

# METHODEN UND WERKZEUGE IN DER SOFTWAREENTWICKLUNG

Video by Jakub Zerdzicki: <https://www.pexels.com/video/dynamic-code-display-on-computer-screen-33187808/>



# EINFÜHRUNG



# VORSTELLUNG



## WAS MACHT DIE SAP?

-  Deutschlands größtes Softwareunternehmen
-  Enterprise Software für Unternehmen weltweit
-  ERP-Systeme, Cloud-Lösungen, Datenbanken
-  400.000+ Kunden in über 180 Ländern
-  ~105.000 Mitarbeiter weltweit

# WER SIND SIE?

Bitte stellen Sie sich kurz vor:

-  Name?
-  Ausbildungsbetrieb?
-  Warum Software Engineering?
-  Erste Programmiererfahrungen gesammelt?
-  Haben Sie Erwartungen an die Vorlesung?  
Wenn ja, welche?
-  Was machen Sie, wenn Sie nicht gerade in der Vorlesung sitzen?

# ALLGEMEINES ZUR VORLESUNG

## ORGANISATORISCHES

-  Es herrscht generelle Anwesenheitspflicht
-  Empfehlung zum Mitschreiben
-  Mischung aus Theorie, Demo und Übungen

## AKTIVE MITARBEIT ERWÜNSCHT

-  Zuhören
-  Nachfragen
-  Diskutieren
-  Aufgaben bearbeiten
-  Feedback geben

# ANWESENHEITSPFLICHT

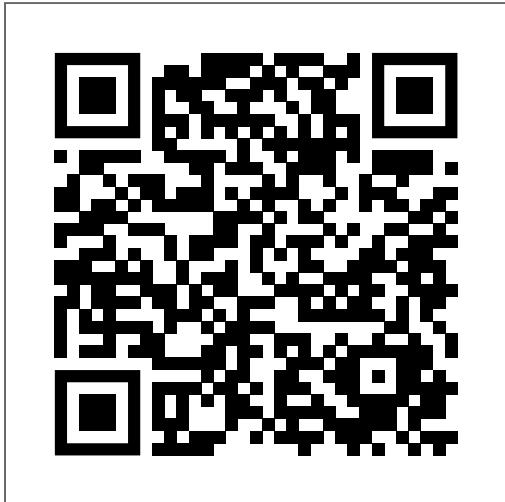


Anwesenheit wird per Unterschriftliste kontrolliert.

# Übungsaufgaben



Übungsaufgaben - und später auch die dazugehörigen Lösungen - finden Sie unter:



**<https://github.com/Niyada/lecture-software-engineering>**

## KONTAKT

 [goetz.nils@edu.dhbw-karlsruhe.de](mailto:goetz.nils@edu.dhbw-karlsruhe.de)



# PRÜFUNGSLEISTUNG



Portfolio-Prüfung über 2 Semester bestehend aus 2 Veranstaltungen:

- 1 Methoden und Werkzeuge in der Softwareentwicklung
- 2 Moderne Programmierkonzepte



Bearbeitung der Aufgaben in eigener Zeit

## AI IM STUDIUM

- 🤖 KI ist ein nützliches, neues Werkzeug
- ⚡ In der Software-Branche wichtig: Am Ball bleiben
- ✅ Manche Aufgaben löst KI sehr gut
- ✗ Andere Aufgaben noch nicht
- 🧠 1. Semester: Grundlagen selbst lernen ist wichtig
- ⚠ Gefahr: Wichtige Basics verpassen
- 💡 Als Lernhilfe zum Erklären sehr hilfreich
- 🎯 Empfehlung: Erst selbst versuchen, dann KI als Hilfe

# ENTWICKLUNGSWERKZEUGE



💻 Wie schreibst und bearbeitest du Code? 🖍

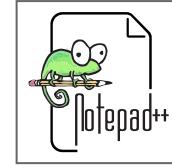


QUESTION MARK Wie schreibst und bearbeitest du Code? 🖊



## EINFACHE EDITOREN

- Leichtgewichtig
- Einfach zu bedienen
- Stark eingeschränktes Feature-Set
- Beispiele:
  - Notepad++
  - Sublime Text
  - Windows Editor



# ENTWICKLUNGSUMGEBUNGEN



Entwicklungsumgebungen sollten den kompletten Projektlebenszyklus in allen Phasen unterstützen



**DHBW**  
Duale Hochschule  
Baden-Württemberg

# WAS IST EINE IDE

- 🏗 IDE = Integrated Development Environment
- 📦 Software-Anwendung, die umfassende Einrichtungen für Softwareentwicklung bereitstellt
- 🔧 Typischerweise beinhalten IDEs Werkzeuge, die über einen einfachen Texteditor hinaus gehen:
  - ⟳ Compiler/Interpreter-Integration
  - ⚙ Build-Automatisierungstools
  - ✓ Syntax-Check
  - 🐛 Debugger
  - 🔍 Code-Analyse
  - 🧪 Testautomatisierung
  - 🧭 Code-Navigation
  - 📋 Versionskontrolle
  - ✨ AI-Integration
  - 🧩 Plugin-Architektur
  - ⚡ u.v.m.

# When I am writing the code in IDE..



# IDE VS. EDITOR

Feature	Editor	IDE
 Code-Bearbeitung	Ja	Ja
 Syntax-Hervorhebung	Ja	Ja
 Code-Vervollständigung	Begrenzt (Plugins)	Ja
 Debugger	Nein (Plugins)	Ja
 Build-Tools	Nein (Plugins)	Ja
 Versionskontrolle	Nein (Plugins)	Ja
 Projektmanagement	Nein	Ja
 Ressourcenverbrauch	Niedrig	Hoch

**Feature**

**Editor**

**IDE**

 **Lernkurve**

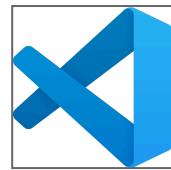
Flach

Steil



# BEKANNTE IDES

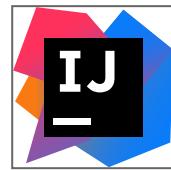




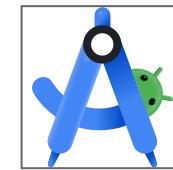
Visual Studio Code



Xcode



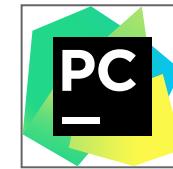
IntelliJ



Android Studio

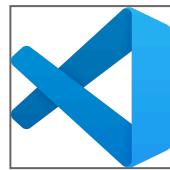


Eclipse

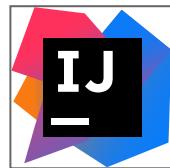


PyCharm

## BEKANNTE IDES



Visual Studio Code



IntelliJ



**Eclipse**



## BELIEBTESTE JAVA IDES

Quelle: Perforce JRebel. (2025). *2025 Java Developer Productivity Report: Java Development Trends + Analysis*.

Abgerufen von <https://www.jrebel.com/resources/java-developer-productivity-report-2025>

# ECLIPSE-PROJEKT



## ALLGEMEIN

- Offene Plattform für Entwicklungswerkzeuge
- Läuft auf vielen Betriebssystemen
- Mit und ohne grafische Oberfläche
- Sprachneutral
  - Java, C, HTML, XML, Python, PHP, ...
- Open Source, große Community



## PLATTFORM

- Über Plugin-Konzept sehr flexibel erweiterbar
- Java-IDE ist ein "Package" auf Basis der Plattform und den Java Development Tools



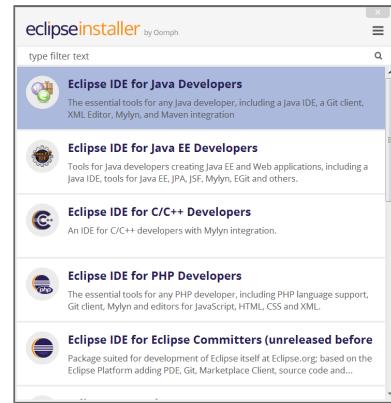
**DHBW**  
Duale Hochschule  
Baden-Württemberg

# ECLIPSE - VERSIONEN & PACKAGES



## VERSIONEN

- 📅 2006 – 2018: Jährlich eine neue Version mit Projektnamen  
Beispiele: „Callisto“ (2006), „Juno“ (2012), „Photon“ (2018)
- 🔄 Seit 2018: 3-monatiger Release-Zyklus.  
Namen „Jahr-Monat“ (2018-09, 2018-12, ...)

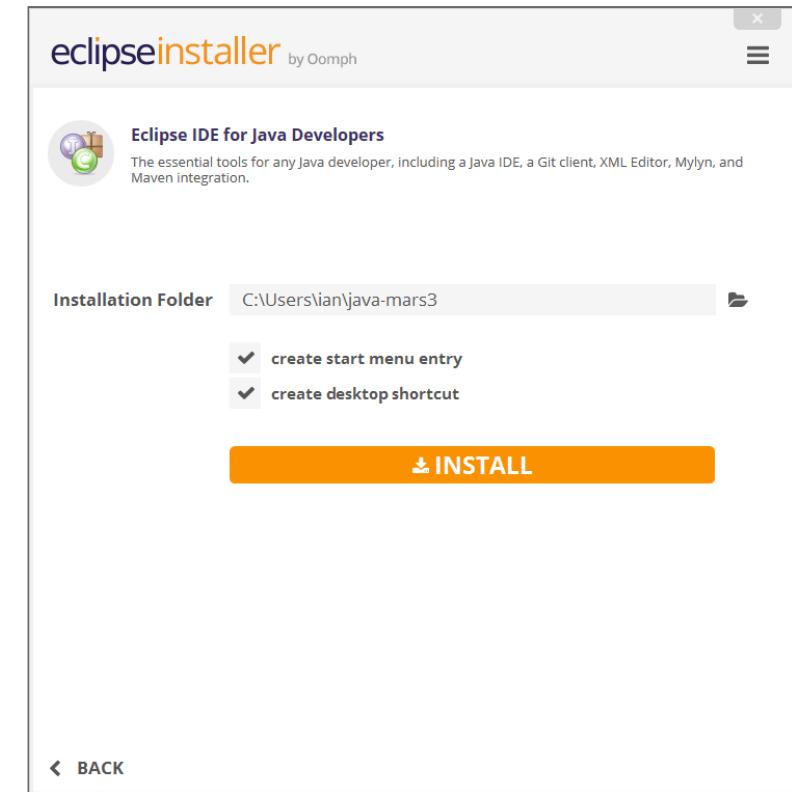


## PACKAGES

- 🔧 Verschiedene Pakete mit verschiedenen Tools
- ☕ Ideal für Java-Entwicklung:  
„Eclipse IDE für Java Developers“
- 🧩 Zusätzliche Komponenten können jederzeit nachinstalliert werden

# ECLIPSE IDE FOR JAVA DEVELOPERS

-  Java IDE
-  Versionskontrolle mit Git
-  XML Editor
-  Build-Management
  -  Maven
  -  Gradle
  -  Ant



 Das "Eclipse IDE for Java Developers" Paket enthält alle wichtigen Werkzeuge für die Java-Entwicklung 

# INTELLIJ IDEA



## ALLGEMEIN

- 💻 Entwickelt von JetBrains
- ☕ Primär für Java-Entwicklung
- 🧩 Unterstützt viele weitere Sprachen (Kotlin, Groovy, Scala, JavaScript, TypeScript, SQL, u.v.m.)
- 🔒 Community Edition (Open Source) & Ultimate Edition (kommerziell)
- 🛠 Starke Integration von Build-Tools, Versionskontrolle, Datenbanken, u.v.m.



## FEATURES

- 💡 Intelligente Code-Vervollständigung
- 🐛 Integrierter Debugger und Test Runner
- 🔍 Erweiterte Code-Analyse und Refactoring-Werkzeuge
- 🧩 Plugin-Ökosystem



**DHBW**  
Duale Hochschule  
Baden-Württemberg

# VISUAL STUDIO CODE



## ALLGEMEIN

- Entwickelt von Microsoft
- Kostenlos und Open Source
- Unterstützt viele Programmiersprachen durch Erweiterungen (Java, Python, JavaScript, C++, u.v.m.)
- Leichtgewichtig und schnell
- Cross-Platform (Windows, macOS, Linux)

Für Java-Entwicklung sollten entsprechende Erweiterungen installiert werden (z.B. [Extension Pack for Java](#))



## FEATURES

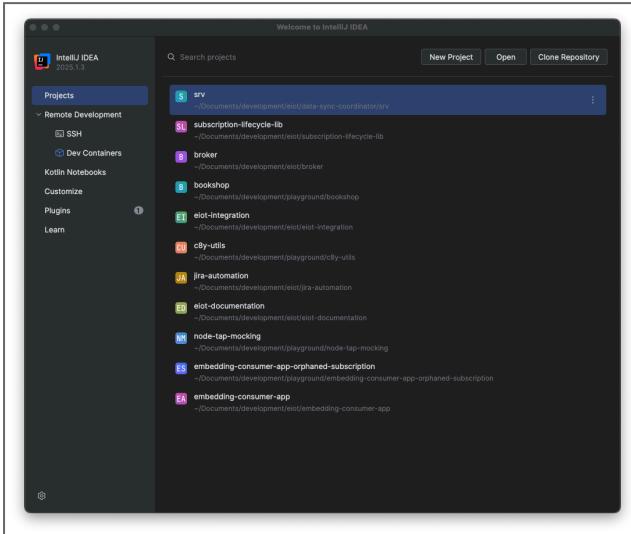
- Intelligente Code-Vervollständigung mit IntelliSense
- Integrierter Debugger
- Eingebaute Git-Unterstützung
- Großes Ökosystem an Erweiterungen und Themes



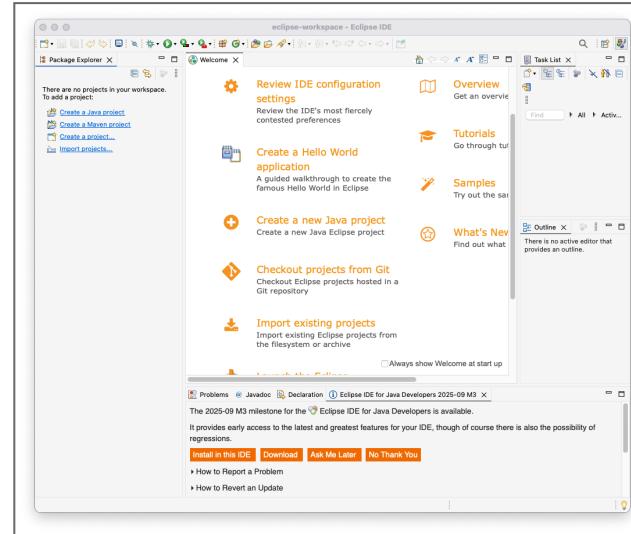
**DHBW**  
Duale Hochschule  
Baden-Württemberg

# START-SCREENS

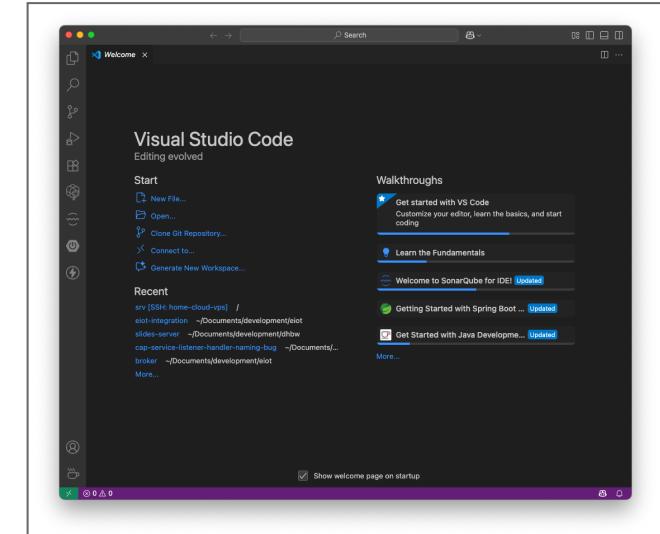
## INTELLIJ IDEA



## ECLIPSE IDE

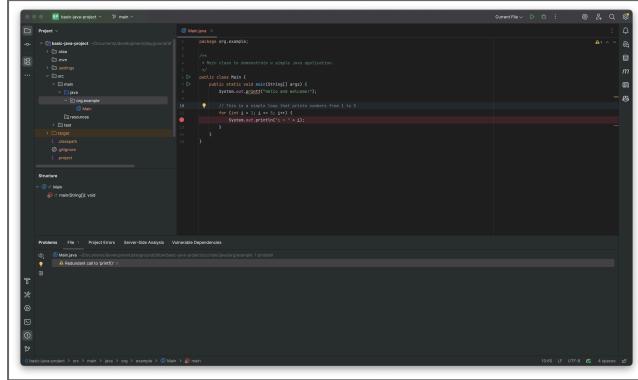


## VISUAL STUDIO CODE

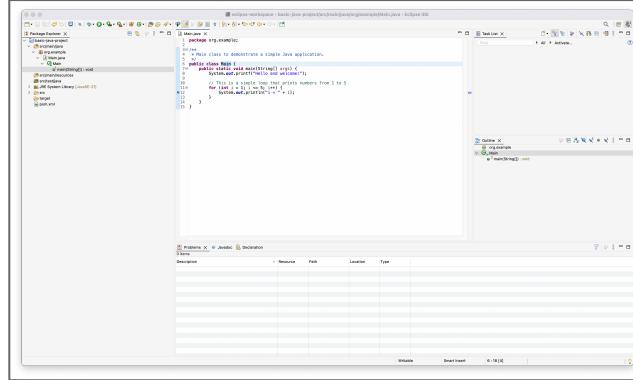


# WORKBENCH

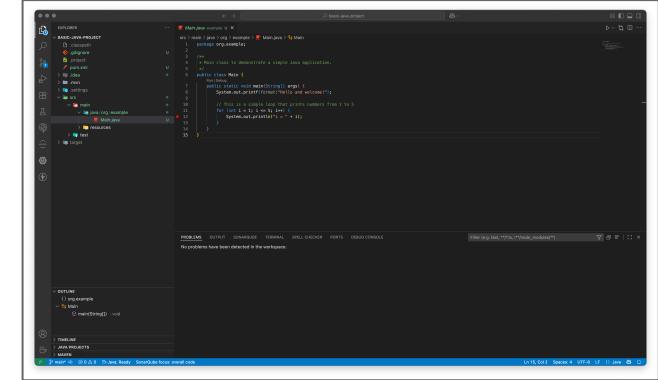
## INTELLIJ IDEA



## ECLIPSE IDE



## VISUAL STUDIO CODE



# WORKBENCH



The image shows three side-by-side screenshots of Java IDEs, likely IntelliJ IDEA, Eclipse IDE, and another Eclipse instance, all displaying the same Java code for a main class.

**Code:**

```
1 package org.example;
2 /**
3  * Main class to demonstrate a simple Java application.
4  */
5 public class Main {
6     public static void main(String[] args) {
7         // This is a simple loop that prints numbers from 1 to 5.
8         for (int i = 1; i <= 5; i++) {
9             System.out.println("i = " + i);
10        }
11    }
12 }
```

**IDE 1 (Left): IntelliJ IDEA**

- Project Structure: Shows the project tree with `basic-java-project` containing `src`, `target`, and configuration files like `.idea`, `.mvn`, and `pom.xml`.
- Code Editor: Shows the `Main.java` file with the above code.
- Toolbars and Menus: Standard IntelliJ IDEA interface.
- Bottom Status Bar: Shows the path `basic-java-project > src > main > java > org > example > Main` and some icons.

**IDE 2 (Middle): Eclipse IDE**

- Project Explorer: Shows the project tree with `basic-java-project` containing `src` (with `main` and `test`), `target`, and configuration files like `pom.xml` and `project.properties`.
- Code Editor: Shows the `Main.java` file with the above code.
- Toolbars and Menus: Standard Eclipse IDE interface.
- Bottom Status Bar: Shows the path `basic-java-project > src > main > java > org > example > Main` and some icons.

**IDE 3 (Right): Eclipse IDE**

- Project Explorer: Shows the project tree with `BASIC-JAVA-PROJECT` containing `src` (with `main` and `test`), `target`, and configuration files like `pom.xml` and `project.properties`.
- Code Editor: Shows the `Main.java` file with the above code.
- Toolbars and Menus: Standard Eclipse IDE interface.
- Bottom Status Bar: Shows the path `src > main > java > org > example > Main` and some icons.

# WORKBENCH



The image displays three side-by-side screenshots of Java IDE environments, all showing the same Java code in a file named `Main.java`.

**Top Screenshot (IntelliJ IDEA):**

IDEA interface with a dark theme. The left sidebar shows the project structure with a tree view of files and folders. The main editor window displays the `Main.java` code. The status bar at the bottom indicates "No problems have been detected in the workspace."

```
package org.example;  
/*  
 * Main class to demonstrate a simple Java application.  
 */  
public class Main {  
    public static void main(String[] args) {  
        System.out.printf("Hello and welcome!");  
    }  
}
```

**Middle Screenshot (Eclipse IDE):**

Eclipse interface with a light theme. The left sidebar shows the "Package Explorer" with the project structure. The main editor window displays the `Main.java` code. The status bar at the bottom indicates "No problems have been detected in the workspace."

```
package org.example;  
/*  
 * Main class to demonstrate a simple Java application.  
 */  
public class Main {  
    public static void main(String[] args) {  
        System.out.printf("Hello and welcome!");  
  
        // This is a simple loop that prints numbers from 1 to 5  
        for (int i = 1; i <= 5; i++) {  
            System.out.println("i = " + i);  
        }  
    }  
}
```

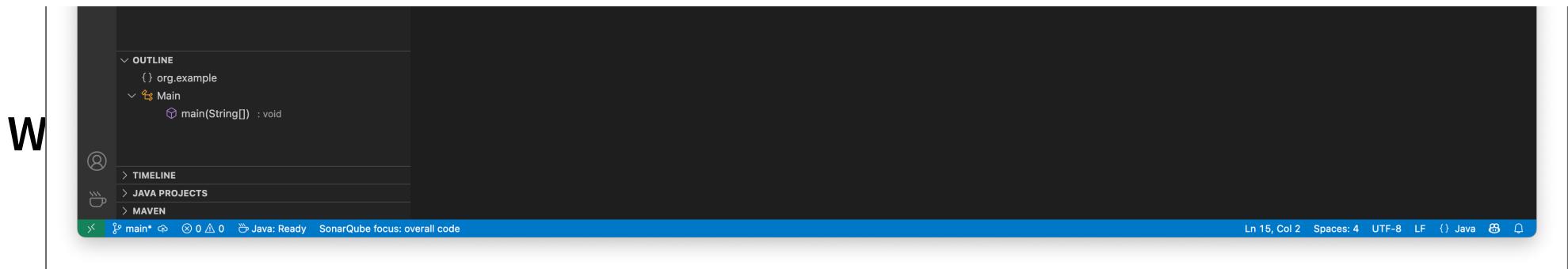
**Bottom Screenshot (VS Code):**

VS Code interface with a dark theme. The left sidebar shows the "EXPLORER" view with the project structure. The main editor window displays the `Main.java` code. The status bar at the bottom indicates "No problems have been detected in the workspace."

```
package org.example;  
/*  
 * Main class to demonstrate a simple Java application.  
 */  
public class Main {  
    public static void main(String[] args) {  
        System.out.printf("Hello and welcome!");  
  
        // This is a simple loop that prints numbers from 1 to 5  
        for (int i = 1; i <= 5; i++) {  
            System.out.println("i = " + i);  
        }  
    }  
}
```

**DHBW Logo:**

The DHBW logo is prominently displayed in the center of the bottom screenshot, partially overlapping the VS Code interface.



## 📁 Project Explorer / Package Explorer

Verwaltet Dateien, Klassen und Packages im Projekt

## 🌳 Outline View

Zeigt Struktur des aktuellen Codes im Editor

Übersicht über Methoden, Variablen und Klassen

## ❗ Problems View

Zeigt Warnungen und Fehler (z.B. Syntaxfehler)

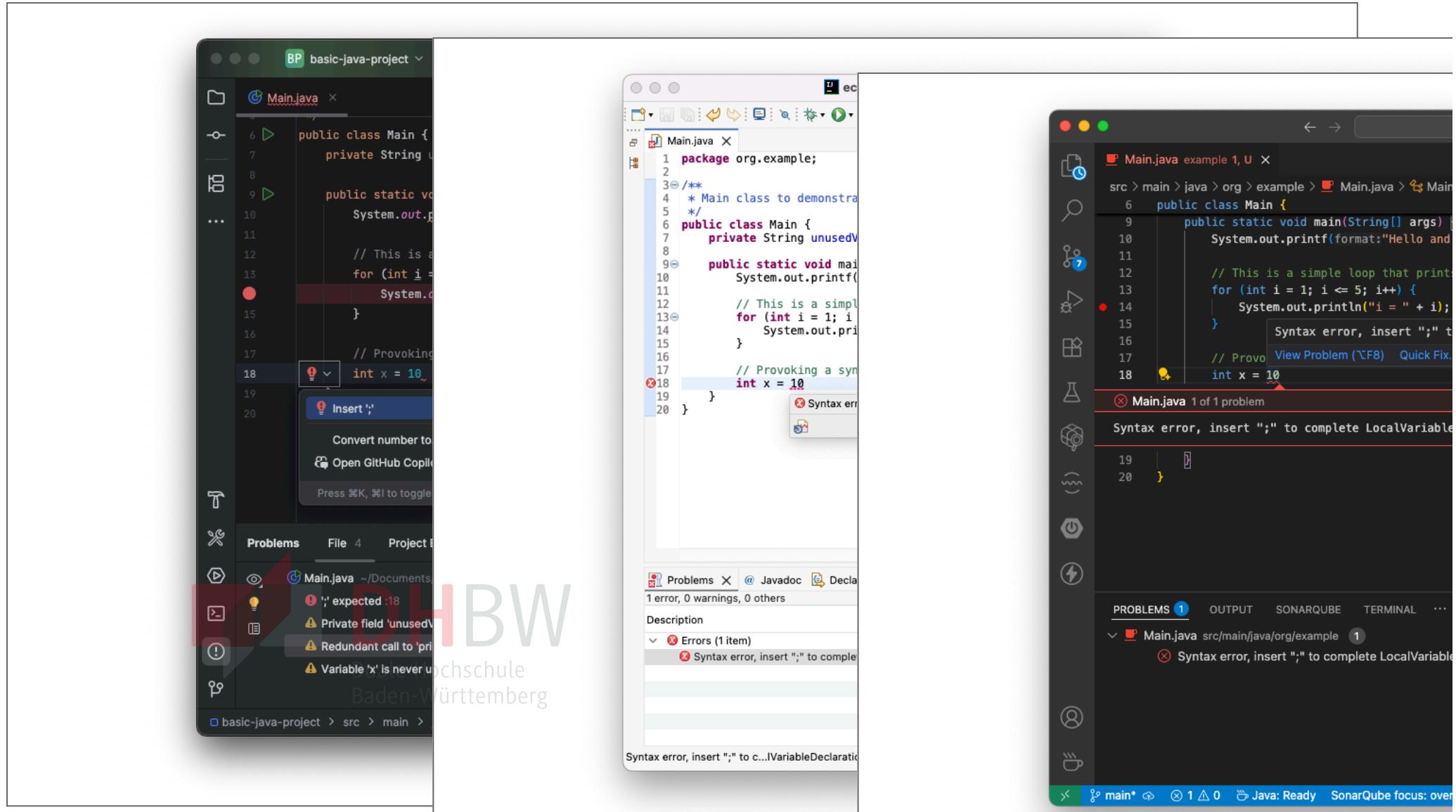
Hilft bei der schnellen Fehlerbehebung

## 📺 Console View

Standard- und Fehlerausgabe von gestarteten Programmen

Debugging-Informationen und Logs

# EDITOR





**DHBW**  
Duale Hochschule  
Baden-Württemberg

# EDITOR

The screenshot shows a Java code editor interface with a dark theme. On the left is the code editor pane displaying `Main.java`. The code contains a simple loop that prints numbers from 1 to 5. A syntax error is highlighted at the end of line 18, where the variable `x` is declared. A tooltip appears over the error, stating "Syntax error, insert ";" to complete LocalVariableDeclarationStatement Java(1610612976)". The status bar at the bottom shows the file path: `basic-java-project > src > main > java > org > example > Main > main`, the line number `Ln 18, Col 19`, and the character offset `Spaces: 4`.

