangliste mit den erreichten Zeiten ausgegeben.

Beispiel einer Ausgabe des Programms:

```
Racer Van Vleuten finished: 3167
Racer Van Der Breggen finished: 2753
Racer Vos finished: 2231
Racer Spratt finished: 1975
Racer Moolman-Pasio finished: 2560
Racer Pieters finished: 3025
Racer Rivera finished: 2003
Racer Van Djik finished: 2093
Racer Niewiadoma finished: 2912
Racer Hosking finished: 1950
***** Rankings *****
1: Hosking: 1950
2: Spratt: 1975
3: Rivera: 2003
4: Van Djik: 2093
5: Vos: 2231
6: Moolman-Pasio: 2560
7: Van Der Breggen: 2753
8: Niewiadoma: 2912
9: Pieters: 3025
10: Van Vleuten: 3167
```

## Implementierungshilfe

Folgende Hinweise helfen Ihnen bei der Implementierung:

- Definieren Sie die Startaufstellung mit allen Fahrern in einem Array.
- Für die Rangliste benötigen sie entweder zwei Arrays (einen für die Namen, einen zweiten für die gefahrene Zeit) ODER falls Sie sich mit zweidimensionalen Arrays auskennen, definieren Sie einen zweidimensionalen Array für die Zeit sowie den Namen eines Rennfahrers.
- Für jeden Fahrer in der definierten Startreihenfolge berechnen Sie die Rennzeit, welche zwischen 1800 und 3600 Sekunden liegen soll und geben diese auf die Konsole aus.

- Nachdem die Zeit für einen Fahrer berechnet wurde, sortieren Sie die Zeit sowie den Fahrer in die Arrays für die Rangliste ein. Dieses Einfügen an der korrekten Position in den Arrays lagern Sie in eine eigene Methode aus, welche Sie aufrufen, nachdem die Zeit berechnet wurde. Die Methode nennen Sie `insert`. Bedenken Sie, dass alle Fahrer, die langsamer gefahren sind wie der aktuelle Fahrer, im Array eine Position nach hinten geschoben werden müssen. Achten Sie darauf, dass Sie keine Werte überschreiben!
- Nachdem alle Fahrer das Rennen beendet haben und an der korrekten Stelle eingesortiert wurden, geben Sie die Rangliste auf die Konsole aus. Dies kann ebenfalls in eine Methode namens `printRanking` ausgelagert werden.

# Effizienter mit Eclipse

## Workspace

Mit Workspace (Arbeitsbereich) wird ein bestimmter Ordner bezeichnet, in dem ein "beliebige" Anzahl an Eclipse-Projekten gespeichert werden können. Mit Vorteil ist der Begriff "workspace" in der Ordnerbezeichnung enthalten. Wenn man in diesen "workspace"-Ordner hinein geht, so ist mindestens ein Unterordner mit dem Namen ".metadata" enthalten, woran man erkennen kann, dass man sich eine Ebene unterhalb des workspace-Ordner befindet.

### ⚠ CAUTION

Wichtig ist jeweils beim öffnen, dass der workspace-Ordner ausgewählt wird, und nicht ein Unterordner.

- Löschen, Umbenennen, Verschieben von Projekten sollten nur mit einer gewissen Vorsicht vollzogen werden, am besten nur aus Eclipse, da sonst der workspace beschädigt oder zerstört werden kann.

## Projekte im Workspace

Der Aufbau von Java-Projekten enthält meist folgende Ordner:

- `bin`: Hauptverzeichnis für kompilierte `.class`-Dateien
- `src`: Hauptverzeichnis für Java-Klassen also `.java`-Dateien
- `.classpath` und `.project`: Eclipse spezifische Dateien, welche das Projekt beschreiben

### ⚠ CAUTION

Auch Projekte im workspace sollte man nur mit Vorsicht und wenn man weiss, was man tut, ausserhalb von Eclipse ändern.