File: basic-java-project

Main.java example 1, U X

src > main > java > org > example > Main.java > Main > main(String[])

```
6 public class Main {  
9     public static void main(String[] args) {  
10         System.out.printf(format:"Hello and welcome!");  
11  
12         // This is a simple loop that prints numbers from 1 to 5  
13         for (int i = 1; i <= 5; i++) {  
14             System.out.println("i = " + i);  
15         }  
16         Syntax error, insert ";" to complete LocalVariableDeclarationStatement Java(1610612976)  
17         // Prov... View Problem (F8) Quick Fix... (⌘.) Fix (⌘I)  
18         int x = 10  
19     }  
20 }
```

Main.java 1 of 1 problem

Syntax error, insert ";" to complete LocalVariableDeclarationStatement Java(1610612976)

PROBLEMS 1 OUTPUT SONARQUBE TERMINAL ... Filter (e.g. text, \*\*/\*.ts, !\*\*/node\_modules/\*\*)

Main.java src/main/java/org/example 1

Syntax error, insert ";" to complete LocalVariableDeclarationStatement Java(1610612976) [Ln 18, Col 17]

DHBW Duale Hochschule Baden-Württemberg

ain\* 1 △ 0 Java: Ready SonarQube focus: overall code Ln 18, Col 19 Spaces: 4 UTF-8 LF {} Java ⚡ 18:17 : 466

Syntax error, insert ";" to complete LocalVariableDeclarationStatement Writable Smart Insert 18 : 17 : 466

basic-java-project > src > main > java > org > example > Main > main 18:19 LF UTF-8 4 spaces



**DH**BW  
Duale Hochschule  
Baden-Württemberg

# IDE - REFACTORING



Automatisierte Code-Umstrukturierung ohne Funktionsänderung



Umbenennen von Variablen, Methoden und Klassen im gesamten Projekt



Extrahieren von Methoden, Klassen und Interfaces aus bestehendem Code

## INTELLIJ IDEA



Umfangreichste Refactoring-  
Features

## ECLIPSE IDE



Starke Java-Refactoring-  
Unterstützung

## VS CODE



Grundlegende Funktionen  
verfügbar

# IDE - CODE NAVIGATION

## Go to Definition

Springe zur Definition von Klassen, Methoden oder Variablen

## Find Usages

Finde alle Verwendungen eines Elements im gesamten Projekt

## Navigate to File/Class

Schnelles Öffnen von Dateien über Namenssuche

## Call Hierarchy

Zeigt, welche Methoden eine bestimmte Methode aufrufen

## Type Hierarchy

Darstellung der Vererbungsstruktur von Klassen

# DER DEBUGGER

- 🐞 Setzen von sogenannten Breakpoints im Code
- ▶ Schrittweises Ausführen des Codes (Step Over, Step Into, Step Out)
- 👀 Überwachen von Variablenwerten und Ausdrücken
- 🛠️ Untersuchen des Call Stacks und Threads
- 🔁 Hot Code Replacement (Code während der Laufzeit ändern)
- 👉 Werden wir noch gesondert behandeln!

## COOLE FEATURES

- 📁 **Fold/Collapse all:** Überblick in neuem Code verschaffen
- 📝 **Multicursor editing:** Mehrere Stellen gleichzeitig bearbeiten
- 🎯 **Go To Line/Column:** Schnell zu einer bestimmten Zeile springen, z.B. bei der Fehlersuche
- 🔄 **Moving, Cloning lines:** Zeilen schnell verschieben oder duplizieren
- 🔧 **Surround With:** Code schnell in Schleifen, Bedingungen oder Try-Catch-Blöcke einfügen
- ✨ **Code Reformatting:** Automatisches Formatieren des Codes nach definierten Stilregeln
- 🔍 **Command/File Search:** Schnelles Finden von Dateien oder IDE-Befehlen

## ZUSAMMENFASSUNG

- 💻 IDEs sind unverzichtbare Werkzeuge für die Softwareentwicklung
- 🔧 Bieten viele Funktionen zur Steigerung der Produktivität
- 🌐 Vielfältige Auswahl an IDEs, jede mit eigenen Stärken
- ☕ Es gibt **nicht** die eine beste IDE:
  - 👉 es kann helfen, wenn die IDE im Team einheitlich verwendet wird, aber
  - ❓ wähle die IDE, die am besten zu den eigenen Bedürfnissen passt
  - 👉 Nicht abschrecken lassen von komplexen Oberflächen - Ausprobieren und Einarbeitung lohnen sich!

# VERSIONS- KONTROLLE



## MOTIVATION

!?"Oh nein! Ich hätte gerne meine Änderungen von vor 2h wieder!"

!?"Mist! Ich habe die Datei gelöscht!"

!?"Lebenslauf\_v5\_neu\_final\_FINAL2.docx

Versionskontrolle bedeutet das Protokollieren von Änderungen an einer Datei oder einer Anzahl von Dateien über die Zeit hinweg, so dass Änderungen nachvollzogen und zu jedem Zeitpunkt auf ältere Versionen zugegriffen werden kann.

## ANWENDUNG

💡 Softwareentwicklung  
💡 Dokumentation

💡 Texte  
💡 Grafikdesign

## BEISPIELE

- Office 365
- Google Docs
- Dropbox



# WAS IST VERSIONSKONTROLLE?

## GRUNDIDEEN

- Angabe der Version oder eines Zeitstempels im Dateinamen
- Angabe innerhalb einer Datei in den sog. Metadaten
- Anlegen eines Änderungsprotokolls
- Versionierung z.B. durch Cloud-Storage Systeme (Google Drive, Dropbox, ...)
- Oder bei Software: Version Control Systems (VCS)

# VERSIONSKONTROLLE IN DER SOFTWAREENTWICKLUNG

## ⚠ HERAUSFORDERUNGEN

- ⚠ Viele Dateien, die sich stetig ändern
- ⚠ Unterschiedliche Versionen der Software (z.B. Release, Beta, ...)
- ⚠ Unterschiedliche Entwicklungszweige (z.B. Feature-Entwicklung, Bugfixing)
- ⚠ Personen überschreiben gegenseitig ihre Änderungen
- ⚠ Viele Entwickler:innen, die gleichzeitig an den Dateien arbeiten
- ⚠ Wer hat welche Änderungen vorgenommen?
- ⚠ Mein Kollege braucht die Änderungen, die ich auf meiner Maschine vorgenommen habe

# VERSIONSKONTROLLE IN DER SOFTWAREENTWICKLUNG

## 🎯 ZIELE

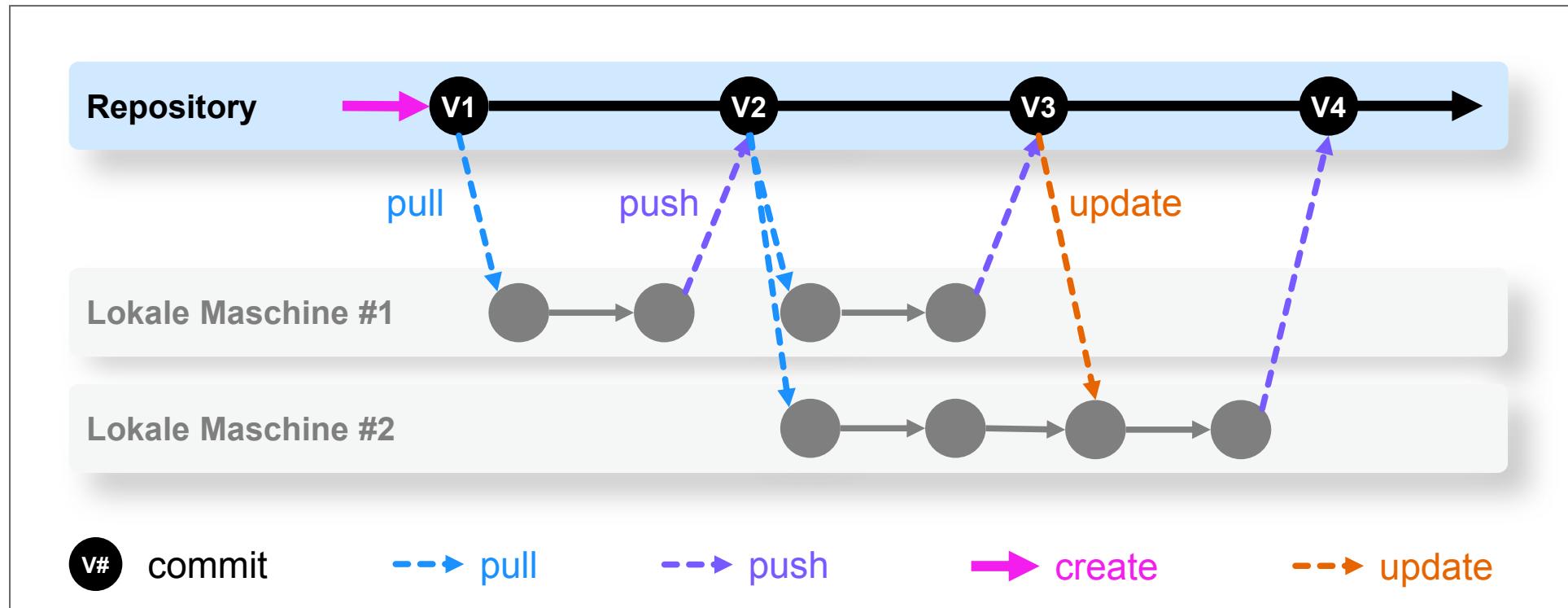
- 🎯 Ermöglichen von kollaborativen Arbeiten und paralleler Entwicklung
- 🎯 Nachvollziehbarkeit von Änderungen
- 🎯 Protokollieren aller Änderungen
- 🎯 Vermeidung von gegenseitigen Überschreibungen
- 🎯 Darstellung von Unterschieden in verschiedenen Versionen

# FUNKTIONSWEISE VON VCS

VCS = Version Control System 

- **create**: Erstellen einer Ablage (sog. Repository)
- **add**: Hinzufügen von Dateien zum Repository
- **commit**: Speichern von Änderungen im Repository;  
Jede Änderung bzw. Jeder Commit erzeugt eine neue Version
- **push**: Übertragen der Änderungen auf einen zentralen Server (optional)
- **pull**: Herunterladen der Änderungen von einem zentralen Server
- **update**: Entwickler aktualisieren ihre lokale Arbeitskopie auf die neueste Version
- **revert**: Entwickler können ihre Arbeitskopie auf frühere Versionen zurücksetzen

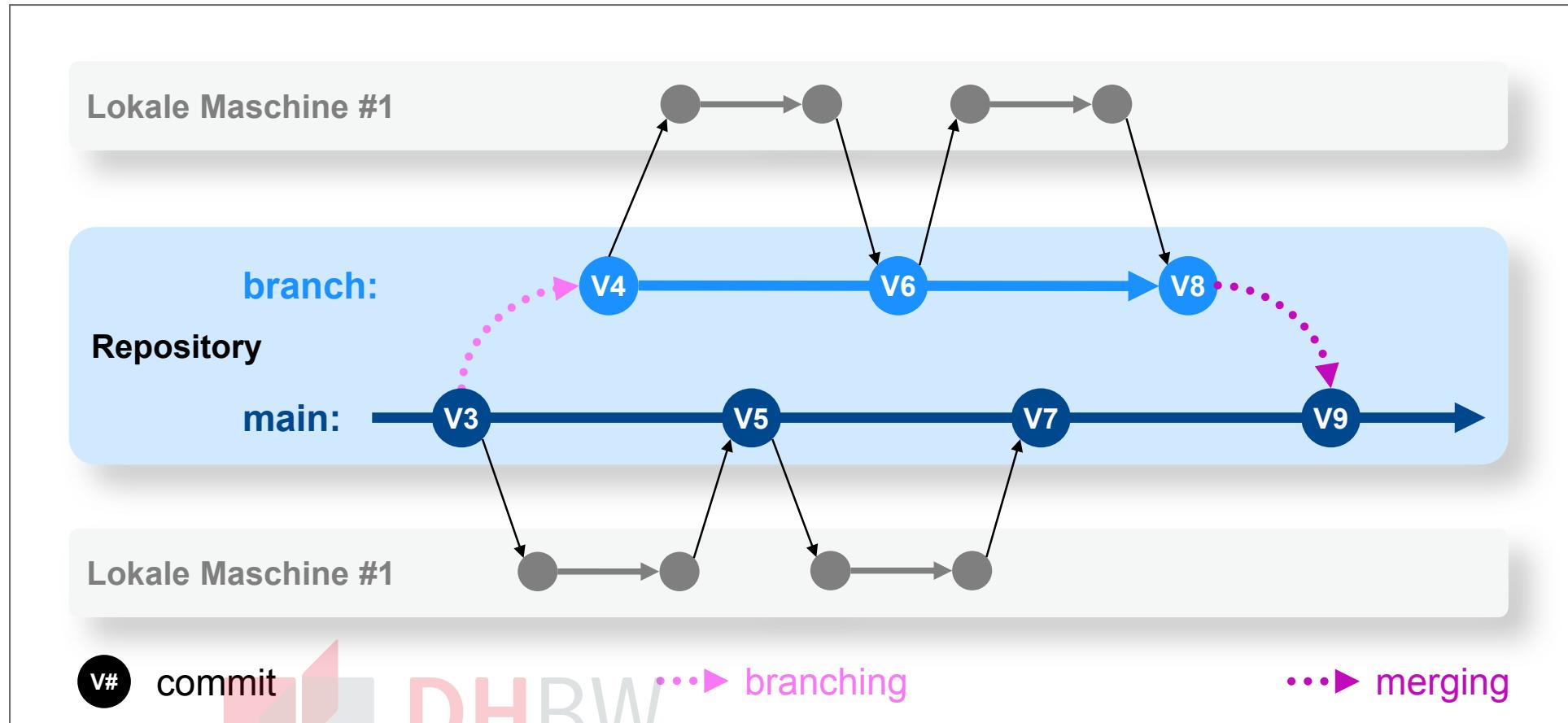
# FUNKTIONSWEISE VON VCS



## FUNKTIONSWEISE VON VCS

- Oftmals paralleles Entwickeln an unterschiedlichen „Baustellen“ eines Projektes
- branching: VCS ermöglichen dazu das Abspalten von Entwicklungszweigen  
Es wird eine Kopie der aktuellen Version erstellt, auf der weiterentwickelt wird
- branch: Entwicklung von Teilprojekten parallel in eigenen Versionspfaden
- merge: Zusammenführen von Änderungen aus verschiedenen Entwicklungszweigen

# FUNKTIONSWEISE VON VCS

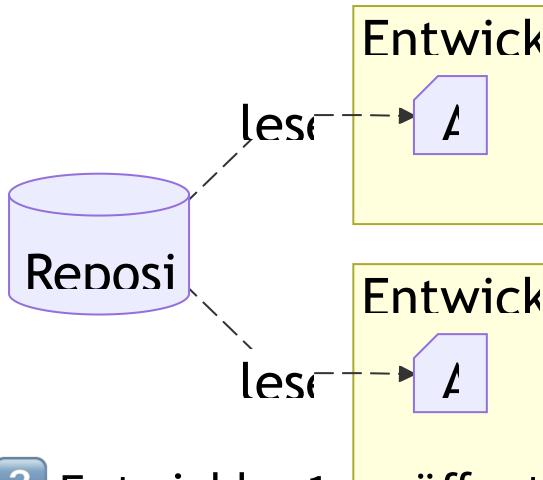


**GEMEINSAMES EDITIEREN**

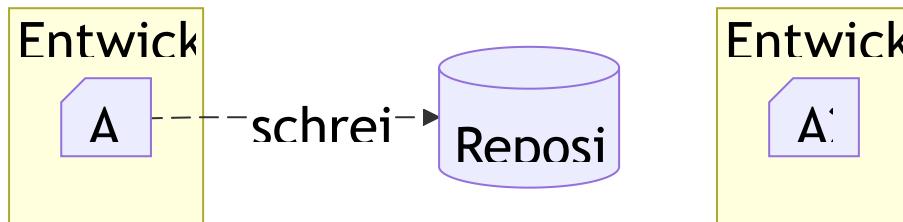
**HERAUSFORDERUNGEN**



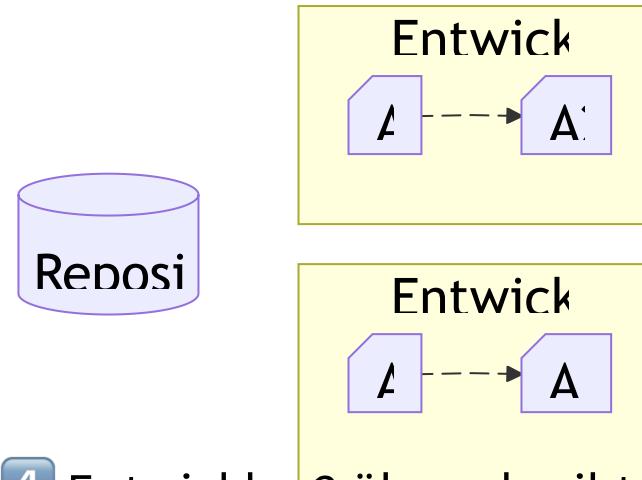
**1** 2 Entwickler lesen die gleiche Datei



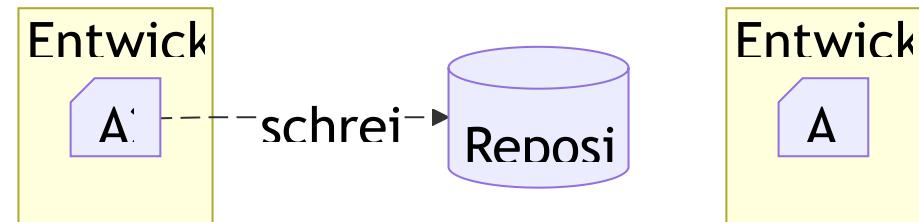
**3** Entwickler 1 veröffentlicht seine Änderungen zuerst



**2** Beide bearbeiten ihre jeweilige Kopie



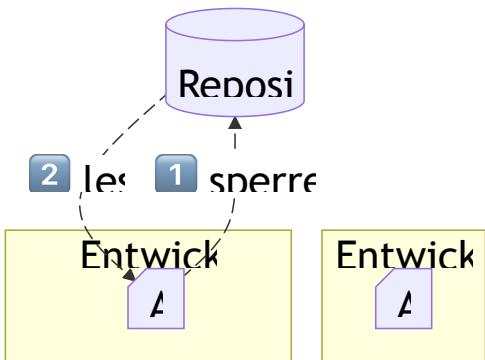
**4** Entwickler 2 überschreibt Änderungen von Entwickler 1



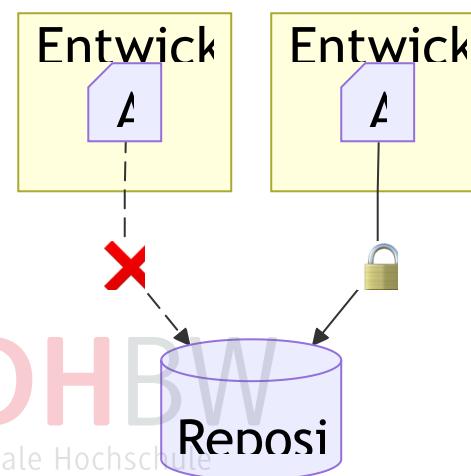
# GEMEINSAMES EDITIEREN

## LOCK-MODIFY-UNLOCK

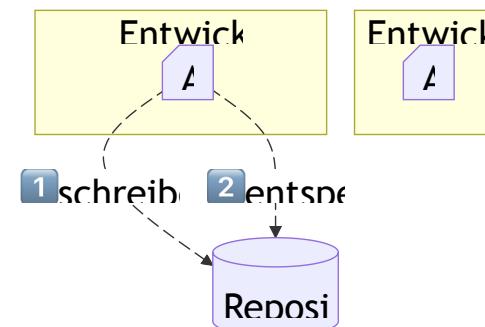
1 Entwickler 1 sperrt die Datei und kopiert sie zum bearbeiten



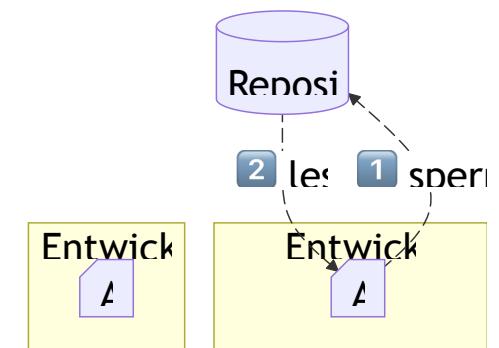
2 Während Entwickler 1 bearbeitet scheitert der Sperr-Versuch von Entwickler 2



3 Entwickler 1 schreibt seine Änderungen und hebt die Sperre auf



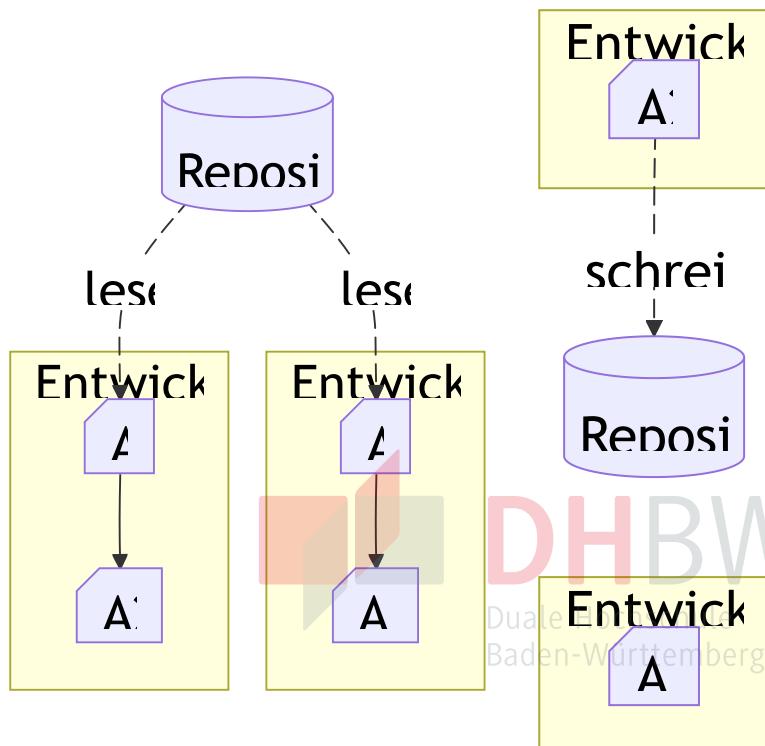
4 Jetzt kann Entwickler 2 seine Änderungen vornehmen



# GEMEINSAMES EDITIEREN

## COPY-MODIFY-MERGE

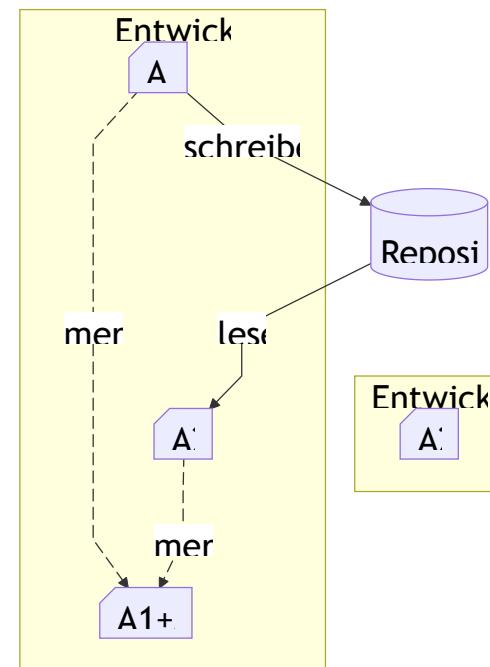
1 Beide Entwickler kopieren die gleiche Datei und bearbeiten diese

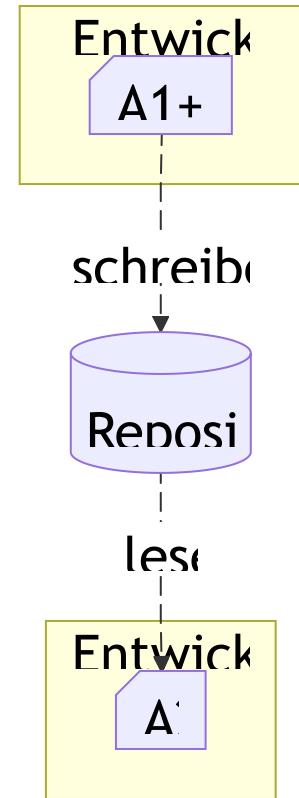


2 Entwickler 2 schreibt seine Änderung zuerst

3 Bevor Entwickler 1 schreiben kann, muss er erst die aktuelle Version übernehmen

4 Entwickler 1 veröffentlicht die neu Version, Entwickler 2 kopiert sie.





# KONFLIKTAUFLÖSUNG

- ✓ Überschneiden sich zwei Änderungen *nicht*, können sie *automatisch* zusammenfließen
- ⚠ Wenn sich Änderungen im Quellcode jedoch Überschneiden, müssen diese Konflikte *manuell* aufgelöst werden
- ✗ Damit ein Commit oder Merge erfolgreich ist, müssen diese Konflikte manuell aufgelöst werden.
- ⟳ Daher immer zuerst neueste Änderungen lokal integrieren, dann Konflikte auflösen und eine neue Revision erstellen.
- 👉 Zur direkten Auflösung markieren VCS die in Konflikt stehenden Codezeilen (*diff*).
- 💻 Außerdem werden *merge-tools* bereitgestellt, meist Texteditoren, in denen die markierten Codezeilen bearbeitet werden können.

# ARTEN VON VCS

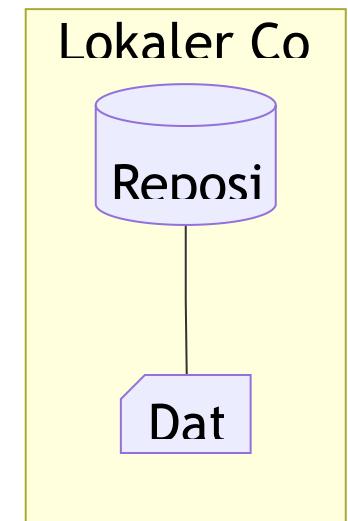
## LOKALE VCS

### EIGENSCHAFTEN

- Dateien und Repository befinden sich auf der selben Maschine
- Oft nur eine einzige Datei
- Teilweise können alter Versionen in der Datei selbst gespeichert sein
- Beispiele: SCCS, RCS

### VOR- UND NACHTEILE

- ✓ Einfach zu handhaben
- ✓ Für einfache Dokumente ausreichend
- ✗ Keine simultane Bearbeitung mit Anderen
- ✗ Keine Versionsverwaltung über mehrere Geräte hinweg



# ARTEN VON VCS

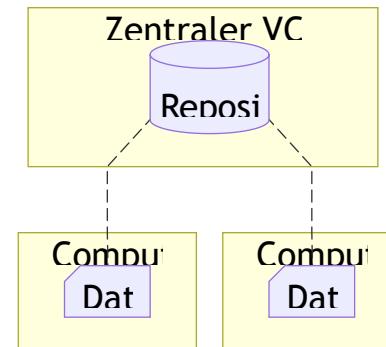
## ZENTRALE VCS

### EIGENSCHAFTEN

- Es gibt ein zentrale Repository
- Arbeitskopie und Repository befinden sich auf unterschiedlichen Maschinen
- Client-Server-Architektur
- Beispiele: CVS, SVN

### VOR- UND NACHTEILE

- ✓ Man arbeitet nie im Repository
- ✓ Einfache Rechte Verwaltung
- ✓ Ausfallsicherheit der lokalen Maschinen
- ✗ Netzwerkzugriff erforderlich
- ✗ Single Point of Failure
- ✗ Client muss Änderungen mit dem Server synchronisieren



# ARTEN VON VCS

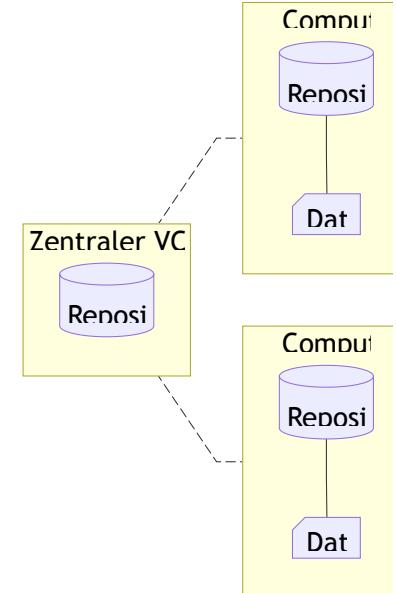
## DEZENTRALE VCS

### EIGENSCHAFTEN

- Zentrales und lokale Repositories
- Neue Versionen werden zunächst lokal erstellt und später zentral zusammengeführt
- Zusätzliche zentrale Repository-Hierarchien möglich (bei großen Projekten)
- Zusätzliche Operationen zum Austausch zwischen Repositories: pull und push
- Beispiele: Git, Mercurial, Bazaar

### VOR- UND NACHTEILE

- ✓ Offline-Arbeit möglich
- ✓ Schnellere Ausführzeit da lokal
- ✓ Keine Abhängigkeit von einzelner Maschine
- ✓ Änderungen können komplexe Hierarchien durchlaufen
- ✗ Repositories müssen zusammengeführt werden (komplizierte Konfliktauflösung)



# ORGANISATION VON VCS

- Je nach Team- und Projektgröße gibt es unterschiedliche Organisationsformen
- Die Organisation kann sich auf...
  - ...die Struktur des Repositories auswirken
  - ...die Art der Zusammenarbeit auswirken
  - ...Berechtigungskonzepte, unterschiedliche Zugriffsrechte und Rollen auswirken
- Wie genau die Organisation aussieht hängt von...
  - ...der Größe des Teams ab
  - ...der Art des Projektes ab
  - ...den Anforderungen an das Projekt ab
  - ...bestimmten Standards ab



# KONZEPTE UND BEGRIFFE

Begriff	Beschreibung
Repository	Ablage oder Archiv für versionskontrollierte Dateien
Revision / Commit	Eine Version (nach Übergabe einer Änderung)
Branch	Entwicklungszweig(e) des Projektes
Master / Trunk	Hauptentwicklungszweig
Branching	Abzweigen eines Entwicklungszweiges
Merging	Zusammenführen von Entwicklungszweigen oder Dateien
Forward Integration	Merge von Trunk in Branch
Reverse Integration	Merge von Branch in Trunk
Tag	Markierung einer bestimmten Version (z.B. als Release)
Fork	Abspaltung in ein neues unabhängiges Projekt
Merge Conflict	Konflikt bei Zusammenführung von Änderungen



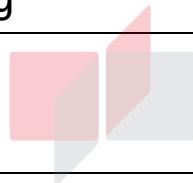


# GRUNDBEFEHLE VON VCS



**DHBW**  
Duale Hochschule  
Baden-Württemberg

Befehl	Beschreibung
Create / Repo	Erzeuge eine neue Ablage (Repository)
Add / Import	Füge eine Datei oder ein Dateisystem erstmalig der Ablage hinzu
Revision	Zeigt die aktuelle Version einer Datei oder eines Verzeichnisses
Checkout	Erzeuge eine lokale Arbeitskopie einer Datei aus der Ablage
Revert	Verwerfe lokale Änderungen und lade die letzte Version
UpdateSync / Sync	Aktualisiere deine lokale Arbeitskopie
Resolve	Zur Konfliktauflösung bei unterschiedlichen Dateiversionen
CommitCheckin / Checkin	Lade die Änderungen deiner lokalen Arbeitskopie in die Ablage
Status	Zeigt den aktuellen Stand: Änderungen, Konflikte etc.
Diff / Change / Delta	Zeige die Unterschiede zwischen zwei Versionen
Log / History / Changelog	Auflistung der Veränderungen
Branch	Erstelle eine separate Kopie des Projektes (neuer Entwicklungszweig)
Merge	Integriere die Änderungen eines Branch in einen anderen



# BEKANNTE VCS BEISPIELE

## APACHE SUBVERSION (SVN)



- Zentrales VCS
- Open Source
- Fortlaufende, strikte Revisionsnummer👉 Revisionskette
- 🛠 Tools: TortoiseSVN (Windows Explorer), Subclipse (Eclipse)

## GIT



- Dezentrales VCS, Open Source, sehr Mächtig
- Referenz auf Vorgängerversion anstatt Revisionsnummer
- Versionen können auf mehrere Vorgänger aufbauen
- Keine implizite Reihenfolge der Versionen👉 Gerichteter azyklischer Graph
- 🛠 Tools: TortoiseGit (Windows Explorer), GitKraken, EGit (Eclipse)



# GIT



# ARBEITEN MIT GIT

## LOCAL REPOSITORY

- 💡 In Git wird immer das komplette Remote-Repository in ein Local-Repository auf dem Computer des Entwicklers ausgecheckt (*cloning*).
- 💿 Das vollständige Abbild eines Repositories liegt lokal im *working directory*.
- ⚙️ Das Local-Repository besitzt alle Funktionalitäten eines VCSs:  
committing, branching, merging usw.
- 🔌 Man arbeitet offline; d.h. man commited in das lokale Repository.
- 🔄 Nach getaner Arbeit synchronisiert man alle Änderungen zwischen Remote- und Local-Repository.

# ARBEITEN MIT GIT

## WAS GEHÖRT IN'S REMOTE REPOSITORY

- Quellcode (z.B. `.java`, `.py`, `.c`, `.cpp`,  
`.js`)
- Textdateien (z.B. `.txt`, `.md`, `.json`, `.xml`,  
`.yaml`)
- Skripte (z.B. `.sh`, `.bat`, `.ps1`)
- Build-Skripte (z.B. `Makefile`,  
`build.gradle`, `pom.xml`)
- Dokumentation (z.B. `.md`)

- Build-Artefakte (z.B. `.class`, `.exe`, `.dll`,  
`.o`, `.so`)
- Abhängigkeiten (z.B. Bibliotheken,  
Frameworks)
- Konfigurationsdateien mit sensitiven  
Daten (z.B. Passwörter, API-Schlüssel)
- Temporäre Dateien (z.B. Log-Dateien,  
Cache-Dateien)
- IDE-spezifische Dateien (z.B. `.vscode`,  
`.idea`)
- Betriebssystem-spezifische Dateien (z.B.  
`.DS_Store`, `Thumbs.db`)

# **ARBEITEN MIT GIT**

## **GITIGNORE**



- Die Datei `.gitignore` legt fest, welche Dateien und Verzeichnisse von Git ignoriert werden sollen.
- Dies ist nützlich, um temporäre Dateien, Build-Artefakte oder sensible Informationen vom Repository auszuschließen.
- Die `.gitignore`-Datei wird im Stammverzeichnis des Repositories abgelegt und kann für jedes Projekt individuell angepasst werden.
- Beispielinhalte einer `.gitignore`-Datei:
  - ❑ `*.log` - Ignoriert alle Log-Dateien
  - ❑ `build/` - Ignoriert das gesamte Build-Verzeichnis
  - ❑ `*.class` - Ignoriert alle Java-Klassendateien
  - ❑ `.env` - Ignoriert Umgebungsdateien mit sensiblen Daten
  - ❑ `.vscode/` - Ignoriert VSCode-spezifische Einstellungen

 Es gibt viele vorgefertigte `.gitignore`-Dateien für verschiedene Programmiersprachen und Frameworks, die online verfügbar sind

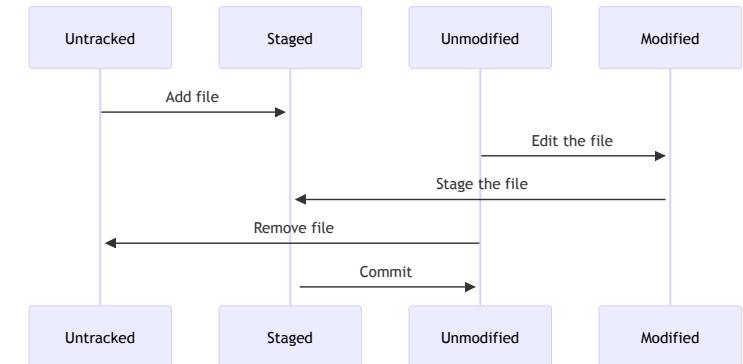
# ARBEITEN MIT GIT

## SYNCHRONISATION: UPLOAD

- **stage**: Alle Änderungen im Local-Repository, welche an das Remote-Repository gesendet werden sollen, müssen auf die Stage (engl.: Bühne) gebracht werden.
- **commit**: schiebt alle ausgewählten Änderungen auf die Stage.

Wichtig: Dies geschieht offline – anders als bei anderen VCS sendet commit die Änderungen nicht an den Server, sondern nur in das lokale Repository bzw. dessen Stage.

- **push**: sendet alle als 'staged' markierten Dateien an das Remote-Repository.



Lebenszyklus zustände von Dateien in Git

# ARBEITEN MIT GIT

## BRANCHING

- Branching bedeutet, die Software von der Hauptentwicklungsline abzuzweigen und daran zu arbeiten, ohne etwas am Hauptentwicklungs Zweig zu verändern.
- Im Gegensatz zu anderen VCS ermutigt Git zu einer Arbeitsweise mit häufigem Branching und Merging.
- Verwendung:
  - 👉 Für jedes Feature oder Bugfix ein neuer Branch
  - 👉 Wird danach in den Haupt-Entwicklungs Zweig integriert (Merge)
  - 👉 Branches zur Wartung älterer Versionen

⚠️ Branches müssen immer vor dem Arbeiten (vor der ersten Modifikation) angelegt werden

# ARBEITEN MIT GIT

## BRANCHING STRATEGIEN

- Es gibt verschiedene Strategien für das Arbeiten mit Branches in Git
  - Die Wahl der richtigen Strategie hängt von Teamgröße, Projektart und Deployment-Zyklen ab
  - Beliebte Ansätze:
    - Feature Branching
    - GitFlow
    - Trunk-Based Development
  - Keine universell "richtige" Strategie - abhängig vom Kontext
-  Weiterführende Informationen: [Git Branching Strategies Guide](#)

# ARBEIT MIT GIT

## MERGING SZENARIEN

- **Fast-forward merge**

Wenn der Branch, in den gemerged wird, keine neuen Commits hat, wird der Branch einfach auf den Ziel-Branch verschoben.

- **Three-way merge**

Wenn beide Branches neue Commits haben, wird ein Merge-Commit erstellt, der die Änderungen zusammenführt.

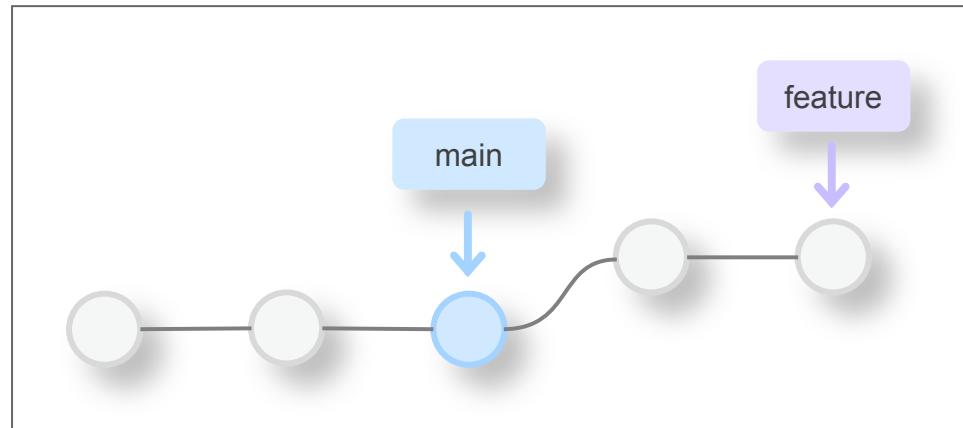
- **Rebase**

Eine alternative Methode zum Merging, bei der die Commits eines Branches auf einen anderen Branch angewendet werden, um eine lineare Historie zu erzeugen.

⚠ Rebase kann die Historie verändern und sollte mit Vorsicht verwendet werden, insbesondere bei gemeinsam genutzten Branches.

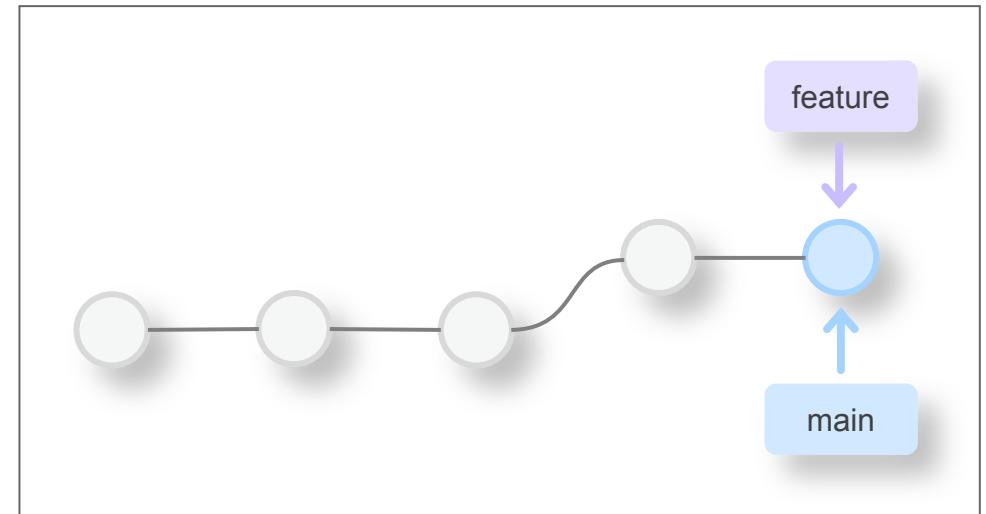
# ARBEIT MIT GIT

## FAST-FORWARD-MERGE



Vor dem Merge

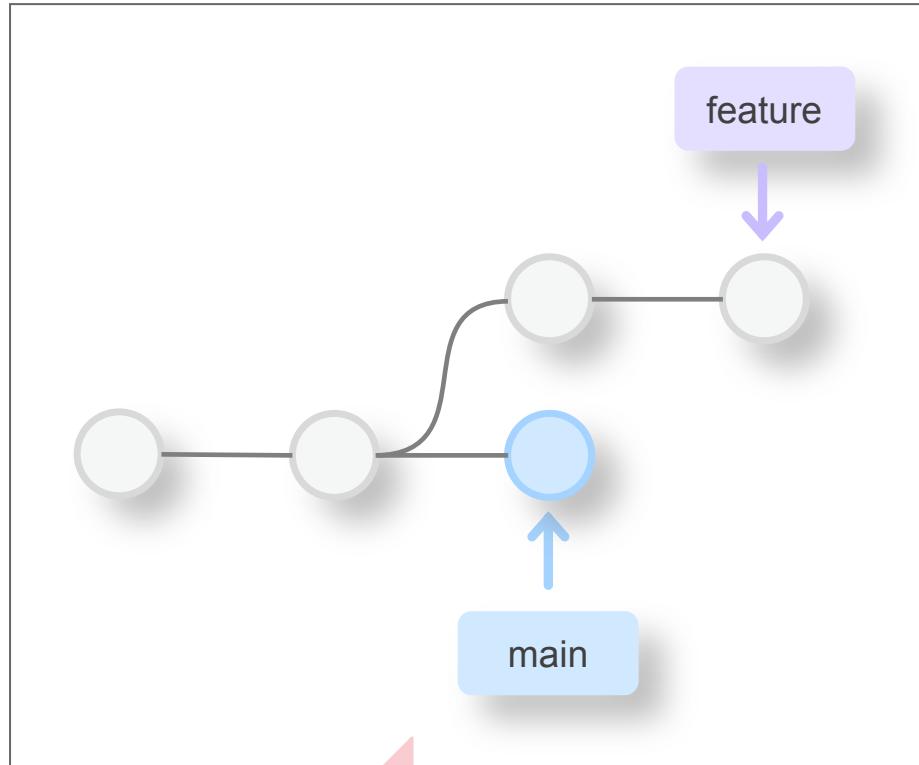
⚠ Fast-Forward-Merges sind nur möglich, wenn der Ziel-Branch keine neuen Commits hat.



Nach dem Merge

# ARBEIT MIT GIT

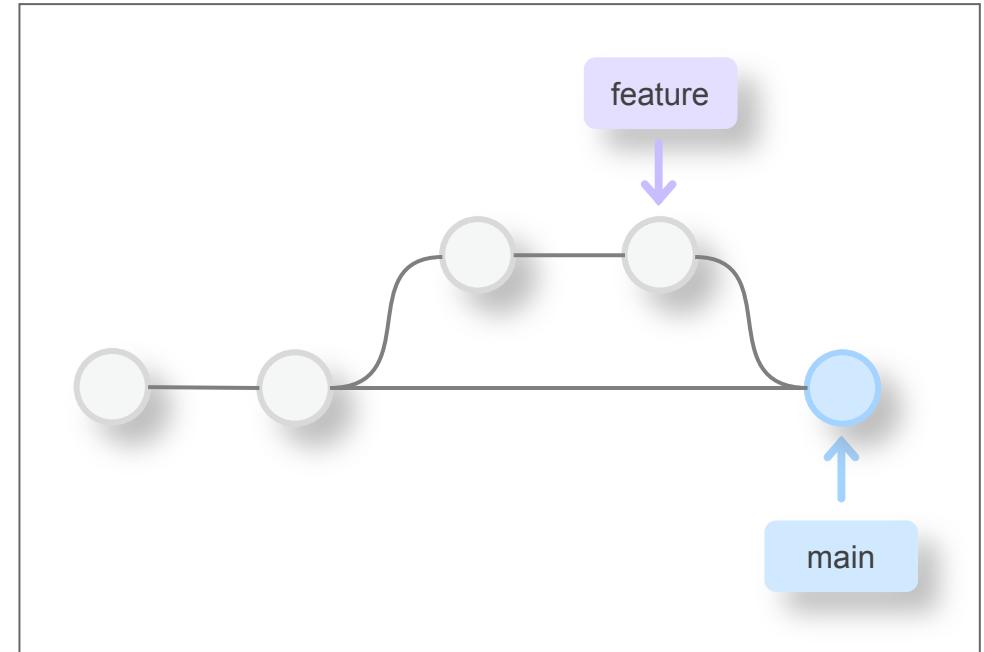
## THREE-WAY-MERGE



Vor dem Merge



**DHBW**  
Duale Hochschule  
Baden-Württemberg

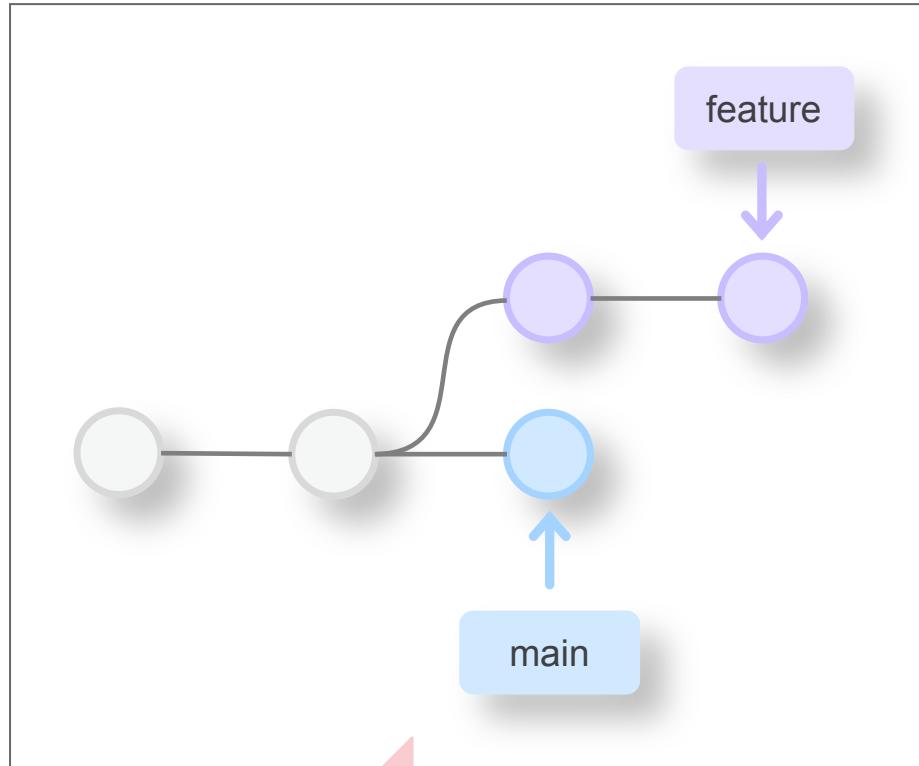


Nach dem Merge

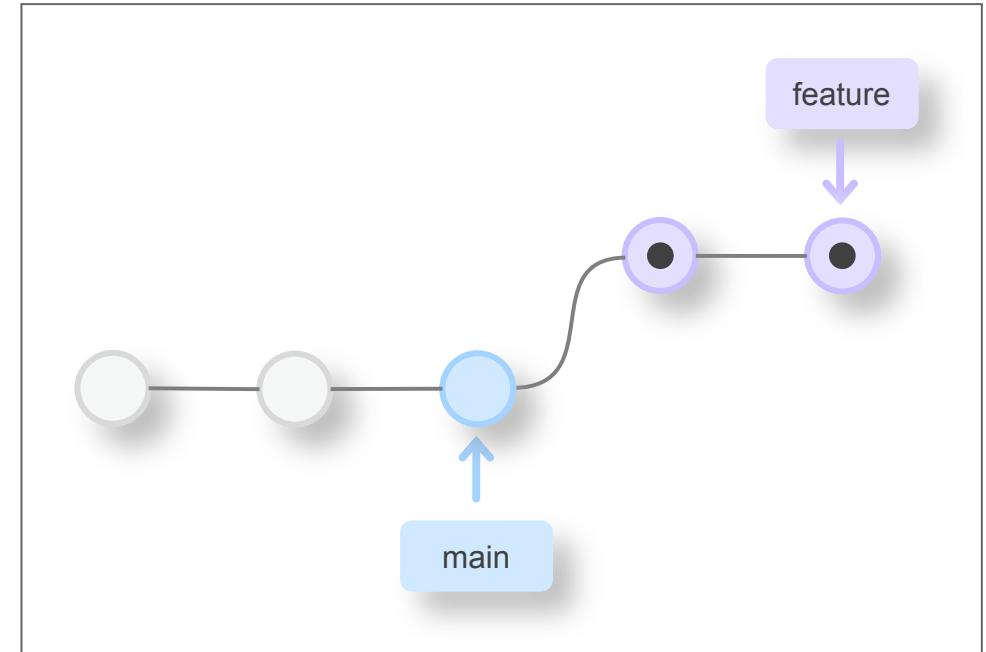
⚠ Three-Way-Merges sind notwendig, wenn beide Branches neue Commits haben.

# ARBEIT MIT GIT

## REBASE



Vor dem Rebase



Nach dem Rebase

⚠ Rebase verändert die Historie des Branches und sollte mit Vorsicht verwendet werden.



# ARBEIT MIT GIT

## WICHTIGSTE BEFEHLE

Git-Befehl	Beschreibung
<code>git clone &lt;URL&gt;</code>	Kopiert ein bestehendes Repository von einer URL auf den lokalen Rechner.
<code>git fetch</code>	Lädt die neuesten Änderungen aus dem Remote-Repository, ohne sie in den aktuellen Branch zu integrieren.
<code>git pull</code>	Lädt die neuesten Änderungen aus dem Remote-Repository und integriert sie in den aktuellen Branch.
<code>git push</code>	Überträgt lokale Commits in das Remote-Repository.
<code>git add &lt;Datei/Ordner&gt;</code>	Fügt geänderte Dateien zur Staging-Area hinzu, um sie für den nächsten Commit vorzubereiten.
<code>git commit -m "Nachricht"</code>	Erstellt einen neuen Commit mit den Dateien in der Staging-Area und einer Beschreibung der Änderungen.
<code>git status</code>	Zeigt den aktuellen Status des Repositories, inklusive uncommitteten Änderungen und Staging-Area.

Git-Befehl	Beschreibung
git checkout <Branch/Commit>	Wechselt zu einem anderen Branch oder einem bestimmten Commit.
git revert <Commit>	Macht einen bestimmten Commit rückgängig, indem ein neuer Commit erstellt wird, der die Änderungen umkehrt.
git diff	Vergleicht Änderungen zwischen Dateien, Staging-Area und Commits.