## Code Editor, Console und weitere Ansichten

- **Laschen** / Begrüssungsbild
  - Doppelklick: die Ansicht wechselt innerhalb Eclipse zwischen Voll- und Teilbild
  - Schliessen: über das X oben rechts an der Lasche
- **Package Explorer**: zeigt den "Projektbaum" mit der src-Ordner, in dem die Java-Klassen dargestellt sind, in der Titelzeile ist ein gelbes Pfeilpaar <->, ist es eingedrückt, wird im Projektbaum die Klasse, welche rechts daneben im Editor geöffnet wird, angezeigt.
- **Code Editor**: dient zum bearbeiten der Java-Klassen
- **Console**: gibt Meldungen aus dem Programm aus
- **Problems**: zeigt eine Zusammenfassung der Probleme im aktuellen Projekt

- **Outline:** zeigt eine schematische Ansicht der aktuellen Klasse

### (!) FEHLT EINE ANSICHT?

Hier ist der Weg, jede beliebige Ansicht wieder anzuzeigen:

- Menü: Window > Show View > Auswahl aus Liste
- ⌘ Ansicht zurücksetzen: Window > Perspective -> Reset Perspective

## Wichtigste Shortcuts

- `CTRL+SHIFT+F`: Formattierung des Quellcodes durch Eclipse
- `CTRL+SPACE`: Auto-Complete in vielen Situationen
- `CTRL+7`: ausgewählte Zeilen werden zeilenweise mit // auskommentiert (für den Compiler unsichtbar)
- `SHIFT+ALT+R`: Umbenennen des ausgewählten Elements

## Warnungen und wirkliche Fehler

Eclipse bringt viele Warnungen, z.B. wenn eine Variable erst angelegt ist, erscheint eine Warnung, dass die Variable noch nicht verwendet wird. Diese Warnungen mit gelbem Ausrufezeichen können ignoriert werden.

Wirkliche Fehler werden durch eine rote Wellenlinie angezeigt. Häufig kann Eclipse feststellen, wo der Fehler ist, und die Wellenlinie und der Hinweis können bei der Fehlersuche helfen. Es gibt aber auch Situation, wo man selbst etwas analysieren muss, wo der Fehler oberhalb oder unterhalb ist.

## Ist der Workspace beschädigt?

### Projekte aus defektem workspace retten

Können Java-Klassen in einem workspace nicht mehr bearbeitet oder ausgeführt werden:

- erstellen Sie zuerst eine Kopie des workspace an einem sicheren Ort (Ordner).
- Prüfen Sie, ob der workspace richtig ausgewählt und geöffnet wurde

Falls keine Verbesserung eintritt, können Projekte wie folgt gerettet werden:

- sichern Sie den alten workspace an einem sicheren Ort (Ordner)
- erstellen Sie zuerst einen neuen sauberen workspace
- öffnen Sie Eclipse nun und wählen diesen workspace aus
- vergewissern Sie sich, dass im workspace-Ordner nur der .metadata-Ordner enthalten ist
- in Eclipse gehen Sie nun via Menü File > Import zum Dialog "Import"
- im 1. Schritt im Dialog "Import" wählen Sie aus der Liste "General > Existing Projects into Workspace > Next"

- im 2. Schritt im Dialog "Import" muss das Wurzelverzeichnis des Workspace, aus dem Sie Projekte retten wollen, ausgewählt werden, wählen Sie unterhalb unbedingt "Copy projects into workspace" an

## Ausserhalb der Schule üben?

Super, nachfolgend finden Sie die Schritte, welche dazu nötig sind.

- **Voraussetzung 1:** Java downloaden und installieren Java auf einem Rechner
- **Voraussetzung 2:** Eclipse downloaden und installieren



### ARBEITEN SIE BEVORZUGT MIT IHREM LAPTOP

Wenn Sie auch in der Schule mit ihrem Laptop arbeiten, hat dies der Vorteil, dass sie ohne Probleme auch zuhause arbeiten könnt, ohne dass die Dateien kopiert werden müssen.

## Alternativen zu Eclipse

Netbeans ist eine ebenfalls Open-Source-Alternative, während es sich bei IntelliJ IDEA um ein kommerzielles Programm der Firma JetBrains handelt (es ist allerdings für Lernende derzeit jeweils für ein Jahr kostenlos lizenzierbar), das bei vielen Entwicklern sehr beliebt ist. IDEA kann, neben anderen Programmen von JetBrains, von Lernenden während der Ausbildung kostenlos genutzt werden.

Beachten Sie, dass im Unterricht nur der Umgang mit Eclipse unterstützt wird. Wählen Sie eine andere IDE aus, so sind Sie auf sich selbst gestellt.