# ARBEITEN MIT GIT

## BEST PRACTICES

- 🎯 Jedes Code-Projekt sollte zumindest in einem lokalen Repository verwaltet werden:  
`git init` als erste "Amtshandlung"
- 💾 Nach jeder "lauffähigen" Änderung committen.
- 📝 Aussagekräftige Commit-Nachrichten schreiben: "Was wurde geändert" und "Warum"
- 🌿 Im Team: eigener Branch für jedes neue Feature oder Bug Fix
- 🎯 Nicht direkt in den `main/master` Branch committen
- ⟳ Regelmäßig `git pull` ausführen, um Konflikte zu vermeiden
- ⚠ Keine großen Binärdateien (Bilder, Videos, ZIPs, ...) committen: Verwendung von  
`.gitignore` für automatisches Ausschließen
- ✨ **Verwendung von Git zur Gewohnheit machen!**

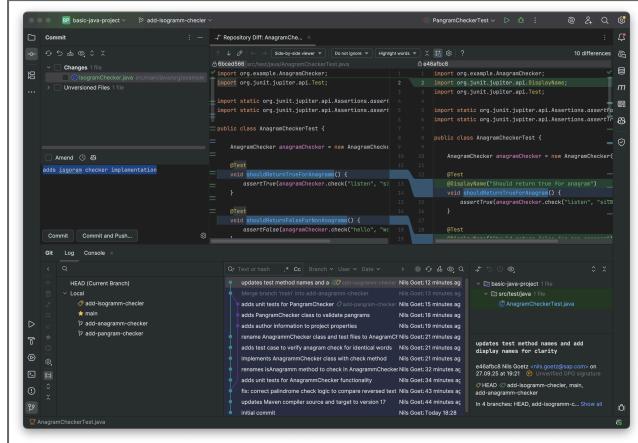
# ARBEITEN MIT GIT

## ÜBUNG - COMMIT MESSAGES

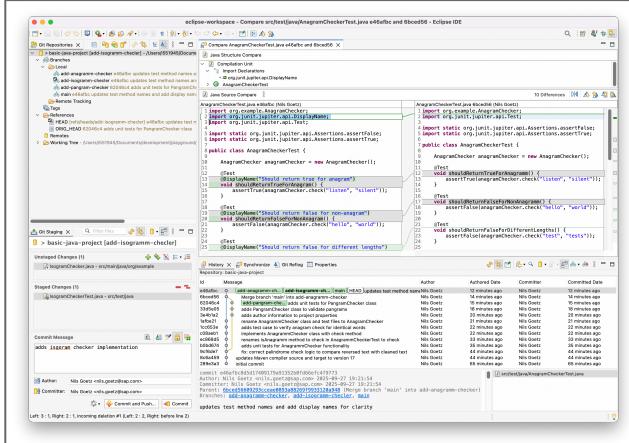
- 👉 fixes bug - ✗
- 👉 adds new function 'getName' for Animal class - ✓
- 👉 fixes null pointer exception in UserService when user is not found - ✓
- 👉 added some stuff - ✗
- 👉 updates README - ✗
- 👉 improve performance of data processing algorithm - ✓
- 👉 Add user authentication and update UI styles - ✗
- 👉 feat(login): implement user login functionality - ✓
- 👉 refactors DatabaseConnection to improve performance and readability - ✓

# GIT IN DER IDE

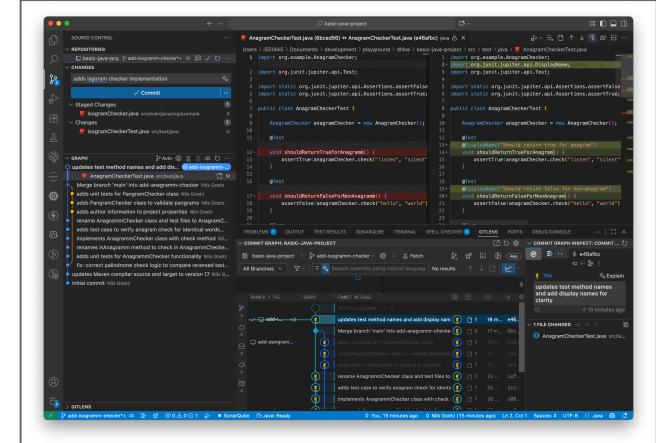
## INTELLIJ IDEA



## ECLIPSE IDE

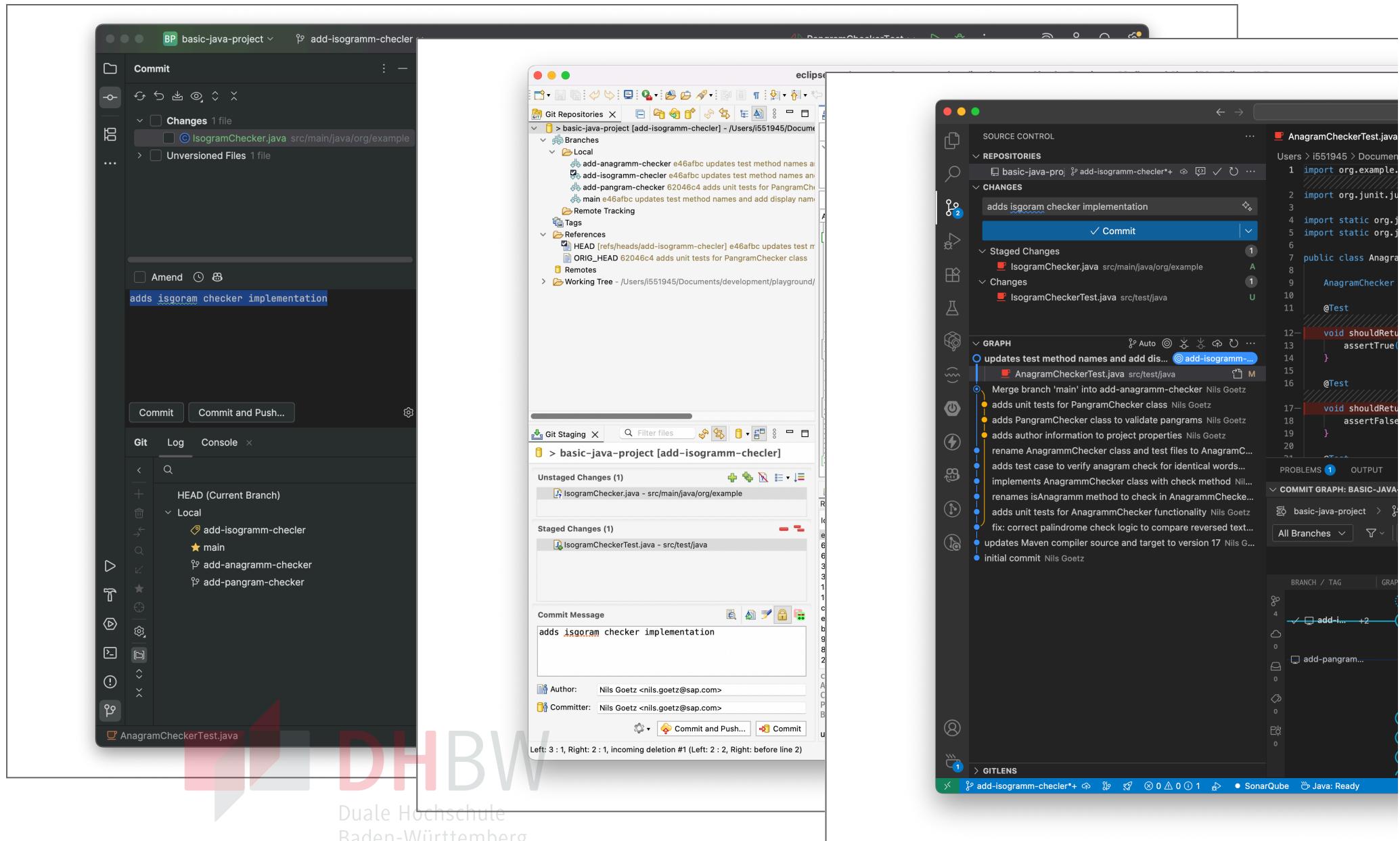


## VISUAL STUDIO CODE



# GIT IN DER IDE





# GIT IN DER IDE



A screenshot of a Git client interface showing a repository diff. The top bar shows the project name "basic-java-project" and the branch "add-isogramm-checker". The left sidebar shows a "Commit" section with a "Changes" list containing "IsogramChecker.java" and an "Unversioned Files" section. The main area displays a "Repository Diff: AnagramChe..." between commit "6bcd566" (src/test/java/AnagramCheckerTest.java) and commit "e46afbc8" (src/main/java/org/example/AnagramChecker.java). The diff highlights 10 differences, showing imports being moved from the main file to the test file.

```
diff --git a/src/main/java/org/example/AnagramChecker.java b/src/test/java/AnagramCheckerTest.java
--- a/src/main/java/org/example/AnagramChecker.java	2023-09-15 14:45:12.000+02:00
+++ b/src/test/java/AnagramCheckerTest.java	2023-09-15 14:45:12.000+02:00
@@ -1,5 +1,5 @@
-import org.example.AnagramChecker;
-import org.junit.jupiter.api.Test;
-import static org.junit.jupiter.api.Assertions.assertFalse;
-import static org.junit.jupiter.api.Assertions.assertTrue;
+import org.example.AnagramChecker;
+import org.junit.jupiter.api.DisplayName;
+import org.junit.jupiter.api.Test;
+
+import static org.junit.jupiter.api.Assertions.assertFalse;
+import static org.junit.jupiter.api.Assertions.assertTrue;
```

A screenshot of the Eclipse IDE showing a "Compare" view. The title bar says "eclipse-workspace - Compare src/test/java/AnagramCheckerTest.java e46afbc and 6bcd56 - Eclipse IDE". The left sidebar shows "Git Repositories" with a local repository "basic-java-project [add-isogramm-checker]". The main area has two tabs: "Java Structure Compare" and "Java Source Compare". The "Java Source Compare" tab shows the difference between "AnagramCheckerTest.java e46afbc (Nils Goetz)" and "AnagramCheckerTest.java 6bcd56 (Nils Goetz)". It highlights 10 differences, showing the addition of import statements and the move of assertions from the main file to the test file.

```
diff --git a/src/main/java/org/example/AnagramChecker.java b/src/test/java/AnagramCheckerTest.java
--- a/src/main/java/org/example/AnagramChecker.java	2023-09-15 14:45:12.000+02:00
+++ b/src/test/java/AnagramCheckerTest.java	2023-09-15 14:45:12.000+02:00
@@ -1,5 +1,5 @@
-import org.example.AnagramChecker;
-import org.junit.jupiter.api.Test;
-import static org.junit.jupiter.api.Assertions.assertFalse;
-import static org.junit.jupiter.api.Assertions.assertTrue;
+import org.example.AnagramChecker;
+import org.junit.jupiter.api.DisplayName;
+import org.junit.jupiter.api.Test;
+
+import static org.junit.jupiter.api.Assertions.assertFalse;
+import static org.junit.jupiter.api.Assertions.assertTrue;
```

A screenshot of a modern Git client, likely GitHub Desktop, showing a commit graph and a detailed commit view. The top bar shows the project name "basic-java-project". The left sidebar shows "REPOSITORIES" and "CHANGES" sections. The main area shows a commit titled "adds isogram checker implementation" with a "Commit" button. Below it, a "GRAPH" section shows the commit's dependencies and history. The right side shows the diff between the commit "6bcd56" and "e46afbc" for the file "AnagramCheckerTest.java". The diff highlights 15 changes, including the addition of imports and the movement of assertions. A "GITLENS" tab is also visible at the bottom.

```
diff --git a/src/main/java/org/example/AnagramChecker.java b/src/test/java/AnagramCheckerTest.java
--- a/src/main/java/org/example/AnagramChecker.java	2023-09-15 14:45:12.000+02:00
+++ b/src/test/java/AnagramCheckerTest.java	2023-09-15 14:45:12.000+02:00
@@ -1,5 +1,5 @@
-import org.example.AnagramChecker;
-import org.junit.jupiter.api.Test;
-import static org.junit.jupiter.api.Assertions.assertFalse;
-import static org.junit.jupiter.api.Assertions.assertTrue;
+import org.example.AnagramChecker;
+import org.junit.jupiter.api.DisplayName;
+import org.junit.jupiter.api.Test;
+
+import static org.junit.jupiter.api.Assertions.assertFalse;
+import static org.junit.jupiter.api.Assertions.assertTrue;
```

✓ add-isogramm-checker → add-anagramm-checker

Merge branch 'main' into add-anagramm-checker

updates test method names and add display nam... 1 16 m... e4b...

adds unit tests for PangramChecker class 1 17 m... 6bc...

adds PangramChecker class to validate pangram 1 19 m... 620...

adds author information to project properties 1 22 ... 33d...

rename AnagrammChecker class and test files to 3 25 ... 1af...

adds test case to verify anagram check for identi 1 25 ... 1cc...

implements AnagrammChecker class with check 1 26 ... c08...

1 FILE CHANGED +0 ~1 -0

AnagramCheckerTest.java src/test/java



# GITHUB

## WAS IST GITHUB?

- GitHub ist eine **webbasierte Hosting-Plattform** für Git-Repositories
- Bietet **kollaborative Entwicklung** mit Features wie Issues, Pull Requests und Code Reviews
- Ermöglicht **öffentliche und private** Projekt-Repositories
- De-facto Standard für **Open-Source-Projekte** und moderne Softwareentwicklung
- Gehört seit 2018 zu **Microsoft**

## 🎯 KERNFUNKTIONEN

- Repository Hosting
- Issue & Bug Tracking
- Pull Requests
- Code Reviews
- GitHub Actions (CI/CD)
- Project Management

# CLEAN CODE



# HERAUSFORDERUNGEN IN DER SOFTWAREENTWICKLUNG

- 📚 Komplexität von Software
- ⌚ Wartbarkeit und Erweiterbarkeit
- 💰 Technical Debt und steigende Kosten
- 🐞 Unklare oder widersprüchliche Anforderungen
- 🧹 Nachträgliches Bereinigen und Aufräumen von Quellcode (Refactoring)

# SOFTWARE QUALITÄT



Software Qualität umfasst verschiedene Aspekte, die sicherstellen, dass eine Softwarelösung den Anforderungen entspricht und zuverlässig funktioniert. Dazu gehören:

Qualitätsmerkmal	Beschreibung
------------------	--------------

---

<input checked="" type="checkbox"/> <b>Funktionalität</b>	Erfüllt die Software die Anforderungen?
---	---

	Zuverlässigkeit
Wie oft treten Fehler auf?	

	Performance
Wie schnell reagiert die Software?	

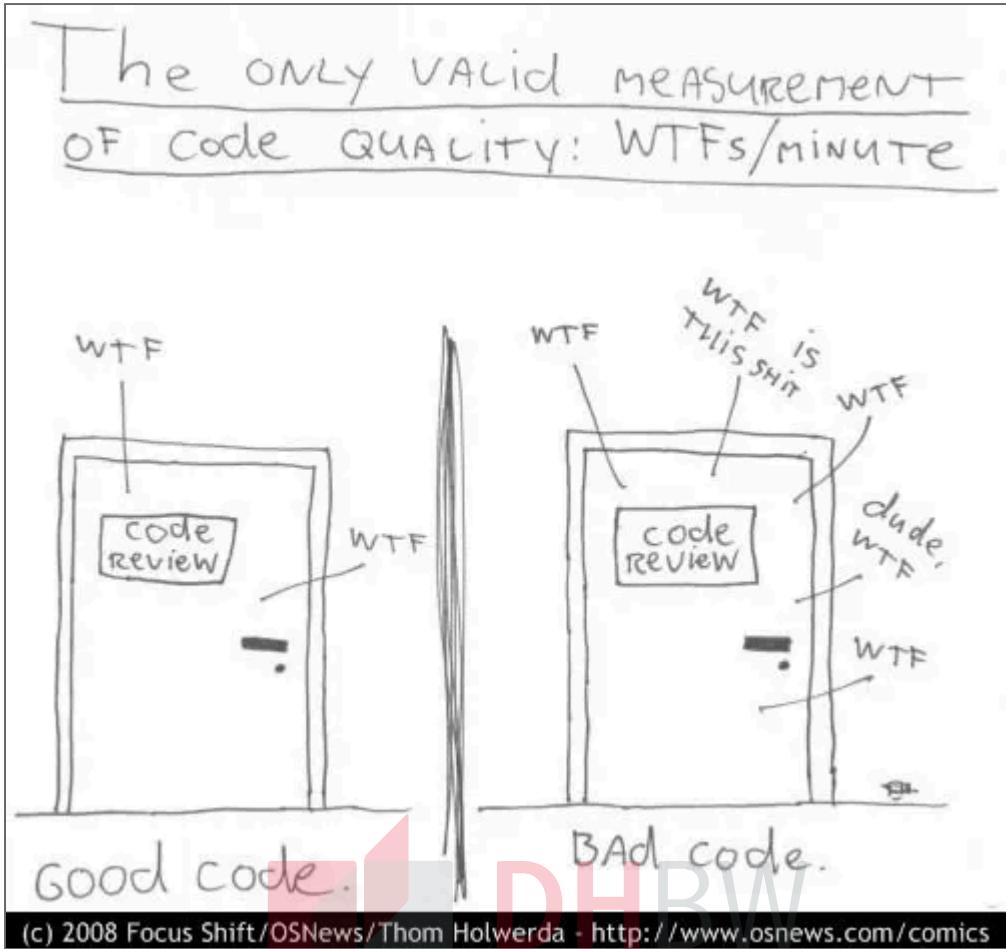
	Sicherheit
Wie gut ist die Software gegen Angriffe geschützt?	

	Wartbarkeit
Wie einfach ist es, den Code zu verstehen, anzupassen und zu erweitern?	

	Testbarkeit
Wie leicht lassen sich Tests schreiben und ausführen?	

👉 ISO 25010: Ein internationaler Standard, der Qualitätsmerkmale für Software definiert.

# WAS IST CLEAN CODE?



# WAS IST CLEAN CODE?

🎯 **"Clean code does one thing well."**

- Bjarne Stroustrup, inventor of C++

📖 **"Clean code is simple and direct. Clean code reads like well-written prose."**

- Grady Booch, author of Object Oriented Analysis

🤝 **"Clean code can be read, and enhanced by a developer other than its original author."**

- Dave Thomas, godfather of the Eclipse strategy

🖌️ **"Clean code always looks like it was written by someone who cares."**

- Michael Feathers, author of Working with Legacy Code

🤔 **"Code never lies, comments sometimes do"**

- Ron Jeffries, author of Extreme Programming books

Quelle: Clean Code by Robert C. Martin

## WIE VERBRINGEN ENTWICKLER IHRE ZEIT?

R. Minelli, A. Mocci, and M. Lanza, “I know what you did last summer - an investigation of how developers spend their time,” in *23rd ICPC*, 2015.

X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, “Measuring program comprehension: A large-scale field study with professionals,” *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 951-976, 2017.

👉 Entwickler verbringen einen Großteil ihrer Zeit (bis zu 60%) mit dem Lesen und Verstehen von Code.

## READABILITY

Lesbarkeit ist ein entscheidendes Merkmal von Clean Code. Gut lesbarer Code erleichtert es Entwicklern, den Code zu verstehen, zu warten und zu erweitern. Wichtige Aspekte der Lesbarkeit sind:

-  Klar und verständlich
-  Gut strukturiert und organisiert
-  Aussagekräftige Namen für Variablen, Funktionen und Klassen
-  Konsistente Formatierung und Einrückung
-  Angemessene Kommentare zur Erklärung von komplexem Code

## READABILITY - NAMING

Was macht der folgende Code?

● ● ●

```
1 public double discP(double amt, int cStat, int y) {  
2     double p = 0.0;  
3  
4     if (cStat == 1) {  
5         p = amt * 0.05;  
6     } else if (cStat == 2 && y > 2) {  
7         p = amt * CUtils.specD;  
8     } else if (cStat == 3) {  
9         double specDisPr = amt * 0.15;  
10        if (amt > 1000) specDisPr += 50;  
11        p = specDisPr;  
12    }  
13    return p;  
14 }
```

# READABILITY - NAMING

Was macht der folgende Code?

```
● ● ●  
1  public double calculateDiscountPrice(double purchaseAmount, int  
customerStatus, int yearsAsCustomer) {  
2      double price = 0.0;  
3  
4      if (customerStatus == 1) {  
5          price = purchaseAmount * 0.05;  
6      } else if (customerStatus == 2 && yearsAsCustomer > 2) {  
7          price = purchaseAmount * Customer.SILVER_CUSTOMER_DISCOUNT;  
8      } else if (customerStatus == 3) {  
9          double goldCustomerDiscount = purchaseAmount * 0.15;  
10         if (purchaseAmount > 1000) goldCustomerDiscount += 50;  
11         price = goldCustomerDiscount;  
12     }  
13  
14     return price;  
15 }
```



# READABILITY - NAMING CONVENTIONS

-  Verwende aussagekräftige und beschreibende Namen
-  Verwende englische Namen für bessere internationale Zusammenarbeit
-  Gib Einheiten in Variablennamen an (z.B. `delayInSeconds`, `weightInKg`, `priceInEuros`)  
Oder noch besser: Verwende Typen, die Einheiten repräsentieren (z.B. `Duration`, `Weight`, `Money`)
-  Verwende Verben für Funktionen/Methoden (z.B. `calculateTotal`, `validateInput`)
-  Verwende Nomen für Klassen/Objekte (z.B. `Customer`, `OrderProcessor`)
-  Ein Wort pro Konzept (z.B. `fetch` ODER `retrieve` ODER `get` - nicht alle drei)
-  Keine Angst vor langen Namen! Moderne IDEs nutzen code completion
-  Vermeide Abkürzungen, es sei denn, sie sind allgemein bekannt
-  Verwende konsistente Namenskonventionen (z.B. `camelCase`, `PascalCase`)
-  Vermeide Noise Words (z.B. `Data`, `Utils`, `Info`, `Manager`, `Helper`)
-  Namen sollten aussprechbar sein (z.B. `0cmTx` ist schlecht)
-  Namen sollten suchbar sein (z.B. `i` oder `7` sind schlechte Namen)

# NAMING CONVENTIONS - ÜBUNG

Gibt es Verbesserungsbedarf bei den folgenden Namen?

Code



`int d;`



`int day0fTheMonth;`



`class PerformAgeCheck {...}`



`public void awaitConfirmation(int maxWaitTime) {...}`



`public void validateInput(String input) {...}`



`class TextActionsHelper {...}`



`List<User> users;`



# READABILITY - FORMATTING CONVENTIONS



```
1 boolean isAllowedToDrive(Person person) {  
2     if (person.getAge() >= 18)  
3     {  
4         if (person.hasValidLicense()) {  
5             if(person.isSober()) {  
6                 Logger.log("The Person is sober has a valid drivers license and is  
therefore by all means allowed to drive.");  
7                 return true;  
8             } else {  
9                 Logger.log("Person is not sober.");  
10                return false;  
11            }  
12        } else {  
13            Logger.log("Person does not have a license.");  
14            return false;  
15        }  
16    }  
17    else {  
18        Logger.log("Person is underaged."); return false;  
19    }  
20}
```





# READABILITY - FORMATTING CONVENTIONS



```
1 boolean isAllowedToDrive(Person person) {  
2     if (person.getAge() < 18) {  
3         Logger.log("Person is underaged.");  
4         return false;  
5     }  
6     if (!person.hasValidLicense()) {  
7         Logger.log("Person does not have a valid license.");  
8         return false;  
9     }  
10    if (!person.isSober()) {  
11        Logger.log("Person is not sober.");  
12        return false;  
13    }  
14    return true;  
15 }
```

# READABILITY - FORMATTING CONVENTIONS

- ➡ Konsistente Einrückung und Formatierung
- 📝 Eine Anweisung pro Zeile für bessere Lesbarkeit
- ➡ Begrenze Zeilenlänge (z.B. 80-120 Zeichen)
- ⬆ Vermeide zu tiefe Verschachtelung (max. 3-4 Ebenen)
- ⬆ Nutze Leerzeilen zur Strukturierung von Codeblöcken
- abc Konsistente Klammerstil (z.B. K&R, Allman)
- 1234 Konsistente Verwendung von Leerzeichen (z.B. um Operatoren, nach Kommas)
- 📚 Verwende Code-Formatierungswerzeuge (z.B. Prettier, Black, clang-format)
- 🔍 Nutze Linting-Tools zur automatischen Überprüfung von Code-Standards (z.B. ESLint, Pylint, Checkstyle)
- 📋 Befolge etablierte Styleguides (z.B. [Google Java Style Guide](#), PEP 8 für Python)

## READABILITY - KOMMENTARE



```
1 /**
2 * Returns the day of the month.
3 * @return the day of the month (1-31)
4 */
5 public int getDayOfMonth() {
6     return dayOfMonth; // returns the day of the month
7 }
```



```
1 // customer must be older than 18 and either a customer for longer than 5 years
2 // or possess a pro membership
3 return (customer.getYearsAsCustomer() > 5 || customer.hasProMembership())
4     && customer.getAge() > 18;
```

## READABILITY - KOMMENTARE

● ● ●

```
1 public int getDayOfMonth() {  
2     return dayOfMonth;  
3 }
```

● ● ●

```
1 boolean isLongTimeCustomer = customer.getYearsAsCustomer() > 5;  
2 boolean isAdult = customer.getAge() > 18;  
3 boolean hasProMembership = customer.hasProMembership();  
4 return isAdult && (isLongTimeCustomer || hasProMembership);
```

## READABILITY - KOMMENTARE

-  Kommentare sollten den Code erklären, nicht wiederholen!
-  Kommentare Lügen, Code nicht
-  Halte Kommentare aktuell und konsistent mit dem Code
-  Nutze Dokumentationskommentare für öffentliche APIs (z.B. Javadoc, docstrings)
-  Vermeide "to-do" Kommentare - nutze stattdessen Issue-Tracker
-  Kommentiere Annahmen, Einschränkungen und bekannte Probleme

# DEBUGGING



# EXCEPTIONS IN JAVA

- Als Exception bezeichnet man im Allgemeinen:
  - 👉 ein Ereignis, das den normalen Ablauf eines Programms unterbricht
  - 👉 ein Objekt, das Informationen über dieses Ereignis enthält
- Tritt eine Exception auf, so ist es dem Programm überlassen, ...
  - ✓ die Exception zu behandeln
  - ⬆ die Exception an den aufrufenden Code weiterzugeben
  - ✗ das Programm zu beenden
- In Java sind Exceptions im Gegensatz zu Errors in der Regel behebbar bzw. behandelbar

# EXCEPTIONS IN JAVA

## CHECKED EXCEPTIONS

- werden während der Kompilierzeit überprüft
- Der Compiler erzwingt, dass diese Exceptions entweder behandelt oder weitergegeben werden
- Beispiele für Checked Exceptions:
  - 👉 IOException
  - 👉 SQLException
  - 👉 ClassNotFoundException

## UNCHECKED EXCEPTIONS

- treten zur Laufzeit auf und werden nicht vom Compiler überprüft
- Diese Exceptions können unbehandelt bleiben, ohne dass der Compiler einen Fehler meldet
- Beispiele für Unchecked Exceptions:
  - 👉 NullPointerException
  - 👉 ArrayIndexOutOfBoundsException
  - 👉 IllegalArgumentException

# EXCEPTIONS WERFEN

Exceptions können mit dem Schlüsselwort `throw` geworfen werden:



```
1 public void doSomethingDangerous(Object someObject) throws
IllegalArgumentException {
2     if (someObject.getValue() == null) {
3         throw new IllegalArgumentException("Wert fehlt!");
4     }
5     // do something dangerous here safely
6 }
```

# EXCEPTIONS BEHANDELN

Exceptions können mit dem Schlüsselwort `try-catch-finally` behandelt werden:

```
● ● ●  
1 try {  
2   // ⚠ Methode, die eine Exception werfen könnte  
3   doSomethingDangerous(someObject);  
4 } catch (IllegalArgumentException e) {  
5   System.out.println("Falsches Argument: " + e.getMessage());  
6   // ⬆ Behandlung der IllegalArgumentException  
7 } catch (Exception e) {  
8   // ⬆ Behandlung aller anderen Exceptions  
9   System.out.println("Sonstiger Fehler: " + e.getMessage());  
10 } finally {  
11   // ⬆ wird immer ausgeführt, egal ob eine Exception aufgetreten ist oder  
12   // nicht  
13 }
```

## CHECKED EXCEPTIONS BEHANDELN

Checked Exceptions müssen entweder behandelt (try-catch-block) oder weitergegeben werden:



```
1 public String readLineFromFile(String filePath) throws IOException {  
2     BufferedReader reader = new BufferedReader(new FileReader(filePath));  
3     String line = reader.readLine(); // ⚠ IOException könnte hier auftreten  
4     reader.close();  
5     return line;  
6 }
```

## CUSTOM EXCEPTIONS

Eigene Exceptions können erstellt werden, um spezifische Fehlersituationen zu modellieren:



```
public class InsufficientFundsException extends Exception {  
    public InsufficientFundsException(String message) {  
        super(message);  
    }  
}
```

# EXCEPTIONS BEST PRACTICES

## ✓ DO'S

- **Fail Early:** Exceptions sofort werfen, wenn ein Problem erkannt wird
- **Spezifische Exceptions:** Konkrete Exception-Typen verwenden
- **Aussagekräftige Nachrichten:** Detaillierte Fehlerbeschreibungen
- **Exceptions dokumentieren:** JavaDoc für geworfene Exceptions
- **Finally-Block:** Für Ressourcen-Cleanup verwenden

## ✗ DON'TS

- **Don't Fail Silently:** Niemals Exceptions "verschlucken"
- **Keine Exception für Kontrollfluss:** Exceptions sind keine if-Statements
- **Catch-All vermeiden:** Nicht alle Exceptions pauschal fangen
- **Keine leeren Catch-Blöcke:** Immer angemessen reagieren
- **Stack Trace nicht verlieren:** Original Exception weitergeben

Weiterführende Informationen:

- [OWASP \(Open Worldwide Application Security Project\) Error Handling Guide](#)
- [Blog Post: Mastering Error Handling: A Comprehensive Guide](#)
- [Blog Post: Best Practices of Error Handling in Software Development](#)

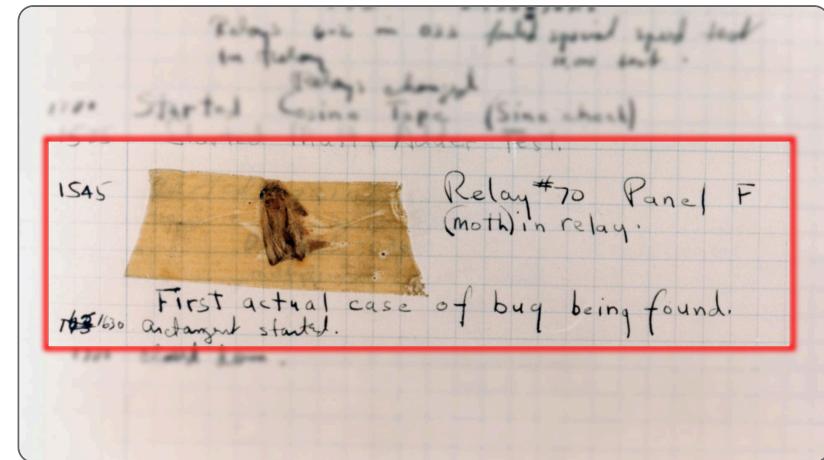


**DH**BW  
Duale Hochschule  
Baden-Württemberg

# DEBUGGING

## WAS IST DEBUGGING?

- **Definition:** Der Prozess des Auffindens und Behebens von Fehlern (Bugs) in Software
- **Etymologie:** Begriff geht zurück auf Grace Hopper (1947)
- **Ziel:** Fehlerhafte Programme zum korrekten Verhalten bringen
- **Systematischer Ansatz:** Fehler reproduzieren, lokalisieren und beheben



Der erste "Bug" - 1947

# DEBUGGER

## SINN UND ZWECK EINES DEBUGGERS

- Fehlerarten in Programmen:

- Fehlerarten in Programmen:
  - 🐛 Falsches Ergebnis / unerwartetes Verhalten
  - 💥 Programmabsturz durch Exceptions
  - ⟳ Endlosschleifen
  - ⚠️ Unbehandelte Benutzereingaben

- Debugger-Funktionen:

- Debugger-Funktionen:
  - 🔍 Schrittweise Code-Analyse (Step Into/Over/Out)
  - 📊 Inspektion von Variableninhalten zur Laufzeit
  - ⏸ Breakpoints für gezielte Programmunterbrechung
  - 📞 Analyse der Aufrufhierarchie (Call Stack)
  - 👁️ Watch-Expressions für kontinuierliche Überwachung
- Qualitätsmaß: "Defect Density" (Bugs pro KLOC - 1000 Lines of Code)

# LOGGING VS DEBUGGING



Debugger

`System.out.println()`



**DHBW**  
Duale Hochschule  
Baden-Württemberg

# LOGGING VS DEBUGGING

## LOGGING

-  **Produktive Systeme:** Überwachung im laufenden Betrieb
-  **Fehleranalyse:** Nach dem Auftreten von Problemen
-  **Audit-Trail:** Wer hat wann was gemacht?
-  **Alerting:** Automatische Benachrichtigungen bei kritischen Fehlern
-  **Persistierung:** Logs werden (dauerhaft) gespeichert

 **Wichtiger Tipp:** Verwendet die Debugging-Tools eurer IDE statt `System.out.println()` oder `console.log()` für die Fehlersuche!

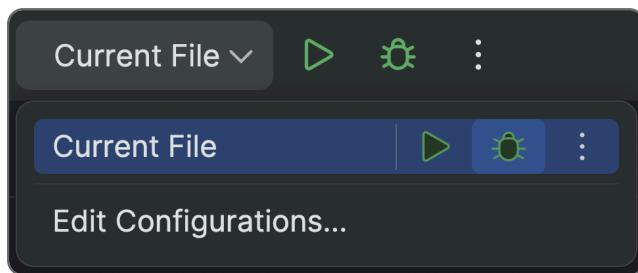
## DEBUGGING

-  **Entwicklungszeit:** Aktive Fehlersuche beim Programmieren
-  **Gezieltes Untersuchen:** Spezifische Bugs reproduzieren und beheben
-  **Live-Inspektion:** Variablen und Zustand in Echtzeit prüfen
-  **Programmfluss verstehen:** Schritt-für-Schritt Code-Durchlauf
-  **Hypothesen testen:** "Was passiert, wenn...?"

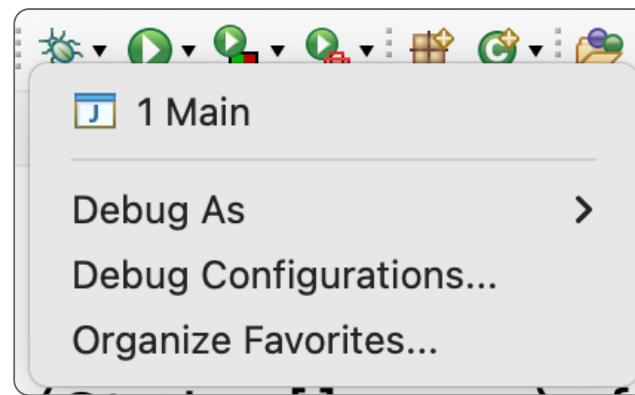
# DEBUGGER IN DER IDE

Starten des Debuggers in den verschiedenen IDEs:

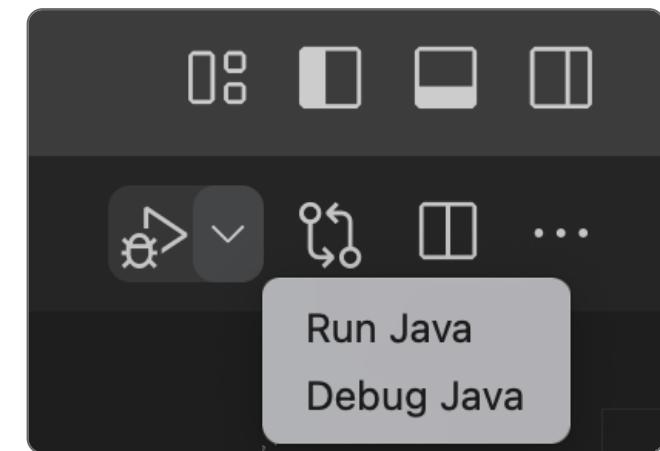
## INTELLIJ IDEA



## ECLIPSE IDE



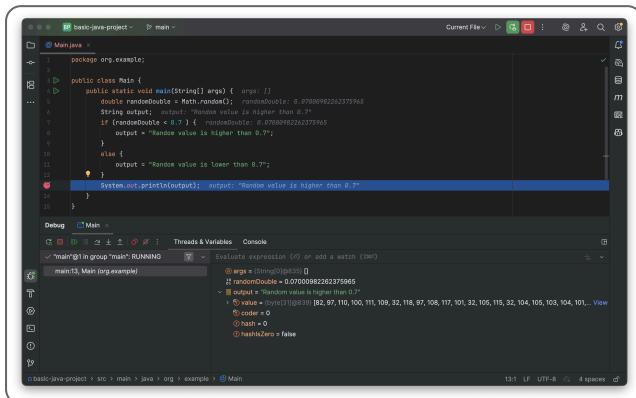
## VISUAL STUDIO CODE



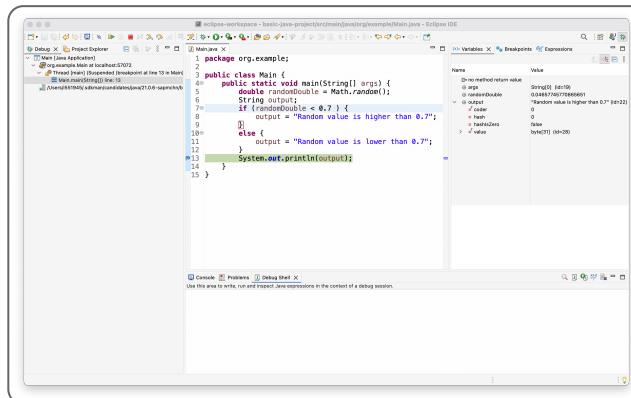
# DEBUGGER IN DER IDE

Die Debug-Ansicht in den verschiedenen IDEs:

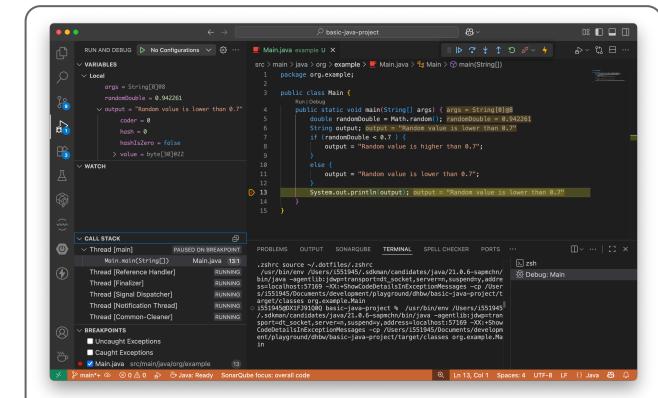
INTELLIJ IDEA



ECLIPSE IDE



VISUAL STUDIO CODE



# DEBUGGER IN DER IDE



The screenshot shows a Java development environment with three main panes:

- Left Pane:** Project Explorer showing a single file, Main.java, under the package org.example. The code prints a random double value between 0.0 and 1.0.
- Middle Pane:** Debug perspective showing a suspended thread at line 13. The variable output is set to "Random value is lower than 0.7".
- Right Pane:** Variables, Watch, and Call Stack toolbars. The Variables pane shows local variables: args (String[0]), randomDouble (double), output (String), coder (int), hash (int), hashIsZero (boolean), and value (byte[30]). The Watch pane shows the expression output. The Call Stack pane lists threads: Thread [main] (Paused on breakpoint), Thread [Reference Handler], Thread [Finalizer], Thread [Signal Dispatcher], Thread [Notification Thread], and Thread [Common-Cleaner].

**Code in Main.java:**

```
package org.example;

public class Main {
    public static void main(String[] args) {
        double randomDouble = Math.random();
        String output;
        if (randomDouble < 0.7) {
            output = "Random value is lower than 0.7";
        } else {
            output = "Random value is higher than 0.7";
        }
        System.out.println(output);
    }
}
```

**Terminal Output:**

```
.zshrc source ~/dotfiles/.zshrc
/usr/bin/env /Users/i551945/.sdkman/candidates/java/21.0.6-sapmchn/bin/java -agentlib:jdwp=transport=dt_socket,server=n,address=localhost:57072 -XX:+ShowCodeDetailsInExceptionMessages
i551945@DX1FJ91Q0Q basic-java-project % /usr/bin/env ./sdkman/candidates/java/21.0.6-sapmchn/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:57072 -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/i551945/Documents/development/playground/dhbw/basic-classes org.example.Main
```

# DEBUGGER IN DER IDE

Step Controls in den verschiedenen IDEs:

**INTELLIJ IDEA**



**ECLIPSE IDE**



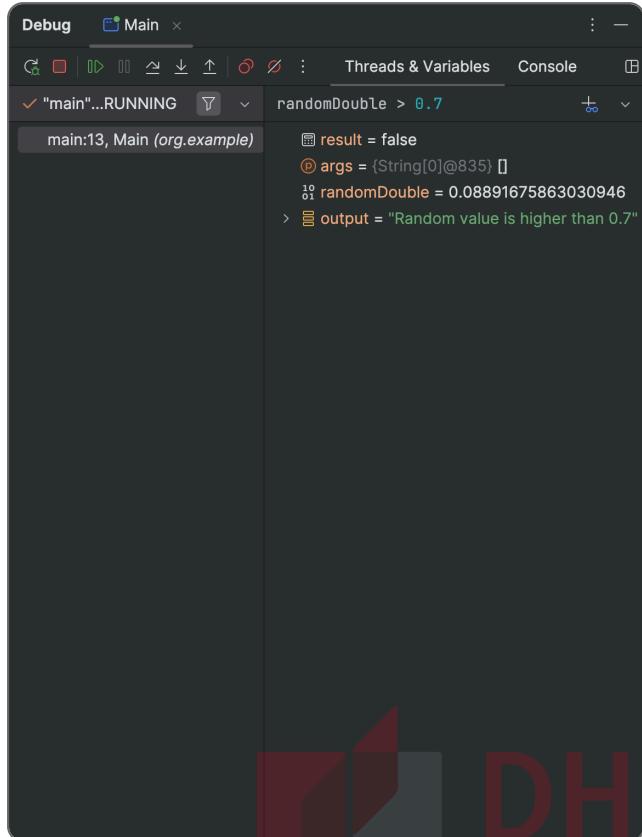
**VS CODE**



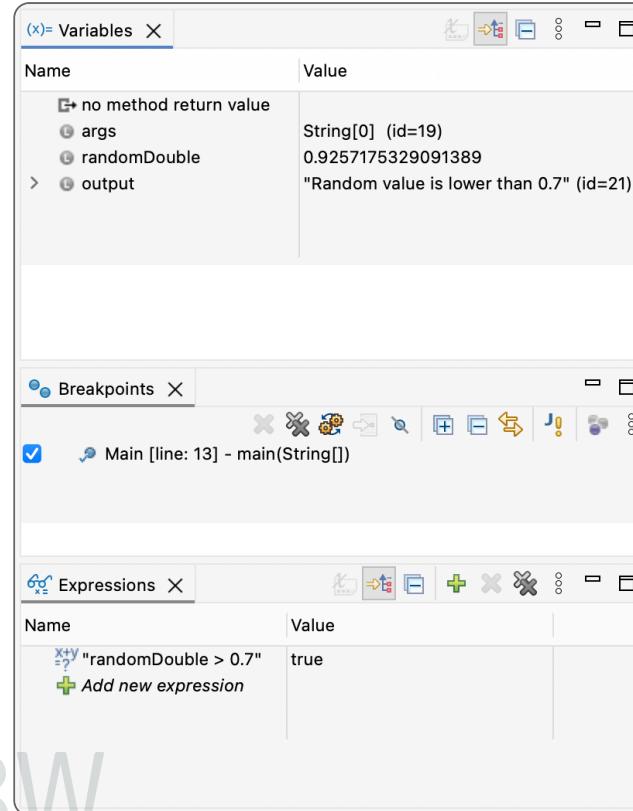
- 👉 **Step Into:** In die Methode hineinspringen
- 👉 **Step Over:** Die Methode ausführen, ohne hineinzuspringen
- 👉 **Step Out:** Aus der aktuellen Methode herausspringen
- 👉 **Resume:** Programm bis zum nächsten Breakpoint fortsetzen
- 👉 **Evaluate Expression:** Beliebigen Ausdruck zur Laufzeit auswerten
- 👉 **Run to Cursor:** Programm bis zur Cursor-Position ausführen

# DEBUGGER IN DER IDE

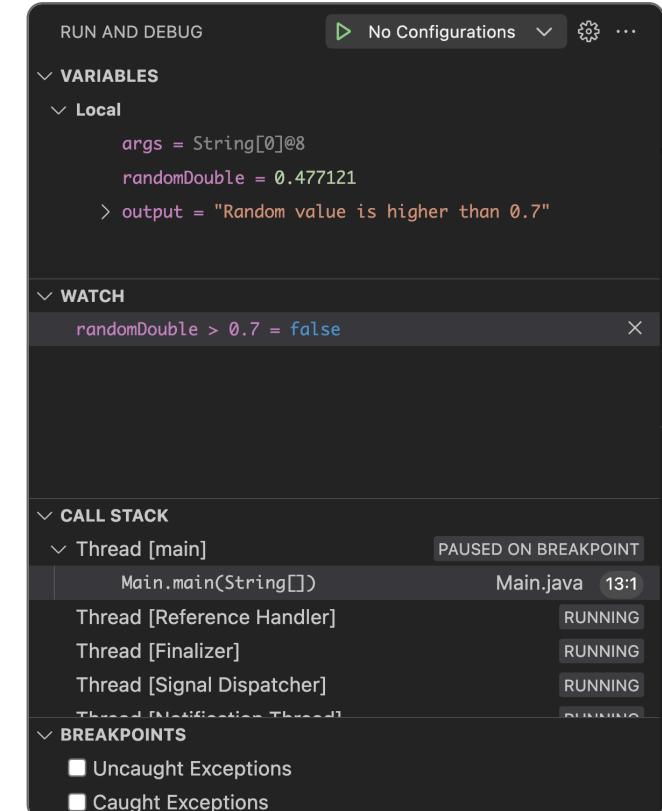
## INTELLIJ IDEA



## ECLIPSE IDE



## VS CODE



# BREAKPOINTS

- 🐢 Anhalten bei Breakpoints nicht effizient bei großen Softwareprojekten
- 👉 Besser: Gezieltes Überwachen/Anhalten des Programms notwendig

Breakpoint-Art	Beschreibung
Standard Breakpoint	Hält die Programmausführung an einer bestimmten Codezeile an.
Conditional Breakpoint	Hält die Ausführung nur an, wenn eine bestimmte Bedingung erfüllt ist.
Logpoint	Schreibt eine Nachricht in die Konsole, ohne die Ausführung zu unterbrechen.
Exception Breakpoint	Hält die Ausführung an, wenn eine bestimmte Exception geworfen wird.



# WEITERE MÖGLICHKEITEN IM DEBUGGER

## ERWEITERTE FEATURES

 **Ändern von Variablenwerten:** Zur Laufzeit Werte modifizieren und Auswirkungen testen

 **Manuelles Ausführen von Methoden:** Beliebige Methoden während des Anhaltens aufrufen

 **Hot Swapping:** Code-Änderungen ohne Neustart anwenden

## REMOTE DEBUGGING

 **Server-Debugging:** Code auf entfernten Servern debuggen

 **Debug-Port:** Verbindung über spezielle Debug-Ports (z.B. Port 5005)

 **Mobile Apps:** Debugging von Apps auf Geräten oder Emulatoren

 **Wichtiger Hinweis:** Remote Debugging in Produktionsumgebungen sollte nur in Ausnahmefällen und mit entsprechenden Sicherheitsvorkehrungen durchgeführt werden.

# XML



# DATEN

- Unstrukturiert (z.B. )
- Semi-Strukturiert (z.B. )
- Strukturiert (z.B. )

## XML

... ist eine Auszeichnungssprache sowie ein Datenaustauschformat und ist

- Maschinenlesbar und interpretierbar
- Semi-strukturiert



## PIZZA ONLINE BESTELLEN

Welche Daten fallen an?

### UNSTRUKTURIERT

Hi, ich hätte gerne *eine extragroße Pizza Funghi mit zusätzlich Artischocken*. Bitte liefern zu *Lieschen Müller* in die *Erzbergerstraße 1337*. Pronto!

### SEMI-STRUKTURIERT

```
1 <pizza>
2   <art>funghi</art>
3   <groesse>extragroß</groesse>
4   <extra_zutaten>
5     <zutat>artischocken</zutat>
6   </extra_zutaten>
7   <lieferort>
8     <name>Lieschen Müller</name>
9     <strasse>Erzbergerstraße
10    1337</strasse>
11   </lieferort>
12   <lieferzeit>sofort</lieferzeit>
13 </pizza>
```

Frage: Warum ist XML "nur" semi-strukturiert?

# XML - ÜBERSICHT

- eXtensible Markup Language
- Auszeichnungssprache
  - d.h. sie ist nicht ausführbar und tut nichts
- "Extensible", also erweiterbar
  - es können eigene Elemente hinzugefügt werden
  - daher nur semi-strukturiert
- Entworfen als Web Standard in 1996 vom World Wide Web Consortium (W3C)

# DESIGN ZIELE

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

Quelle: [W3C Empfehlung](#)

# LITERATUR



- Webseiten des W3-Konsortiums ([w3.org](http://w3.org))
- Tutorials:
  - [W3Schools: Introduction to XML](#)
  - [MDN: XML introduction](#)
- Bücher
  - Verhoegen, Helmut: Einstieg in XML. Bonn: Galileo Press GmbH.
  - Sebestyen, Thomas J.: XML: Einstieg für Anspruchsvolle, Pearson Studium.

# TOOLS

- Editoren
  - Beliebige Code-Editoren: [VS Code](#), [Notepad++](#), ...
  - Spezialisierte Editoren: [Altova XMLSpy](#), [XMLNotePad](#), ...
  - Online-Editoren: [JSONFormatter](#): [XML Editor](#), [xmleditoronline.org](#), ...
- Validierung
  - [W3C Validator](#)
  - [Truugo XML, DTD, XSD Validator](#)

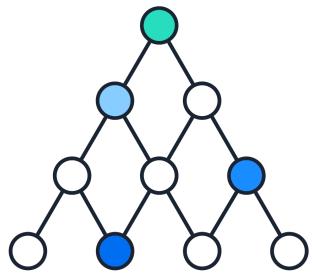
# XML AUFBAU UND SYNTAX



# WAS IST EIN DOKUMENT?



## Struktur



## Inhalt



Text



Tabelle



Bild

Formatierung



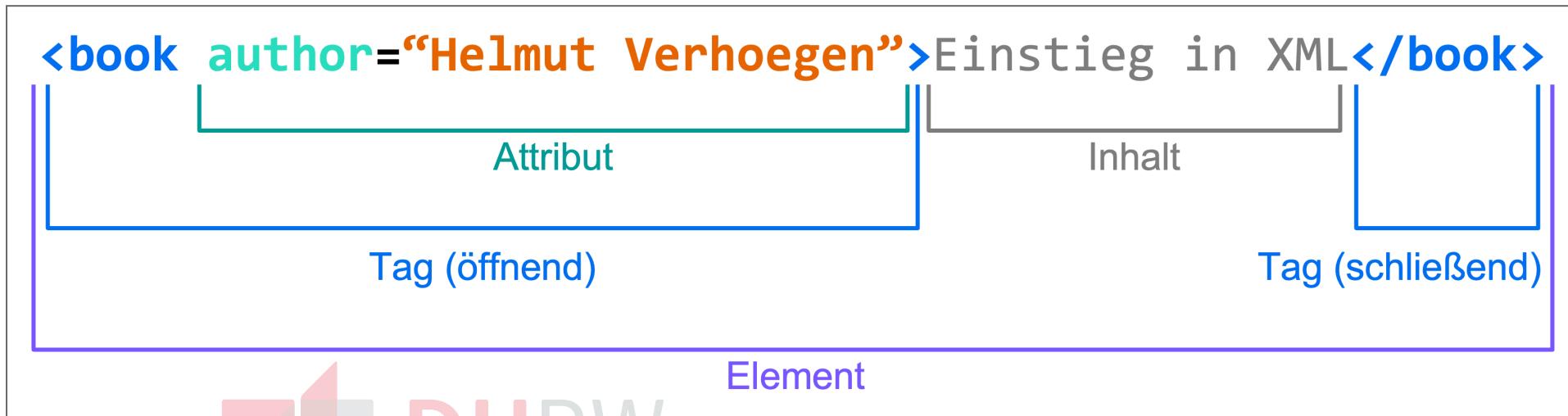
Dokument

Teil von XML

kein Teil von XML

## XML ELEMENTE

- Ein XML-Dokument besteht aus *Elementen*
- Elemente sind definiert durch *Tags*
- Elemente haben i.d.R. einen Inhalt
- Elemente können leer sein
- Elemente können Attribute besitzen, welche Eigenschaften des Elements beschreiben

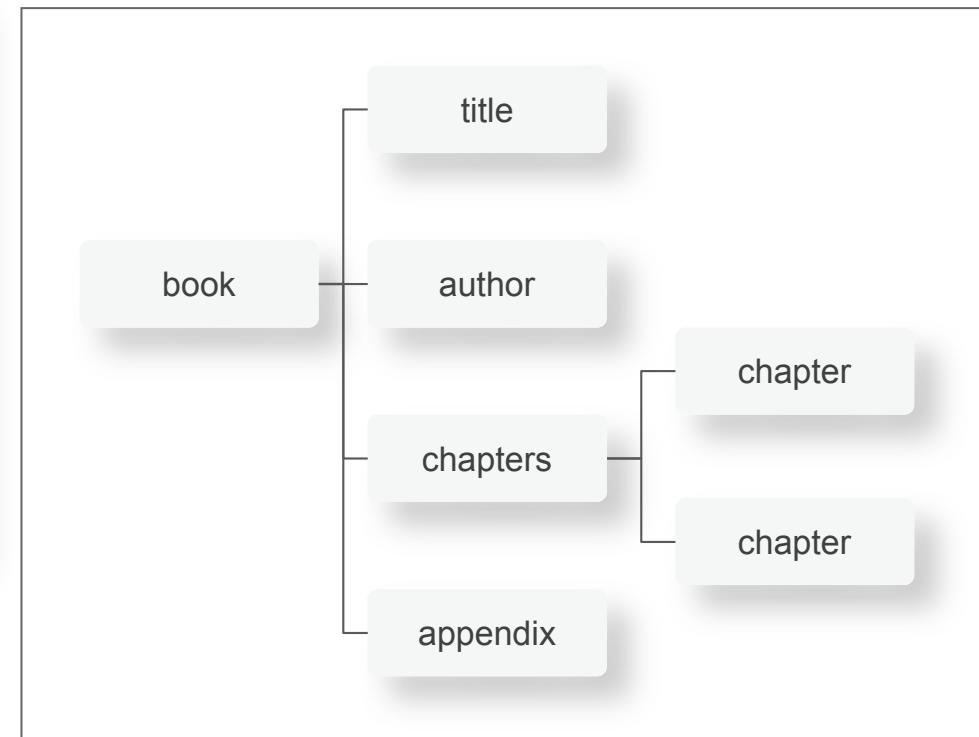


# XML HIERARCHIE

## XML-DOKUMENT

```
1 <book>
2   <author>...</author>
3   <title>...</title>
4   <chapters>
5     <chapter>
6       <title>...</title>
7       <content>...</content>
8     </chapter>
9     <chapter>
10       <title>...</title>
11       <content>...</content>
12     </chapter>
13   </chapters>
14   <appendix>...</appendix>
15 </book>
```

## HIERARCHIE



# STRUKTUR EINES XML-DOKUMENTS

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- XML-Deklaration ↑ -->
3
4  <?xml-stylesheet type="text/css" href="mystyle.css"?>
5  <!-- Verarbeitungsanweisung (optional) ↑ -->
6
7  <!DOCTYPE book SYSTEM "book.dtd">
8  <!-- Dokumenttypdeklaration mit optionalem Verweis auf DTD ↑ -->
9
10 <!-- XML-Daten (getaggter Text) ↴ -->
11 <book title="Einführung in XML" author="Max Mustermann">
12 <...>
13 </book>
```

# ÜBUNG

Was ist das Problem mit folgendem XML-Dokument?



```
1 <Book>
2   <Title>Mathe für Ingenieure 1 & 2 Lernhefte Set</Title>
3   <Author>Daniel Jung</Author>
4   <ISBN>9783947506378</ISBN>
5   <Inhalt>
6     <Kapitel>Mathematischer Werkzeugkoffer</Kapitel>
7     <Kapitel>Warum ist 0 < 1?</Kapitel>
8   </Inhalt>
9 </Book>
```

Parse XML

# ÜBUNG

Wir ersetzen das &-Symbol



```
1 <Book>
2   <Title>Mathe für Ingenieure 1 und 2 Lernhefte Set</Title>
3   <Author>Daniel Jung</Author>
4   <ISBN>9783947506378</ISBN>
5   <Inhalt>
6     <Kapitel>Mathematischer Werkzeugkoffer</Kapitel>
7     <Kapitel>Warum ist 0 < 1?</Kapitel>
8   </Inhalt>
9 </Book>
```

Parse XML

# ÜBUNG - WORKAROUND

Wir ersetzen das unvollständige Tag beginnend mit <

```
● ● ●  
1 <Book>  
2   <Title>Mathe für Ingenieure 1 und 2 Lernhefte Set</Title>  
3   <Author>Daniel Jung</Author>  
4   <ISBN>9783947506378</ISBN>  
5   <Inhalt>  
6     <Kapitel>Mathematischer Werkzeugkoffer</Kapitel>  
7     <Kapitel>Warum ist 0 kleiner als 1?</Kapitel>  
8   </Inhalt>  
9 </Book>
```

Parse XML

Was ist, wenn wir die Zeichen nicht umschreiben können oder wollen?



## XML-INHALT & ESCAPING

- Inhalt sind (fast) alle beliebigen Zeichen aus dem Unicode-Alphabet
- Einige Zeichen sind reserviert und müssen escaped werden:
  - < → &lt;
  - > → &gt;
  - & → &amp;
  - " → &quot;
  - ' → &#39;

# ÜBUNG - LÖSUNG



```
1 <Book>
2   <Title>Mathe für Ingenieure 1 & 2 Lernhefte Set</Title>
3   <Author>Daniel Jung</Author>
4   <ISBN>9783947506378</ISBN>
5   <Inhalt>
6     <Kapitel>Mathematischer Werkzeugkoffer</Kapitel>
7     <Kapitel>Warum ist  $1 \gt; 0$ ?</Kapitel>
8
9   </Inhalt>
10 </Book>
```

Parse XML

## XML-KOMMENTARE

- Dürfen überall stehen, wo Text oder Elemente stehen dürfen
- Werden vom Parser in der Verarbeitung ignoriert
- Beginnen mit `<!--` und enden mit `-->`
- Sind auch über mehrere Zeilen möglich

● ● ●

```
1 <book>
2   <!-- Das ist ein Kommentar innerhalb eines book-Elements -->
3 </book>
4 <!--
5   Das ist ein Kommentar
6   über mehrere Zeilen
7 -->
```

## XML-WHITESPACE

- Whitespace = Leerzeichen, Tabs, Zeilenumbrüche
- Für die Verarbeitung gilt:
  - Außerhalb von Elementen darf beliebig Whitespace verwendet werden  
👉 wird für Verarbeitung ignoriert
  - Innerhalb von Elementen zählt Whitespace als Inhalt und wird beibehalten
- Whitespace kann und soll verwendet werden, um die Lesbarkeit zu erhöhen

## CDATA-ABSCHNITTE

- CDATA steht für "Character Data"
- Erlaubt das Einfügen von Text, der nicht geparsst wird
- Beginnen mit <![CDATA[ und enden mit ]]>
- Wird verwendet, um Zeichen zu verwenden, die sonst escaped werden müssten
- Innerhalb ist alles erlaubt außer ]]>



```
1 <book>
2   <title>Einführung in XML</title>
3   <description>
4     <![CDATA[
5       Dies ist ein Text, der nicht geparsst wird.
6       Reservierte Zeichen wie >, <, &, " und '
7       müssen nicht escaped werden.
8     ]]>
9   </description>
10 </book>
```

## XML-REGELN

- ! Ein XML-Dokument muss *genau ein* Wurzelement haben
- ! XML ist case-sensitive, d.h. Groß- und Kleinschreibung unterscheiden sich
- ! Elemente dürfen nicht überlappen
- ! Tags müssen immer geschlossen werden
- ! Elemente müssen korrekt geschachtelt sein, d.h. sie müssen in der richtigen Reihenfolge geschlossen werden
- ! Attribute müssen in (doppelten) Anführungszeichen stehen

# XML IM ALLTAG

- XML wird in vielen Bereichen eingesetzt, z.B.:
  - Webentwicklung (z.B. XHTML, SVG)
  - Datenbanken (z.B. XML-Datenbanken)
  - Konfigurationsdateien (z.B. Maven, Android)
  - Dokumentenaustausch (z.B. Office Open XML, OpenDocument)
- XML ist ein Standardformat für den Datenaustausch zwischen Systemen:
  - SOAP-API
  - RSS/Atom-Feeds

# BEISPIELE: XML IM ALLTAG

## WEBENTWICKLUNG

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport"
6          content="width=device-width, initial-
7          scale=1.0">
8      <title>Hello World! Site
9      Title</title>
10     </head>
11     <body>
12       <h1>Hello World!</h1>
13     </body>
14   </html>
```

## KONFIGURATIONSDATEI

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project
3    xmlns="http://maven.apache.org/POM/4.
4    0.0"
5    xmlns:xsi="http://www.w3.org/2001/XML
6    Schema-instance"
7    xsi:schemaLocation="http://maven.apac
8    he.org/POM/4.0.0
9    http://maven.apache.org/maven-
10   v4_0_0.xsd">
11   <modelVersion>4.0.0</modelVersion>
12   <groupId>com.sample.project</groupId>
13   <artifactId>core</artifactId>
14   <version>1.0.0-SNAPSHOT</version>
15   <name>Component Name</name>
16   <properties>
17     <java.version>17</java.version>
18   <project.build.sourceEncoding>UTF-
19   8</project.build.sourceEncoding>
20   </properties>
21   <dependencies>
22     <dependency>
23       <groupId>junit</groupId>
```

```
20 <artifactId>junit</artifactId>
21   <version>4.12</version>
22   <scope>test</scope>
23 </dependency>
24 <dependency>
25 <groupId>org.assertj</groupId>
26   <artifactId>assertj-
27   core</artifactId>
28   <version>3.6.2</version>
29   <scope>test</scope>
30 </dependency>
31 </dependencies>
32 <build>
33   <plugins>
34     <plugin>
35       <artifactId>maven-
36       compiler-plugin</artifactId>
37       <version>3.7.0</version>
38       <configuration>
39         <source>${java.version}</source>
40         <target>${java.version}</target>
41       </configuration>
42     </plugin>
43   </plugins>
44 </build>
```

# XML ≠ XML

order\_type\_1.xml

```
● ● ●  
1 <pizza>  
2   <art>funghi</art>  
3   <groesse>extragroß</groesse>  
4   <extra_zutaten>  
5     <zutat>artischocken</zutat>  
6   </extra_zutaten>  
7   <lieferort>  
8     <name>Lieschen Müller</name>  
9     <strasse>Erzbergerstraße  
10    1337</strasse>  
11   </lieferort>  
12   <lieferzeit>sofort</lieferzeit>  
13 </pizza>
```

order\_type\_2.xml

```
● ● ●  
1 <pizza art="funghi" groesse="xxl">  
2   <belaege>  
3     <belag>artischocken</belag>  
4   </belaege>  
5   <anschrift>  
6     <vorname>Lieschen</vorname>  
7     <nachname>Müller</nachname>  
8   <strasse>Erzbergerstraße</strasse>  
9     <nr>1337</nr>  
10   </anschrift>  
11   <liefern_in>-1</liefern_in>  
12 </pizza>
```

Was sind die Unterschiede in den folgenden Pizza-Bestellungen?

# DATA TYPE DEFINITION

- Data Type Definition = DTD
- Beschreibung der Dokumentstruktur für eine Klasse von Dokumenten
- Die DTD legt fest
  - welche Elemente erlaubt sind ↗ ELEMENT
  - welche Inhalte Elemente haben dürfen
  - wie Elemente verschachtelt sein dürfen (welche Beziehungen sie zueinander haben)
  - welche Attribute es geben darf ↗ ATTLIST
  - welche Werte ein Attribut annehmen darf
  - welche (komplexeren) Bausteine es gibt ↗ ENTITY

Empfohlene Literatur: [W3Schools: DTD Tutorial](#)

# DTD-BEISPIEL

## XML-DOKUMENT

```
1 <pizza art="funghi"  
2   groesse="xxl">  
3     <belaege>  
4       <belag>artischocken</belag>  
5     </belaege>  
6     <anschrift>  
7       <vorname>Lieschen</vorname>  
8       <nachname>Müller</nachname>  
9  
10      <strasse>Erzbergerstraße</strasse  
11    >  
12      <nr>1337</nr>  
13  </anschrift>  
14  <liefern_in>60</liefern_in>  
15 </pizza>
```

## DTD-DATEI

```
1  <!DOCTYPE pizza [  
2    <!ELEMENT pizza (belaege, anschrift,  
3      liefern_in)*>  
4    <!ATTLIST pizza  
5      art CDATA #REQUIRED  
6      groesse CDATA #REQUIRED  
7    >  
8    <!ELEMENT belaege (belag)*>  
9    <!ELEMENT belag (#PCDATA)>  
10   <!ELEMENT anschrift (vorname, nachname,  
11     strasse, nr)>  
12   <!ELEMENT vorname (#PCDATA)>  
13   <!ELEMENT nachname (#PCDATA)>  
14   <!ELEMENT strasse (#PCDATA)>  
15   <!ELEMENT nr (#PCDATA)>  
16   <!ELEMENT liefern_in (#PCDATA)>  
17 ]>
```

# DTD - SYNTAX

## ELEMENTE

- Der Elementname
  - ⚠ beginnt mit einem Buchstaben oder Unterstrich
  - ⚠ kann danach zusätzlich Zahlen, Bindestriche und Punkte enthalten
  - ⚠ ist case-sensitive
  - ⚠ darf keine Leerzeichen enthalten
  - ⚠ darf nicht mit der Buchstabenkombination XML beginnen
  - ❗ Doppelpunkte sollten nur mit Namespaces (Namensräumen) verwendet werden

# ÜBUNG

Welche Elementnamen sind gültig?

<Name1> 

<Complete Name> 

<Name.1> 

<\_Name> 

<Name=Max> 

<Name-1> 

< Name > 

<1Name> 

<XMLName> 

<CompleteName> 

# DTD - SYNTAX

## ELEMENTE

Syntax: <!ELEMENT Elementname (Inhalt)\*>

Leeres Element	<!ELEMENT Elementname EMPTY>
Element mit Textinhalt	<!ELEMENT Elementname (#PCDATA)>
Element mit jeglichem Inhalt	<!ELEMENT Elementname ANY>
Element mit Kindelementen	<!ELEMENT Elementname (Kind1, Kind2, Kind3)>
Element mit gemischem Inhalt	<!ELEMENT Elementname (Kind1, Kind2, #PCDATA)>

# DTD - SYNTAX

## KARDINALITÄTEN

1..1 ➔ Genau einmal	<!ELEMENT person (name, age, height)>
0..1 ➔ Optional	<!ELEMENT person (name, age, partner?)>
1..* ➔ Mindestens einmal	<!ELEMENT book (titel, chapter+)>
0..* ➔ Optional, aber beliebig oft	<!ELEMENT pizza (topping, extraTopping*)>

## **DTD - SYNTAX**

### **ATTRIBUTLISTE**

Syntax: <!ATTLIST Elementname Name Typ Wert>



## ATTRIBUTTYPEN

CDATA	Beliebiger Textinhalt
ID	Eindeutige ID innerhalb des Dokuments
IDREF	Referenz auf ein anderes Element mit ID
IDREFS	Eine Liste von IDREF
ENTITY	Name einer vordefinierten, externen Entität
ENTITIES	Eine Liste von ENTITY
ENUMERATION	Eine Liste von erlaubten Werten

## ATTRIBUTWERTE

<Wert>	Standartwert des Attributs
#REQUIRED	Attribut muss angegeben werden
#IMPLIED	Attribut ist optional
#FIXED <wert>	Attribut hat einen festen Wert, der nicht geändert werden kann

# DTD - SYNTAX

## ATTRIBUTLISTE - BEISPIEL



```
1 <!ATTLIST recepie
2   id          ID                      #REQUIRED
3   cookbook    IDREF                  #REQUIRED
4   author      CDATA                 #IMPLIED
5   type        ( cooking | baking ) "cooking"
6   title       CDATA                 #IMPLIED>
```

# PCDATA VS. CDATA

PCDATA ≠ CDATA

- PCDATA
  - 👉 steht für "Parsed Character Data"
  - 👉 Text, der vom Parser interpretiert (geparst) wird
  - 👉 Reservierte Zeichen müssen escaped werden
- CDATA
  - 👉 steht für "Character Data"
  - 👉 Text, der nicht geparsst wird
  - 👉 Reservierte Zeichen müssen nicht escaped werden

## ELEMENTE VS. ATTRIBUTE

Es gibt keine Regel👉 Personal Preference

- 💡 Attribute für Metadaten (also Infos über das Element)
- 💡 Elemente für Inhalt (also eigentliche Daten des Dokuments)
- 💡 Elemente für Listen/Aufzählungen

Generelle Empfehlung: 💡 Elemente für hierarchische Strukturen

# XML UND DTD - BEISPIEL

## DTD-DEFINITION

```
1 <!ELEMENT lunch (drink*, meal+)>
2 <!ELEMENT drink (#PCDATA)>
3 <!ATTLIST drink size
   (small|medium|large) #REQUIRED>
4 <!ELEMENT meal (#PCDATA)>
5 <!ATTLIST meal isVegan (yes|no)
   "no">
```



## XML-DOKUMENT

```
1 <?xml version="1.0" encoding="UTF-
8"?>
2 <!DOCTYPE lunch SYSTEM "lunch.dtd">
3 <lunch>
4   <drink size="medium">Cola</drink>
5   <drink
      size="large">Wasser</drink>
6   <meal isVegan="yes">French
Fries</meal>
7   <meal>Chicken Nuggets</meal>
8 </lunch>
```

# DTD-EINBINDUNG

INTERN

oder EXTERN

```
1 <!DOCTYPE book [  
2 ...  
3 ]>  
4 <!-- ↓ restliches Dokument ↓ -->
```

```
1 <!DOCTYPE book SYSTEM  
"path_to_book_dtd.dtd">  
2 <!-- ↓ restliches Dokument ↓ -->
```

## NACHTEILE VON DTD

- ⚠ Wiederverwendung bzw. Erweiterung ist schwierig
  - ➡ Keine Möglichkeit, nur einen Teil einer DTD zu verwenden
  - ➡ Keine Vererbung
- ⚠ Keine Datentypen/Wertebereiche
- ⚠ Keine komplexeren Kardinalitäten

Alternative: XML Schema Definition (XSD) bietet mehr Möglichkeiten und Flexibilität

# WOHLGEFORMTES VS. VALIDES XML

## WELL FORMED

Ein Dokument ist *wohlgeformt*, wenn...

...es der XML-Syntax genügt

...es genau ein Wurzelement hat

...es korrekte Schachtelung aufweist

## VALID

Ein Dokument ist *valide*, wenn...

...es wohlgeformt ist

...es auf eine existierende DTD verweist

...dessen Inhalt der DTD entspricht

# ÜBUNG: WOHLGEFORMT?

- <user phone=049176129537></user> X
- <user e-mail='sara.softwareengineer@gmail.com'></user> ✓
- <user e-mail="sara.softwareengineer@gmail.com"> </user> ✓
- <user e-mail="garkeineemail" semester='second'></user> ✓
- <dhw location="Karlsruhe" location="Deutschland"></dhw> X
- <Car type="compact" brand="VW">polo</car> X
- <vehicle is-electric="tRuE" brand="tesla">3</vehicle> ✓
- <Book xml-related="yes" title="XML Grundlagen"></Book> X
- <Pizza Menu>🍕</Pizza Menu> X

# XPATH

Eine Query Language (Abfragesprache) zum Durchsuchen von XML Dokumenten

## BEISPIEL

```
1   <order>
2     <pizza type="funghi" size="xxl">
3       <additional_ingredients>
4         <ingredient>artischocken</ingredient>
5           </additional_ingredients>
6         </pizza>
7         <pizza type="quattro formaggi" size="m">
8           <additional_ingredients>
9             <ingredient>jalapenos</ingredient>
10            <ingredient>ananas</ingredient>
11              </additional_ingredients>
12            </pizza>
13        </order>
```

💡 Weitere Infos [hier.](#)



## XPATH QUERY

```
<!-- alle Extrazutaten der Bestellung -->
/	order/pizza//ingredient/text()

<!-- alle extragroßen Pizzen der Bestellung -->
/	order/pizza[@size="xxl"]

<!-- Anzahl der Pizzen in der Bestellung -->
count("//pizza")
```

[XPath Tester WebToolkitOnline](#)

[XPath Tester FreeFormatter](#)

# XPATH - SYNTAX

## SELEKTOREN

```
● ● ●  
<?xml version="1.0"  
encoding="UTF-8"?>  
<bookstore>  
  <book>  
    <title lang="en">Harry  
Potter</title>  
    <price>29.99</price>  
  </book>  
  <book>  
    <title  
lang="en">Learning  
XML</title>  
    <price>39.95</price>  
  </book>  
</bookstore>
```

Selektor	Beispiel	Erklärung
<i>Elementtyp</i>	book	Wählt alle <book> Elemente
/	/<bookstore>/<book>	Auswahl auf nächster Hierarchieeben
//	//title	Auswahl auf beliebiger Hierarchieeben
	/bookstore/book/.	Auswahl aktueller Knoten



..	/bookstore/book/..	Auswahl Elternknoten
@attributname	//title/@lang	Attributauswahl
text()	//title/text()	Textauswahl

# XPATH - SYNTAX

## PRÄDIKATE





```
<?xml version="1.0"
encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry
Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title
lang="en">Learning
XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

Prädikattyp	Beispiel	Erklä
Index	//book[1]	Se El
Attribut	//title[@lang='en']	Se El At
Kindelement	//book[price > 30]	El Ü in
Funktion	//book[last()] //book[contains(title/@lang,"e")]	Se El El Fn

# XPATH - SYNTAX

## RELATIONEN

● ● ●	<?xml version="1.0" encoding="UTF-8"?> <bookstore> <book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book> </bookstore>
-------	--

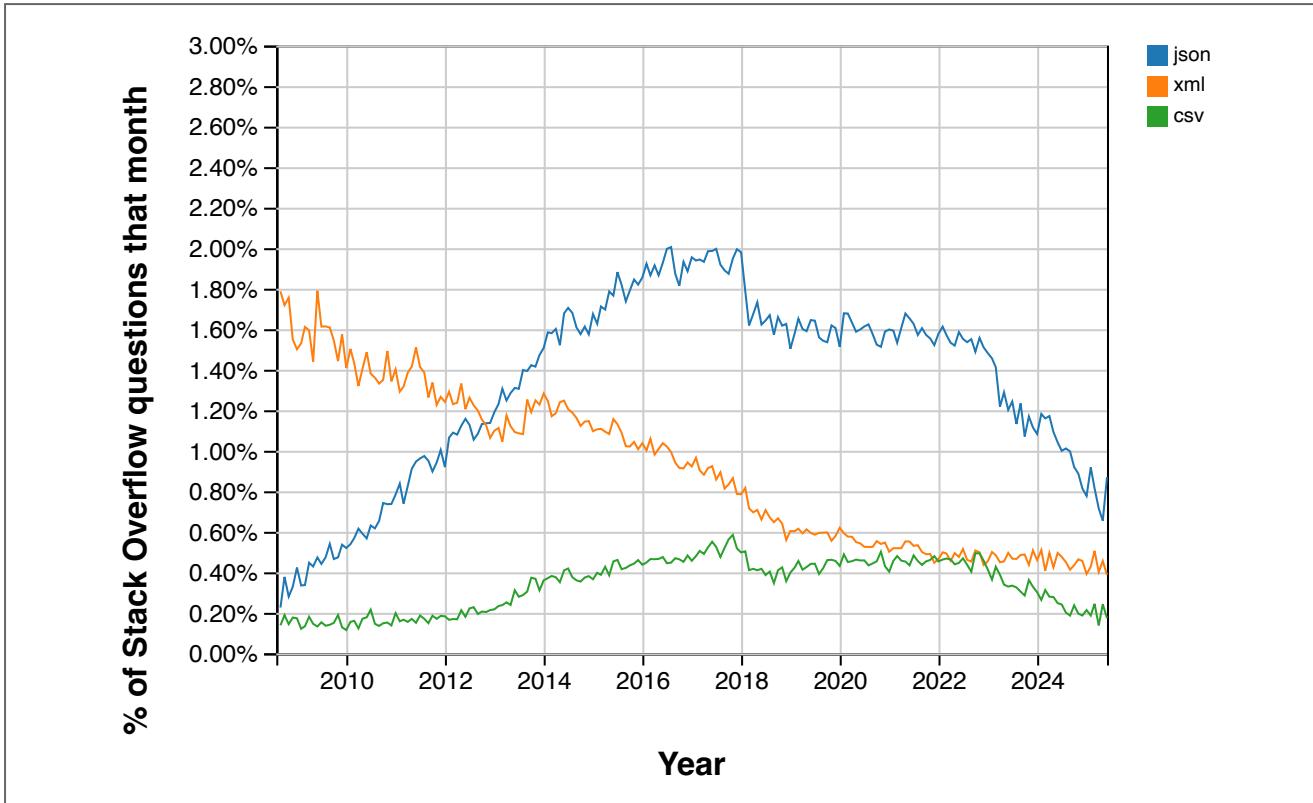
Relation	Beispiel	Erklärung
descendant / ancestor	/bookstore/descendant::price	Rekursiv Kindknoten des Elementes
child / parent	//book[1]/child::price	Alle Kindknoten des Typs
following / preceding	//book[1]/following::price	Alle Nachfolger des Typs
following-sibling /	//book[1]/following-sibling::book	Alle Nachfolger des Typs

preceding-  
sibling

gleichen  
Elternkn



## WEITERE DATENAUSTAUSCHFORMATE



# XML JSON YAML

## XML

```
● ● ●  
<person id="123">  
  <name>Max</name>  
  <age>21</age>  
  <city>Karlsruhe</city>  
  <hobby>Coding</hobby>  
  <hobby>Pizza</hobby>  
  
<isStudent>true</isStudent>  
</person>
```

## JSON

```
● ● ●  
{  
  "id": 123,  
  "name": "Max",  
  "age": 21,  
  "city": "Karlsruhe",  
  "hobbies": ["Coding",  
  "Pizza"],  
  "isStudent": true  
}
```

## YAML

```
● ● ●  
id: 123  
name: Max  
age: 21  
city: Karlsruhe  
hobbies:  
  - Coding  
  - Pizza  
isStudent: true
```

# JSON XML

	XML	JSON
Struktur	Elemente, Attribute	Key-Value Paare
Lesbarkeit/Effizienz	Mehr "verbose"	Relativ einfach & kompakt
Datentypen	keine (alles Text)	Simple Datentypen (string, number, array, bool)
Metadaten	als Attribute neben dem "eigentlichen" Inhalt	als Key-Value neben Daten
Validierung	DTD, XSD	JSON Schema
Kommentare	Ja	Nein
Einsatz	Ideal für komplexere Dokumente	Simpler, "ad-hoc"-Datenaustausch

Copy

# HTML UND CSS



## DARSTELLUNG VON XML-DOKUMENTEN

- XML-Dokumente enthalten keine Informationen über die Darstellung
- Die Darstellung von XML-Dokumenten ist daher abhängig vom verwendeten Browser/Viewer
- Es gibt jedoch Möglichkeiten, XML-Dokumente darzustellen:
  - 👉 Verwendung von Cascading Stylesheets
  - 👉 Verwendung von XSL/T

# CASCADING STYLE SHEETS (CSS)



- CSS ist eine Stylesheet-Sprache, die verwendet wird, um das Layout und die Darstellung von HTML- und XML-Dokumenten zu steuern
- CSS ermöglicht es, Stile für verschiedene Elemente in einem Dokument zu definieren, z.B. Schriftarten, Farben, Abstände usw.

# CSS HISTORIE

- **CSS Level 1.0 Recommendation:**

 17 1. Edition 1996, 11. Edition 2008

 17 Formatierung beschränkt auf Elemente

- **CSS Level 2.0 Recommendation:**

 17 1. Edition 1998, 11. Edition 2008

 17 Unterstützung von Multimedia

 17 Unterstützung mehrerer Ausgabeformate (Braille, Screenreaders)

- **CSS Level 3.0 Recommendation:**

 17 Modularisierung der CSS-Standards in verschiedene Unterstandards

 17 z.B. CSS Print Profile, CSS Color, CSS Speech, ...

# CSS BEISPIEL

## XML-DOKUMENT

```
1 <?xml version="1.0" encoding="UTF-  
2 8"?>  
3 <bestellung nr="111">  
4   <artikel nr="4711">  
5     <bezeichnung>XML für  
6     Einsteiger</bezeichnung>  
7     <preis einheit="euro">47</preis>  
8   </artikel>  
9   <artikel nr="4712">  
10    <bezeichnung>XML für  
11    Fortgeschrittene</bezeichnung>  
12    <preis einheit="euro">52</preis>  
13  </artikel>  
14 </bestellung>
```

## CSS-STYLESHEET

```
1 bestellung {  
2   font-family: Verdana, sans-serif;  
3 }  
4 artikel {  
5   display: list-item;  
6   font-size: 14px;  
7   list-style-type: square;  
8   text-indent: 50px;  
9 }  
10 artikel preis {  
11   font-family: monospace;  
12   color: red;  
13   font-weight: bold;  
14 }
```



# EINBINDUNG VON CSS IN XML

- Prozessanweisung im XML-Dokument:



```
<?xml-stylesheet type="text/css" href="style.css"?>
```

- Parameter der Prozessanweisung:



type: gibt den Typ des Stylesheets an (z.B. text/css, text/xsl)



href: gibt die URI zum Stylesheet an

- CSS-Selektoren:

🎯 Für jedes Element, das speziell formatiert werden soll, muss im CSS ein Selektor angelegt werden

🎯 Beispiel: artikel { display: block; }

# HTML



Duale Hochschule  
Baden-Württemberg

# HYPER TEXT MARKUP LANGUAGE

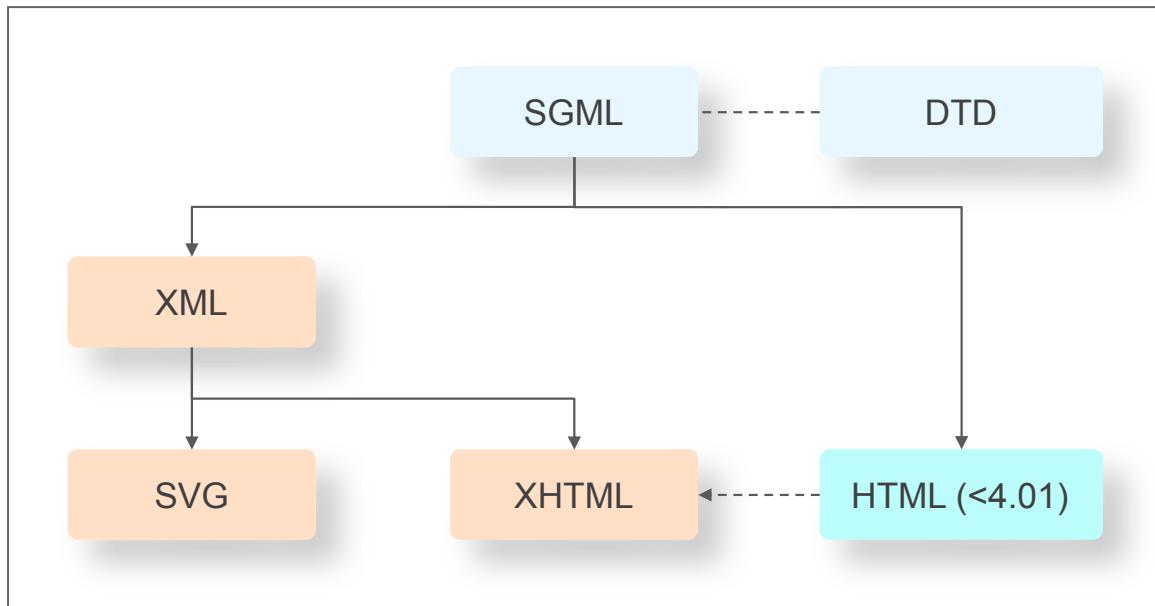


Ist die Grundlage des World Wide Web und definiert die Struktur von Webseiten.

# HTML - WAS IST DAS?

- **HTML ist eine Markup-Sprache** zur Erstellung von Web-Dokumenten
- Basiert auf **XML-Prinzipien**
- **Hypertext-Struktur:**
  - 🔗 Text mit Querverweisen (Links) zu anderen Textpassagen
  - 🔗 Ermöglicht *nicht-lineare Navigation* zwischen Dokumenten
  - 🔗 Direktzugriff auf beliebige Inhalte im Web
- **Technische Grundlage:**
  - ⚙️ HTML-Dokumente sind *wohlgeformte XML-Dokumente*
  - ⚙️ Folgen einer vorgegebenen *DTD* (Document Type Definition)
  - ⚙️ Definieren die *Struktur und Semantik* von Webseiten

# SGML, XML, HTML



- **SGML:** Meta-Sprache zur Definition von Markup-Sprachen
- **HTML:** Spezifische Anwendung von SGML für Webseiten
- **XML:** Vereinfachte Teilmenge von SGML für strukturierte Daten
- **XHTML:** HTML neu formuliert als XML-Anwendung

# HTML

## GRUNDSTRUKTUR

### HTML-DOKUMENT

```
1 <!DOCTYPE html>
2 <html lang="de">
3   <head>
4     <meta charset="UTF-8">
5     <title>Meine Webseite</title>
6   </head>
7   <body>
8     <h1>Willkommen auf meiner Webseite</h1>
9     <p>Dies ist ein einfacher HTML-
Beispieltext.</p>
10    </body>
11 </html>
```



**DHBW**  
Duale Hochschule  
Baden-Württemberg

### IM BROWSER



# HTML ELEMENTE

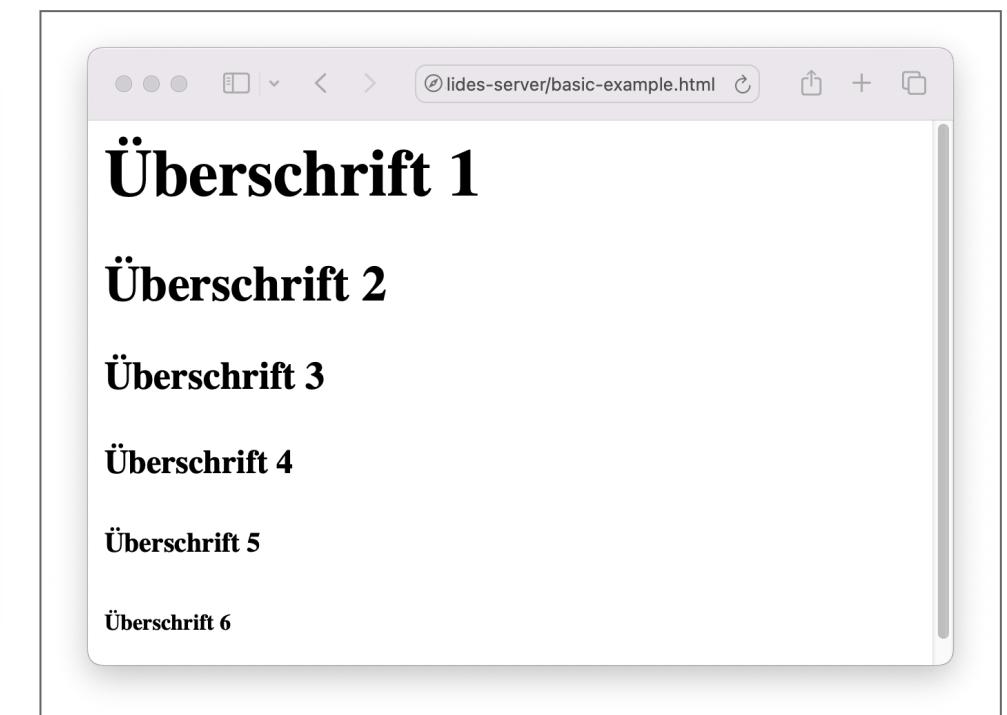
## ÜBERSCHRIFTEN

### HTML-DOKUMENT



```
1 <h1>Überschrift 1</h1>
2 <h2>Überschrift 2</h2>
3 <h3>Überschrift 3</h3>
4 <h4>Überschrift 4</h4>
5 <h5>Überschrift 5</h5>
6 <h6>Überschrift 6</h6>
```

### IM BROWSER



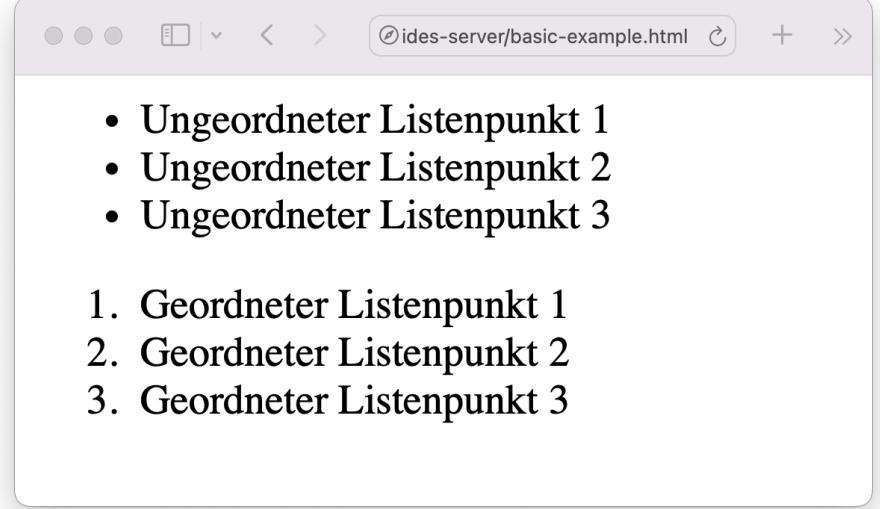
# HTML ELEMENTE

## LISTEN

### HTML-DOKUMENT

```
1 <ul>
2   <li>Ungeordneter Listenpunkt 1</li>
3   <li>Ungeordneter Listenpunkt 2</li>
4   <li>Ungeordneter Listenpunkt 3</li>
5 </ul>
6 <ol>
7   <li>Geordneter Listenpunkt 1</li>
8   <li>Geordneter Listenpunkt 2</li>
9   <li>Geordneter Listenpunkt 3</li>
10 </ol>
```

### IM BROWSER



# HTML ELEMENTE

## TABELLEN

### HTML-DOKUMENT

```
1 <table border="1">
2   <tr>
3     <th>Name</th>
4     <th>Alter</th>
5   </tr>
6   <tr>
7     <td>Anna</td>
8     <td>25</td>
9   </tr>
10  <tr>
11    <td>Max</td>
12    <td>30</td>
13  </tr>
14 </table>
```

### IM BROWSER

Name	Alter
Anna	25
Max	30

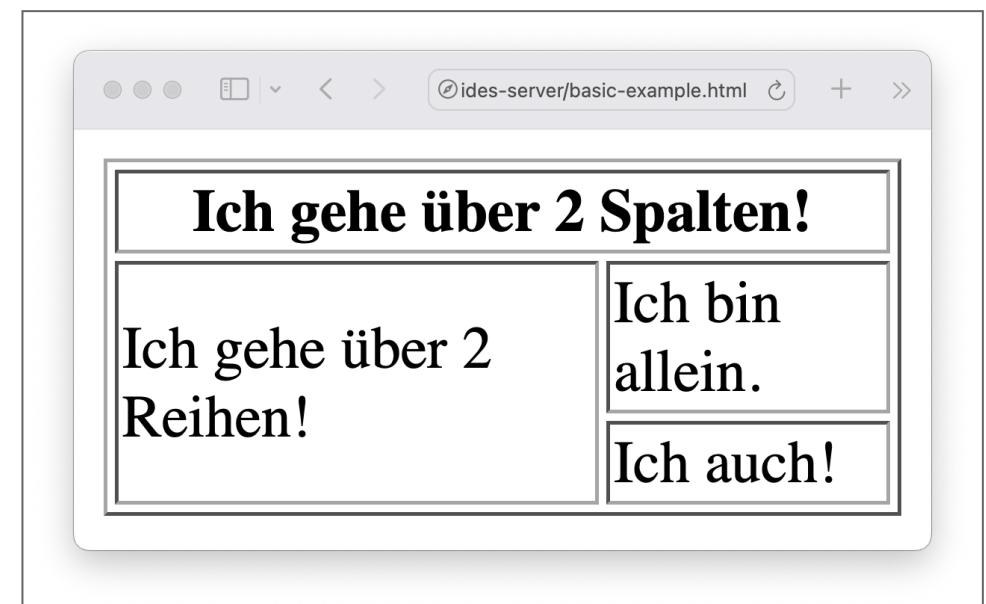
# HTML ELEMENTE

## TABELLEN - ERWEITERTE FUNKTIONEN

### HTML-DOKUMENT

```
1 <table border="1">
2   <tr>
3     <th colspan="2">Ich gehe über 2
4       Spalten!</th>
5     </tr>
6     <tr>
7       <td rowspan="2">Ich gehe über 2
8         Reihen!</td>
9         <td>Ich bin allein.</td>
10        </tr>
11        <tr>
12          <td>Ich auch!</td>
13        </tr>
14      </table>
```

### IM BROWSER



👉 colspan Zelle erstreckt sich über die angegebene Anzahl von Spalten

👉 rowspan Zelle erstreckt sich über die angegebene Anzahl von Zeilen

Hilfreich: [Tables Generator](#) - Online-Tool zum Erstellen von HTML-Tabellen

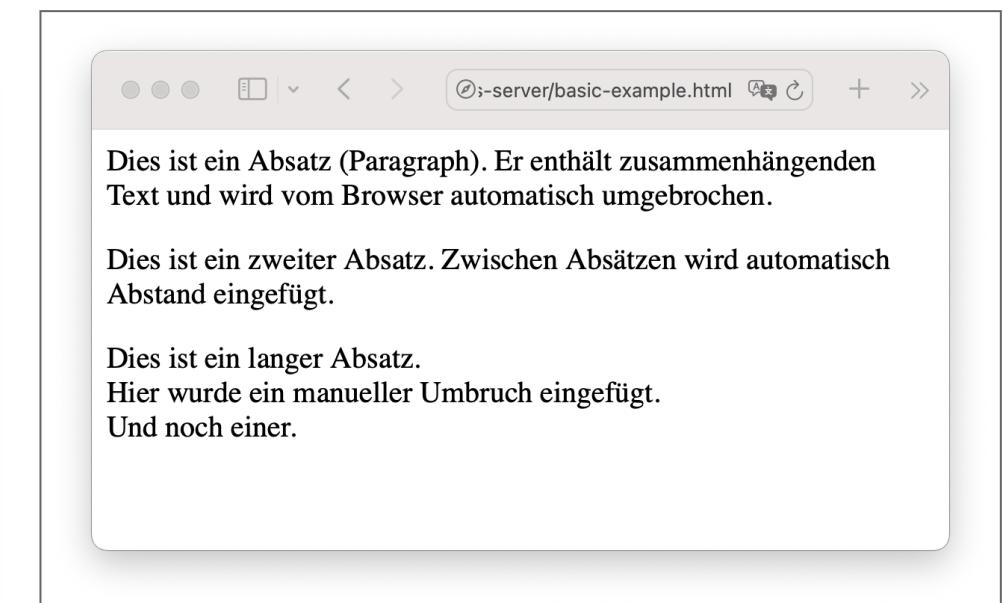
# HTML ELEMENTE

## ABSÄTZE UND UMBRÜCHE

### HTML-DOKUMENT

```
1  <p>  
2    Dies ist ein Absatz (Paragraph).  
3      Er enthält zusammenhängenden Text  
4      und wird vom Browser  
5        automatisch umgebrochen.  
6  </p>  
7  <p>  
8    Dies ist ein zweiter Absatz.  
9      Zwischen Absätzen wird automatisch  
10     Abstand eingefügt.  
11 </p>  
12 <p>  
13   Dies ist ein langer Absatz.<br>  
14   Hier wurde ein manueller Umbruch  
15   eingefügt.<br>  
16   Und noch einer.  
17 </p>
```

### IM BROWSER



- 👉 <p> - Paragraph: Definiert einen Textabsatz mit automatischem Abstand
- 👉 <br> - Break: Erzwingt einen Zeilenumbruch (selbstschließendes Element)

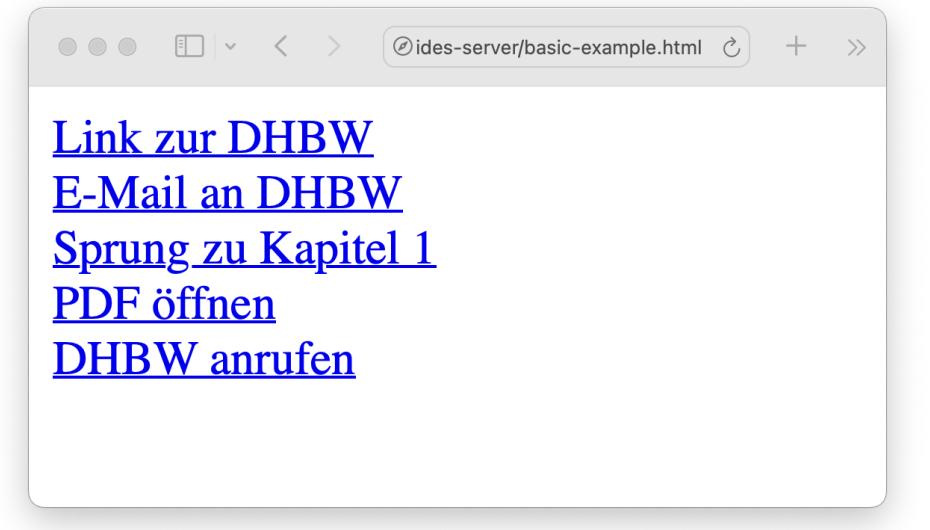
# HTML ELEMENTE

## VERWEISE (HYPERLINKS)

### HTML-DOKUMENT

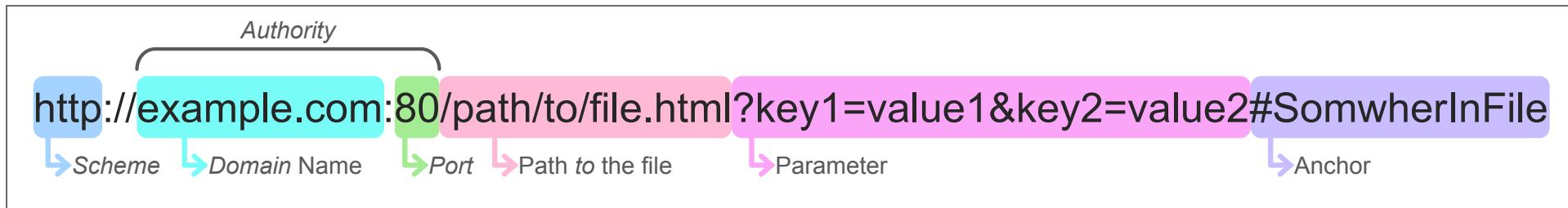
```
1 <a href="https://www.dhbw.de">Link zur  
Hochschule</a>  
2 <br>  
3 <a href="mailto:info@dhbw.de">E-Mail  
an die Hochschule</a>  
4 <br>  
5 <a href="#kapitel1">Sprung zu Kapitel  
1</a>  
6 <br>  
7 <a href="dokument.pdf"  
target="_blank">PDF öffnen</a>  
8 <br>  
9 <a href="tel:+497119212050">Hochschule  
anrufen</a>
```

### IM BROWSER



- 👉 target="\_blank" - Öffnet Link in neuem Fenster/Tab
- 👉 href="#id" - Sprung zu Element mit entsprechender ID auf der selben Seite

# EINSCHUB: URLs



🌐 **URL = Uniform Resource Locator**

🌐 Eindeutige Adresse einer Ressource, die der Browser abrufen kann

🌐 Beispiel: <https://www.karlsruhe.dhbw.de/startseite.html>

👉 Teile können auch ausgelassen werden (z.B. nur Domain)

👉 URLs können auch **relativ** sein (nur Pfad und Parameter)

📁 ./ - der folgende Teilpfad befindet sich im **aktuellen Verzeichnis**

📁 ../ - der folgende Teilpfad befindet sich im **Elternverzeichnis** (eine Ebene höher)

## RELATIVE URLs - BEISPIELE

🏠 Ausgangspunkt: <http://example.org/folder/file.html>

🎯 Ziel: <http://example.org/folder/example.html>

Typ	Beschreibung	Beispiel
Pfad- relativ	Ohne Host und führendes "/", bildet URL immer ausgehend vom aktuellen Verzeichnis	<code>example.html</code> <code>./example.html</code>
Host- relativ	Ohne Host aber mit führendem "/", bildet URL immer ausgehend vom aktuellen Host	<code>/folder/example.html</code>

💡 Pfad-relativ: Bezieht sich auf das aktuelle Verzeichnis der Seite

💡 Host-relativ: Startet immer von der Wurzel der Domain

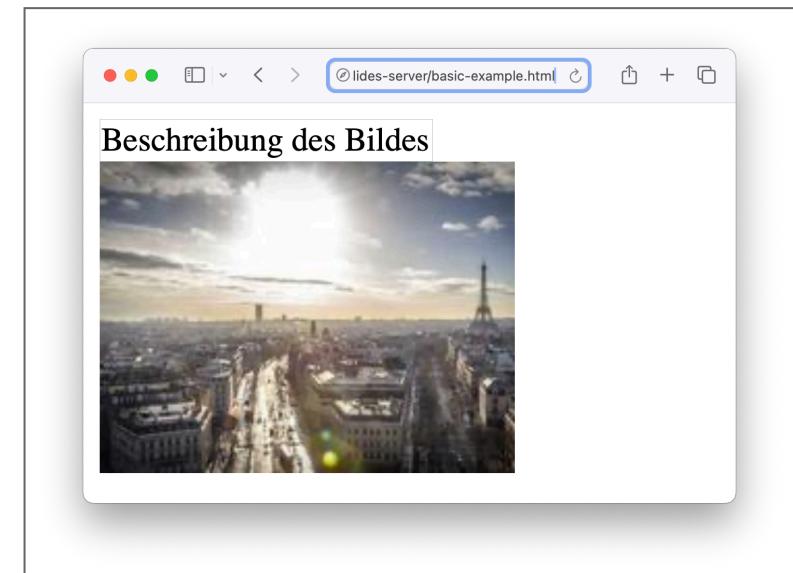
# HTML ELEMENTE

## BILDER

### HTML-DOKUMENT

```
1   
2 
```

### IM BROWSER



- 👉 src - Pfad oder URL zum Bild (erforderlich)
- 👉 alt - Alternative Beschreibung für Screenreader und bei Ladefehlern (erforderlich)
- 👉 width/height - Größe in Pixeln oder über CSS style-Attribut

# BILDFORMATE IM WEB

Format	Eigenschaften	Verwendung
JPEG (.jpg)	Verlustbehaftet, kleine Dateien	Fotos, komplexe Bilder
PNG (.png)	Verlustfrei, Transparenz, größere Dateien	Logos, Screenshots, Grafiken
GIF (.gif)	256 Farben, Animationen, verlustfrei	Einfache Animationen, Icons
SVG (.svg)	Vektorformat, skalierbar, kleine Dateien	Icons, Logos, einfache Grafiken
WebP (.webp)	Modern, sehr kompakt, Transparenz	Alle Arten (mit Fallback)

- 💡 **Performance:** Richtige Formatwahl reduziert Ladezeiten erheblich
- 💡 **Qualität vs. Dateigröße:** JPEG für Fotos, PNG für Grafiken mit Text
- 💡 **Zukunftssicher:** WebP mit JPEG/PNG-Fallback für beste Unterstützung

# HTML ELEMENTE

## META-DATEN

```
1 <head>
2   <title>Wirtschaftsinformatik</title>
3   <meta name="author" content="Katja Wengler">
4   <meta name="generator" content="editpad">
5   <meta name="Keywords" content="Wirtschaftsinformatik,
6       Studium, homepage, Webseite, Informationen">
7   <meta name="description" content="Homepage des FB
8       Wirtschaftsinformatik der DHBW Karlsruhe">
9 </head>
10 <body>
11   Inhalt der Webseite...
12 </body>
```

- 👉 <title> - Titel der Webseite (erscheint im Browser-Tab und bei Suchmaschinen)
- 👉 <meta name="description"> - Kurzbeschreibung für Suchmaschinen
- 👉 <meta name="keywords"> - Schlüsselwörter (heute weniger relevant für SEO)
- 👉 <meta name="author"> - Autor der Webseite

# HTML ELEMENTE

## FORMULARE - GRUNDLAGEN

### HTML-DOKUMENT

```
1 <form action="/submit" method="post">
2   <label for="name">Name:</label>
3   <input type="text" id="name"
4     name="name">
5   <br>
6   <label for="email">E-Mail:</label>
7   <input type="email" id="email"
8     name="email">
9   <br>
10  <input type="submit" value="Absenden">
11 </form>
```



**DHBW**  
Duale Hochschule  
Baden-Württemberg

### IM BROWSER

A screenshot of a web browser window displaying a simple form. The address bar shows the URL "lides-server/basic-example.html". The form contains two text input fields labeled "Name:" and "E-Mail:", and a single button labeled "Absenden".

- 👉 <form> - Container für Formularelement mit action (Ziel-URL) und method (GET/POST)
- 👉 <input> - Eingabefeld mit verschiedenen type-Attributen
- 👉 <label> - Beschriftung für Eingabefelder (mit for-Attribut verknüpft)



# HTML ELEMENTE

## FORMULARE - EINGABEFELDER

Element	Beschreibung	Beispiel
<code>&lt;input type="text"&gt;</code>	Einzeiliges Texteingabefeld	<input type="text"/>
<code>&lt;input type="email"&gt;</code>	Einzeiliges E-Mail-Eingabefeld (validiert E-Mail-Format)	<input type="email"/>
<code>&lt;textarea rows="3" cols="30"&gt;&lt;/textarea&gt;</code>	Mehrzeiliges Texteingabefeld für längere Texte	<input type="text"/> //
<code>&lt;input type="password"&gt;</code>	Passwort-Eingabefeld (Text wird verborgen)	<input type="password"/>
<code>&lt;select name="auswahl"&gt;     &lt;option value="1"&gt;Option 1&lt;/option&gt;     &lt;option value="2"&gt;Option 2&lt;/option&gt;     &lt;option value="3"&gt;Option 3&lt;/option&gt;</code>	Dropdown-Auswahlfeld mit mehreren Optionen	Option 1 ▾

```
3</option>  
  </select>
```

---

```
<input type="checkbox"  
      value="agree">
```

Kontrollkästchen für Ja/Nein-Auswahl

Agree

```
<input type="button"  
      action="test.php">
```

Einfacher Button mit Funktion

Klick mich

```
<input type="submit"  
      value="Absenden">
```

Submit-Button zum Absenden des  
Formulars

Absenden

# HTTP



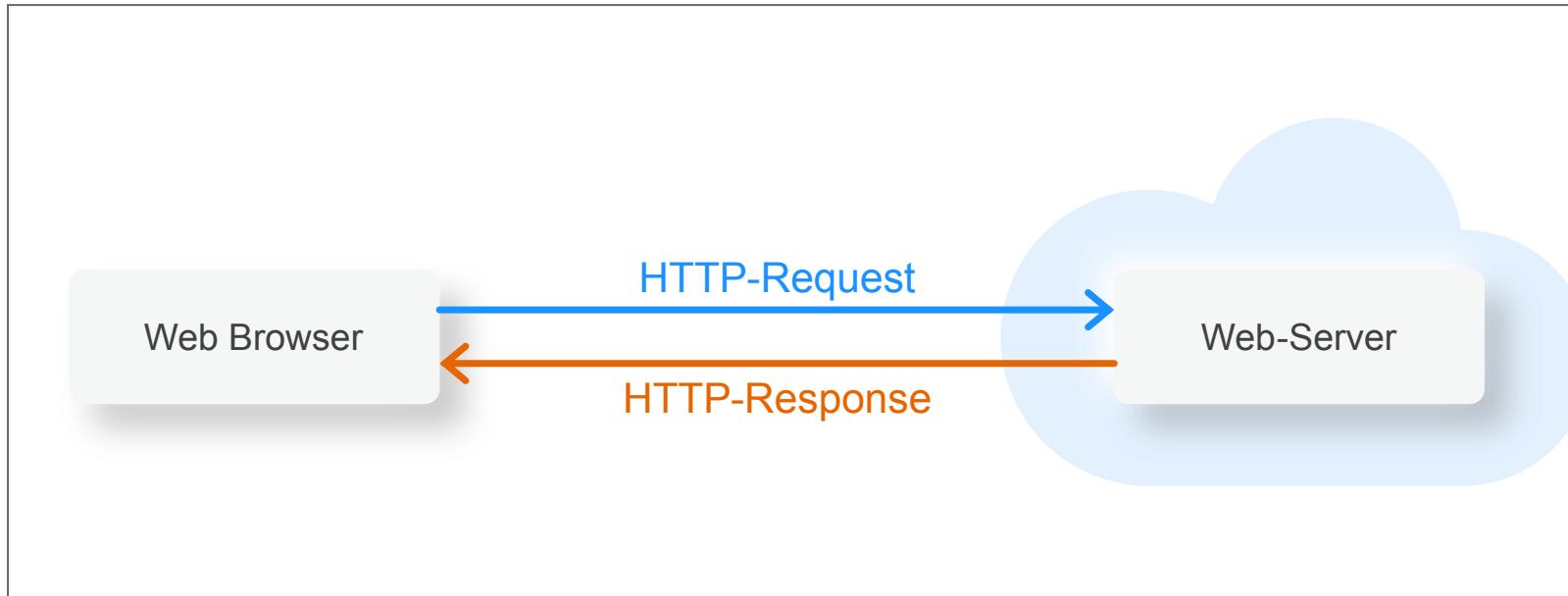
# HTTP

## HYPertext Transfer Protocol

- 👉 Anwendungsschichtprotokoll für die Übertragung von Webinhalten
- 👉 **Nicht-persistente Verbindungen:** Eine TCP-Verbindung pro HTTP-Request/Response
- 👉 **Persistente Verbindungen:** Mehrere HTTP-Requests über eine TCP-Verbindung
- 👉 **HTTP-Nachrichten:** Reine Textnachrichten in strukturiertem Format
- 👉 **Nachrichtenaufbau:**
  - ◆ **Request-Zeile:** Anfrage-Methode, URI, HTTP-Version
  - ◆ **Header-Felder:** Meta-Informationen (optional)
  - ◆ **Leerzeile:** Markiert Ende der Header-Felder
  - ◆ **Message Body:** Eigentlicher Inhalt (optional)

# HTTP

## HTTP REQUEST-RESPONSE MODELL



- 🌐 Client: Sendet HTTP-Requests an den Server
- 🌐 Server: Empfängt Requests, verarbeitet sie und sendet HTTP-Responses zurück
  - 👉 Synchroner Ablauf
- 🌐 HTTP-Methoden: GET, POST, PUT, DELETE, etc. - definieren die Art der Anfrage
- 🌐 HTTP-Statuscodes: Geben den Status der Anfrage an (z.B. 200 OK, 404 Not Found)

# HTTP

## HTTP HEADER-FELDER

**Header-Felder:** Metainformationen für die Bearbeitung der HTTP-Nachricht

Header-Typ	Beschreibung	Beispiele
<b>General-Header</b>	Allgemeine Steuerinformationen für jede HTTP-Nachricht	Date , Connection , Cache-Control
<b>Entity-Header</b>	Zusatzinformationen über den Inhalt des Message Bodys	Content-Type , Content-Length , Content-Encoding
<b>Request-Header</b>	Steuerinformationen nur in der Anfrage	User-Agent , Accept , Host
<b>Response-Header</b>	Steuerinformationen nur in der Antwort	Server , WWW-Authenticate , Set-Cookie

# HTTP

## HTTP REQUEST



```
1 GET /some-path/index.html HTTP/1.1
2 User-Agent: Mozilla/4.0
3 Accept: text/html, image/gif,
image/jpeg
4 Accept-Language: fr, de, en
```

## HTTP RESPONSE



```
1 HTTP/1.1 200 OK
2 Date: Wed, 27 Oct 2021 10.12.13 GMT
3 Server: Apache/1.3.0 (Unix)
4 Last-Modified: Wed, 27 Oct 2021
5 Content-Length: 12345
6 Content-Type: text/html
7
8 <html>
9 ...
10 </html>
```

# HTTP

## WEITERFÜHRENDES

👉 Zum Nachlesen

📚 [HTTP Status Codes](#)

📚 [HTTP Methods](#)

👉 Werkzeuge

🔧 [HTTP Request/Response Tester](#)

🔧 API Testing

- [Postman](#)

- [Insomnia](#)

- [Bruno](#)

🔧 [Browser DevTools \(F12\)](#)



**DHBW**

Duale Hochschule  
Baden-Württemberg

# JAVASCRIPT

- Programmier- / Scriptsprache für dynamische Inhalte im Web (mehr in Vorlesung "Web Programmierung")

● ● ●

```
1 <html>
2 <head>
3   <script type="text/javascript">
4     function quadrat() {
5       let result = document.Formular.Eingabe.value *
6         document.Formular.Eingabe.value;
7       alert("Das Quadrat von " + document.Formular.Eingabe.value + " = " +
8             result)
9     }
10    </script>
11  </head>
12 <body>
13   <form name="Formular" action="">
14     <input type="text" name="Eingabe" size="3">
15     <input type="button" value="Quadrat errechnen" onClick="quadrat()">
16   </form>
17 </body>
```



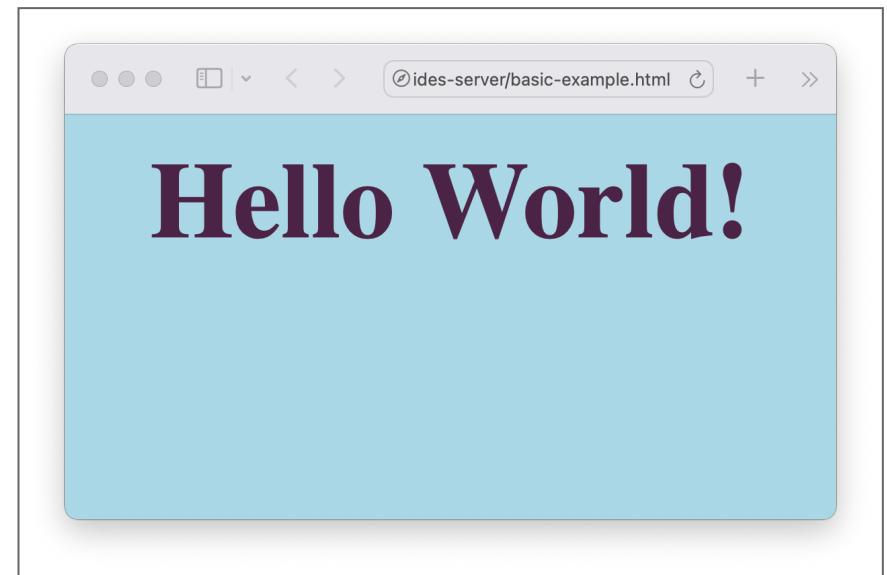
# CSS & HTML

Inline CSS eingebettet direkt in HTML-Dokumenten oder aus externer CSS-Datei geladen.

## HTML-DOKUMENT

```
1 <html>
2 <head>
3   <style>
4     body { background-color: lightblue; }
5     h1 { color: #4d274a; text-align:
center; }
6   </style>
7   <!-- ⏪ oder ⏴ -->
8   <link rel="stylesheet" type="text/css"
href="style.css">
9 </head>
10 <body>
11   <h1>Hello World!</h1>
12 </body>
13 </html>
```

## IM BROWSER



# FARBEN IN CSS

🌈 Farben in CSS können auf verschiedene Arten angegeben werden:

Format	Beispiel	Beschreibung
<b>Farbnamen</b>	red, blue, green	Vordefinierte Farbnamen (z.B. red für Rot)
<b>Hexadezimal</b>	#RRGGBB	RGB-Farben in Hexadezimalform (z.B. #ff0000 für Rot)
<b>RGB</b>	rgb(255, 0, 0)	RGB-Werte im Dezimalformat (z.B. Rot)
<b>RGBA</b>	rgba(255, 0, 0, 0.5)	RGB mit Alpha-Kanal für Transparenz (0-1)
<b>HSL</b>	hsl(0, 100%, 50%)	Farbton, Sättigung und Helligkeit (z.B. Rot)
<b>HSLA</b>	hsla(0, 100%, 50%, 0.5)	HSL mit Alpha-Kanal für Transparenz

# EINHEITEN IN CSS



## CSS unterstützt verschiedene Einheiten für Längenangaben:

Einheit	Beschreibung	Beispiel
px	Pixel - absolute Einheit	width: 100px;
em	Relative Einheit basierend auf der Schriftgröße des Elterelements	font-size: 2em; (doppelte Schriftgröße)
rem	Relative Einheit basierend auf der Schriftgröße des Root-Elements	font-size: 1.5rem; (1.5 mal die Root-Schriftgröße)
%	Prozentuale Einheit basierend auf dem Elterelement	width: 50%; (50% der Breite des Elterelements)
vh	Viewport-Höhe - $1vh = 1\%$ der Höhe des Ansichtsfensters	height: 100vh; (volle Höhe des Viewports)
vw	Viewport-Breite - $1vw = 1\%$ der Breite des Ansichtsfensters	width: 100vw; (volle Breite des Viewports)

# CSS FUNKTIONSWEISSE

```
1 selektor {  
2   eigenschaft: wert;  
3   ...  
4 }
```

```
1 h2 {  
2   text-align: center;  
3   ...  
4 }
```

- **CSS-Selektoren:**

- Für jedes Element, das speziell formatiert werden soll, muss im CSS ein Selektor angelegt werden
- Beispiel: artikel {...} erstellt eine Regel für alle artikel-Elemente

- **CSS-Deklaration:**

- Jede CSS-Deklaration besteht aus einer *CSS-Eigenschaft* und einem Doppelpunkt (:) gefolgt vom *Wert* der Eigenschaft
- Mehrere Deklarationen werden durch Semikolons getrennt und stehen in einem Deklarationsblock
- Ein Deklarationsblock ist von geschwungenen Klammern ( { und } ) eingeschlossen

# CSS SELEKTOREN

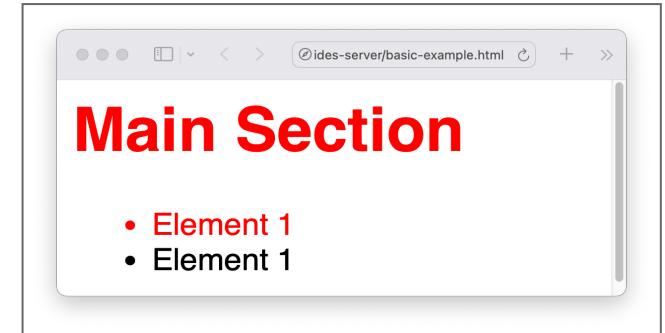
CSS-Selektoren können in 5 Kategorien eingeteilt werden:

- ❑ **Einfache Selektoren:**  
wählen Elemente basierend auf Namen, ID oder Klasse aus
- ❑ **Kombinator-Selektoren:**  
wählen Elemente basierend auf einer spezifischen Beziehung zwischen ihnen aus
- ❑ **Pseudoklassen-Selektoren:**  
wählen Elemente basierend auf einem bestimmten Zustand aus
- ❑ **Pseudoelement-Selektoren:**  
wählen und formatieren einen Teil eines Elements aus
- ❑ **Attribut-Selektoren:**  
wählen Elemente basierend auf einem Attribut oder Attributwert aus

Mehr Infos zu den Selektoren: [W3Schools CSS Selectors](#)

# CSS SELEKTOREN

```
1 <h1 class="red" id="main-sec">Main Section</h1>
2 <ul>
3   <li class="red">Element 1</li>
4   <li>Element 2</li>
5 </ul>
```



```
1 h1 {  
2   font-size: 24pt;  
3 }
```

### Element Selektor

h1 wählt alle h1 Elemente

```
1 .red {  
2   color: red;  
3 }
```

### Klassen Selektor

.red wählt alle Elemente mit der Klasse class="red"

```
1 #main-sec {  
2   text-decoration:  
3     underline;  
4 }
```

### Id Selektor

#main-sec wählt genau das Element mit der id id="main-sec"

```
1 ul, li, .red {  
2   display: inline-  
3     block;  
4 }
```

### Mehrere Selektoren

ul, li, .red wählt alle ul, li und Elemente mit der Klasse .red



**DHBW**  
Duale Hochschule  
Baden-Württemberg

# CSS KOMBINATOR-SELEKTOREN

```
1 <div class="container">
2   <h1>Überschrift</h1>
3   <p>Direktes Kind</p>
4   <div>
5     <p>Verschachteltes Kind</p>
6     <span>Geschwister</span>
7   </div>
8   <p>Nachfolgendes Geschwister</p>
9 </div>
```



```
1 div p {
2   color: blue;
3 }
```

**Nachfahren-Selektor (Leerzeichen)**

div p wählt alle p-Elemente innerhalb von div-Elementen

```
1 .container > p {
2   text-decoration:
3   underline;
```

**Kind-Selektor (>)**

.container > p wählt nur direkte p-Kinder von .container

```
1 h1 + p {
2   background-color:
3   yellow;
```

**Angrenzender Geschwister-Selektor (+)**

h1 + p wählt das erste p-Element direkt nach einem h1

```
1 h1 ~ p {  
2   font-style: italic;  
3 }
```

### Allgemeiner Geschwister-Selektor (~)

h1 ~ p wählt alle p -Elemente, die Geschwister nach einem h1 sind



# HTML5

🚀 Weiterentwicklung von HTML 4.01 und XHTML 1.0

🚀 Bessere Strukturierungsfähigkeit durch semantische Elemente

## NEUE STRUKTURELLE ELEMENTE

- 📄 <section>: Thematische Bereiche
- 📄 <article>: Eigenständige Inhalte
- 📄 <header>: Kopfbereich
- 📄 <footer>: Fußbereich
- 📄 <nav>: Navigation
- 📄 <main>: Hauptinhalt
- 📄 <aside>: Seitenleiste

## MULTIMEDIA & INTERAKTIVITÄT

- 🎯 <audio> / <video> - Native Medienwiedergabe
- 🎯 <canvas> - Zeichnen im Browser
- 🎯 Erweiterte Formular-Inputs:
  - email, url, tel, date, color
  - number, range, search
  - Automatische Validierung
- 🎯 Drag & Drop API
- 🎯 Offline-Funktionalität

## BROWSER DEV-TOOLS

- ❖ Integrierte Werkzeuge in modernen Browsern zur Webentwicklung und -debugging
- ❖ Ermöglichen Inspektion und Bearbeitung von HTML, CSS und JavaScript in Echtzeit
- ❖ Helfen bei der Performance-Analyse, Fehlerbehebung und Optimierung von Webseiten

# RESPONSIVE DESIGN

- 📱 Webdesign-Ansatz für optimale Darstellung auf verschiedenen Geräten und Bildschirmgrößen
- 🌐 Ziel: Einheitliche Benutzererfahrung unabhängig vom Gerät (Desktop, Tablet, Smartphone)
- ⚙️ Verwendet flexible Layouts, Bilder und CSS Media Queries

## TECHNIKEN

- 🔧 Flüssige Layouts mit relativen Einheiten (% , em , rem)
- 🔧 CSS Media Queries für gerätespezifische Stile
- 🔧 Mobile-First Design Ansatz



- 🌐 [Artikel Responsive Design](#)

## VORTEILE

- ✓ Verbesserte Benutzererfahrung auf allen Geräten
- ✓ Höhere Reichweite und Zugänglichkeit
- ✓ Kosteneffizienter als separate mobile Websites
- ✓ Zukunftssicher durch Anpassungsfähigkeit an neue Geräte

# CSS FRAMEWORKS

- ⚙️ Vorgefertigte CSS-Bibliotheken zur schnellen und konsistenten Gestaltung von Webseiten
- 📦 Bieten vordefinierte Klassen, Komponenten und Layout-Systeme
- ⌚ Sparen Entwicklungszeit und verbessern die Wartbarkeit

## BEKANNTES CSS FRAMEWORKS

- 🌐 [Bootstrap](#) - Am weitesten verbreitet, umfangreiche Komponentenbibliothek
- 🌐 [Foundation](#) - Flexibles Grid-System, mobil-first Ansatz
- 🌐 [Bulma](#) - Modernes, flexbox-basiertes Framework
- 🌐 [Tailwind CSS](#) - Utility-first Ansatz für maßgeschneiderte Designs
- 🌐 [Materialize](#) - Basierend auf Googles Material Design Prinzipien

## VORTEILE VON CSS FRAMEWORKS

- ✓ Schnelle Prototypenerstellung durch vorgefertigte Komponenten
- ✓ Konsistentes Design durch einheitliche Stile und Layouts
- ✓ Responsives Design out-of-the-box (z.B. Grid-Systeme)
- ✓ Große Community und umfangreiche Dokumentation
- ✓ Einfache Integration mit JavaScript-Frameworks (z.B. React, Vue)

# API DOKUMENTATION



# Was ist eine API?



# API DEFINITION

**API = Application Programming Interface**

- 🔌 Schnittstelle zwischen verschiedenen Softwarekomponenten
- 📋 Definiert, wie Programme miteinander kommunizieren
- 🎯 Legt fest: Welche Funktionen verfügbar sind
- ⬇️ Bestimmt: Welche Daten eingegeben werden müssen
- ⬆️ Spezifiziert: Welche Antworten zurückgegeben werden

# API ARTEN

## **Web APIs:**

Über das Internet zugänglich (z.B. REST, GraphQL)

## **Bibliotheks-APIs:**

Teil von Softwarebibliotheken (z.B. Java Standard Library)

## **Betriebssystem-APIs:**

Schnittstellen zu Betriebssystemfunktionen (z.B. Windows API)

## **Hardware-APIs:**

Kommunikation mit Hardwarekomponenten (z.B. Drucker-APIs)

# API DOKUMENTATION

Warum ist API Dokumentation wichtig?

- 📖 **Bedienungsanleitung:** Zeigt, wie die API verwendet wird
- ⚡ **Schneller Einstieg:** Entwickler können sofort produktiv werden
- 🐛 **Fehler vermeiden:** Korrekte Parameter und Datenformate
- 🔄 **Wartbarkeit:** Änderungen werden dokumentiert und kommuniziert
- 👥 **Teamarbeit:** Andere Entwickler verstehen die Schnittstelle
- 💰 **Geschäftswert:** Mehr Nutzer durch einfache Integration

# API-DOC FÜR JAVA

☕ **Java Klassenbibliothek:**

Extrem umfangreich (mehrere 1000 Klassen)

📋 **Schnittstelle dokumentiert:**

API-Dokumentation für Anwendungsprogrammierung

🔧 **JavaDoc Tool:**

Generiert HTML-Seiten aus dem JDK

🌐 **Online verfügbar:**

[Oracle Java SE 21 API Dokumentation](#)

# EIGENE API DOKUMENTATION

- ⭐ **Wichtiger Teil der Softwareentwicklung**
- ⏰ **Oft vernachlässigt:** Wird als zeitaufwändig empfunden
- 🔄 **Ziel:** Bessere Wiederverwendbarkeit eigener Komponenten

- 💡 Was dokumentiert werden sollte
  - 📝 Klassenname und Zweck dokumentieren
  - 🔧 Methodename und Funktionalität beschreiben
  - 📥 Typ und Anzahl der Parameter spezifizieren
  - ⚡ Wirkung und Laufzeitverhalten erklären

# BEST PRACTICES FÜR JAVADOC

-  **Prägnant und klar:** Kurz und verständlich formulieren
  -  **Zielgruppenorientiert:** Für Entwickler, die die API nutzen werden
  -  **Vollständig:** Alle öffentlichen Klassen, Methoden und Felder dokumentieren
  -  **Beispiele:** Code-Snippets zur Veranschaulichung der Nutzung
  -  **Warnungen:** Hinweise auf potenzielle Fallstricke
  -  **Aktualität:** Dokumentation bei Code Änderungen anpassen
- 
-  Fragt euch selbst, was ihr wissen wollt bzw. müsst, um euren eigenen Code verwenden zu können.
  -  **Separate Dokumentation:** Fehleranfällig und schwer aktuell zu halten
  -  **JavaDoc-Kommentare:** Direkt im Quellcode → automatische Generierung

# JAVADOC SYNTAX

Ein Dokumentationskommentar...

- beginnt mit `/**` und endet mit `*/`

- besteht aus HTML-Text mit speziellen JavaDoc-Tags (beginnend mit `@`)

- muss vor der entsprechenden Deklaration stehen

- (Klasse, Variable, Konstruktor oder Methode)

Ein Dokumentationskommentar ist zweigeteilt in

1. **Beschreibung:** Text bis zum ersten `@`-Tag, wobei der erste Satz als Zusammenfassung interpretiert wird
2. **Tag-Liste:** Alle `@`-Tags mit beschreibendem Text

 **Vollständige Tag-Referenz:**

[Oracle JavaDoc Tool Documentation](#)

## TYPISCHE JAVADOC TAGS

Kommentar-Syntax	Beschreibung
@see Klassenname oder Methodename	Verweis auf eine andere Klasse/Methode
@version Versionstext	Versions-Informationen
@author Autortext	Autoren-Informationen
@param Parametername Parametertext	Beschreibung der Parameter
@return Rückgabetext	Rückgabewert-Informationen
@throws Exception–Name Text	Beschreibung möglicher Ausnahmen/Fehler
{@link Verweis}	Im Text eingebauter Verweis (wie @see)

# BEISPIEL JAVADOC

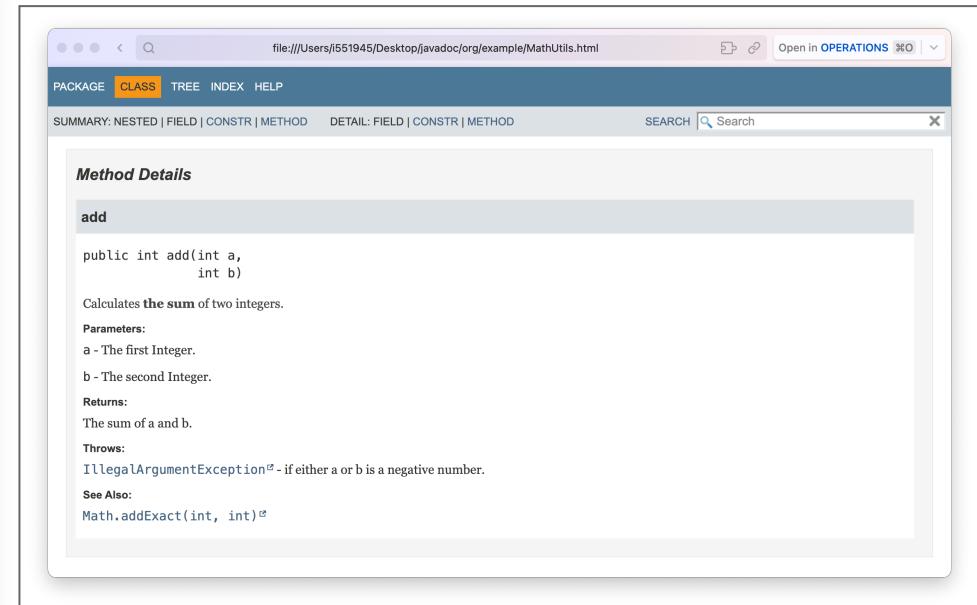


```
1  /**
2   * Calculates <b>the sum</b> of two integers.
3   *
4   * @param a The first Integer.
5   * @param b The second Integer.
6   * @return The sum of a and b.
7   * @throws IllegalArgumentException if either a or b is a negative number.
8   * @see Math#addExact(int, int)
9  */
10 public int add(int a, int b) {
11     if (a < 0 || b < 0) {
12         throw new IllegalArgumentException("Parameter must be non-negative");
13     }
14     return a + b;
15 }
```



# BEISPIEL JAVADOC

```
1  /**
2   * Calculates <b>the sum</b> of two
3   * integers.a
4   *
5   * @param a The first Integer.
6   * @param b The second Integer.
7   * @return The sum of a and b.
8   * @throws IllegalArgumentException
9   * if either a or b is a negative
10  * number.
11  * @see Math#addExact(int, int)
12  */
13 public int add(int a, int b) {
14     if (a < 0 || b < 0) {
15         throw new
IllegalArgumentException("Parameter
must be non-negative");
    }
    return a + b;
}
```



# JAVADOC GENERIEREN

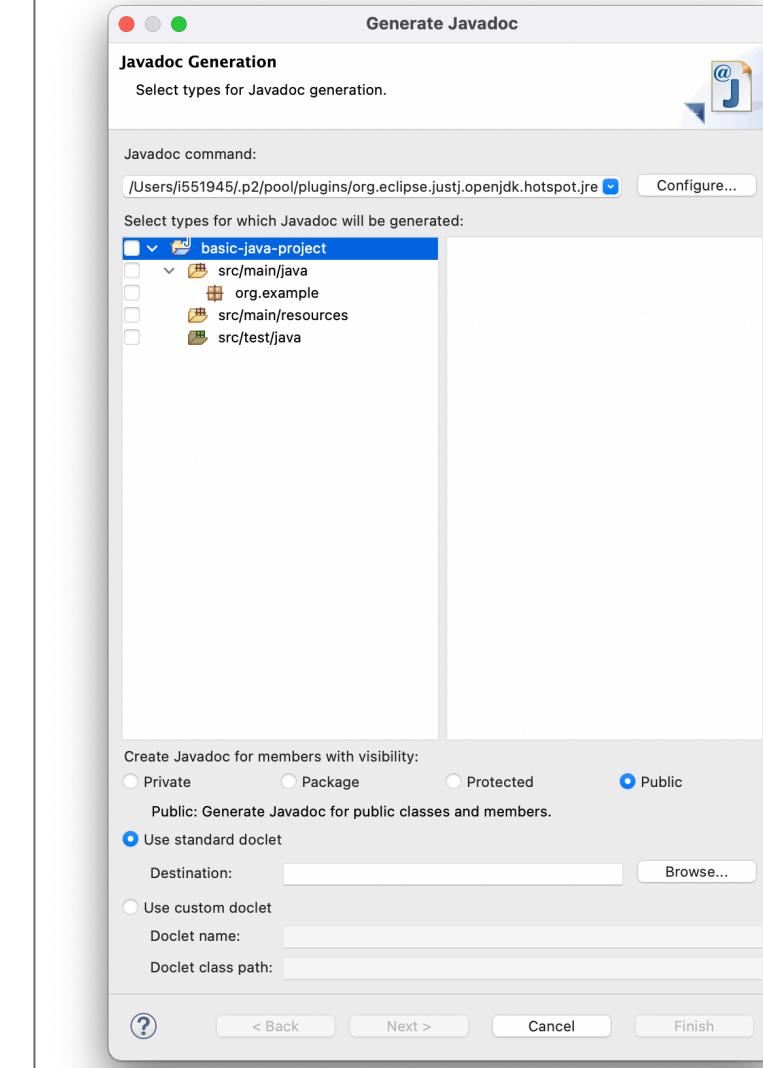
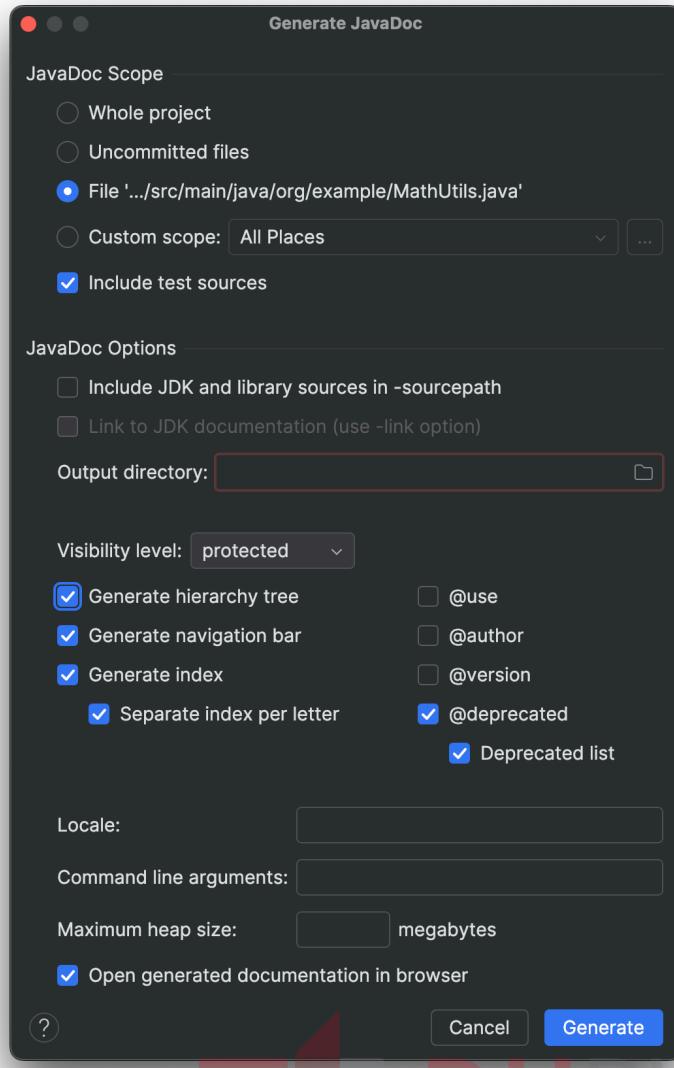


```
javadoc -d <target> Klasse.java
```

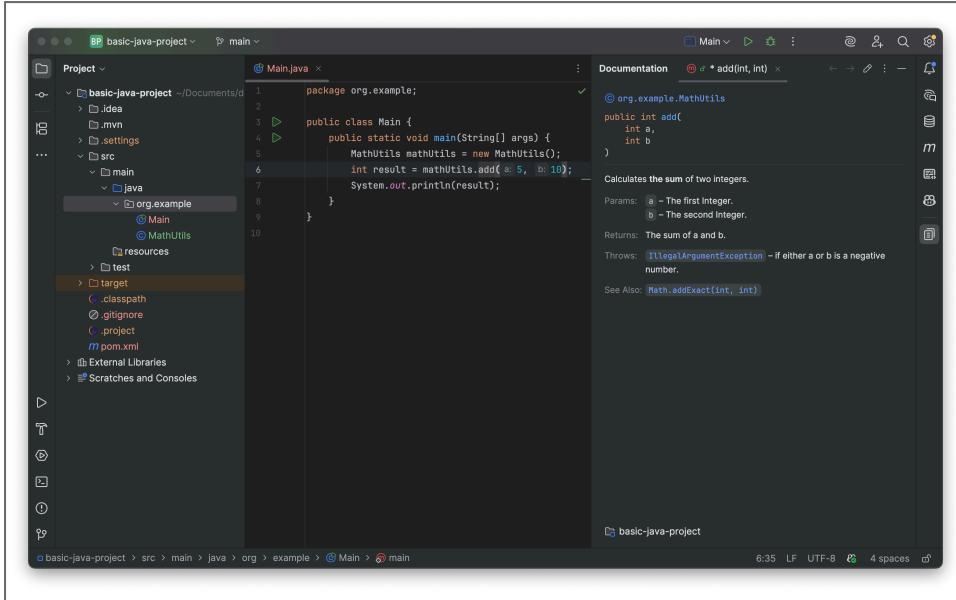
Parameter	Beschreibung
-author	@author wird mit in die Dokumentation der Klasse aufgenommen.
-version	@version wird mit in die Dokumentation der Klasse aufgenommen.
-private	Alle Klassen, Methoden und Variablen werden in die Dokumentation mit aufgenommen.
-protected	Nur protected und public Klassen, Methoden und Variablen werden in die Dokumentation mit aufgenommen. (Standardeinstellung)
-public	Nur public Klassen, Methoden und Variablen werden in die Dokumentation mit aufgenommen.

# JAVADOC GENERIEREN





# JAVADOC NUTZEN IN DER IDE

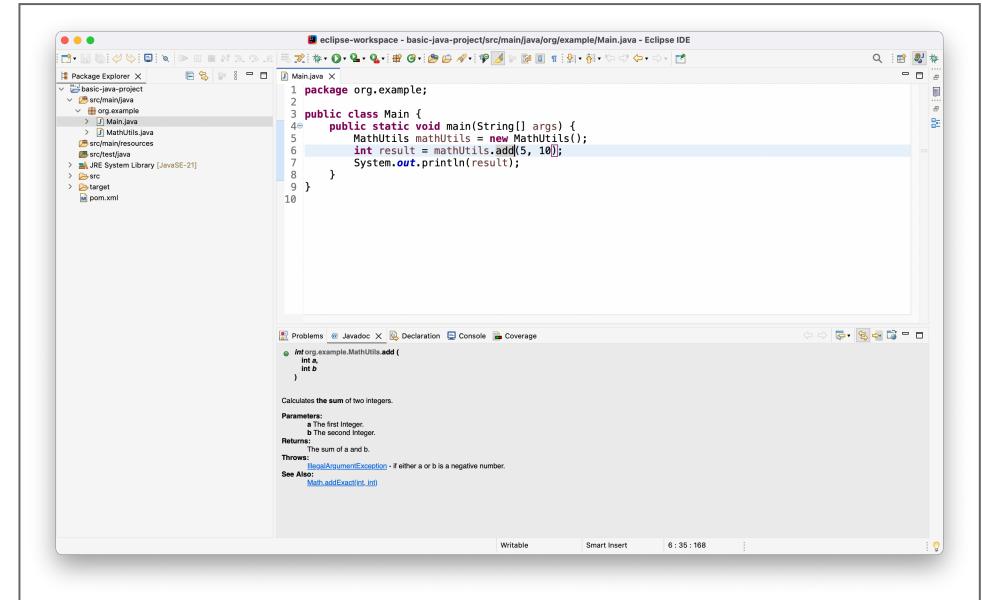


The screenshot shows a Java project named "basic-java-project" in an IDE. The main.java file contains the following code:

```
package org.example;
public class Main {
    public static void main(String[] args) {
        MathUtils mathUtils = new MathUtils();
        int result = mathUtils.add(5, 10);
        System.out.println(result);
    }
}
```

The Javadoc comment for the add method is displayed in the documentation pane:

```
Calculates the sum of two integers.
Parameters:
    a - The first Integer.
    b - The second Integer.
Returns: The sum of a and b.
Throws: IllegalArgumentException - if either a or b is a negative number.
See Also: Math.addExact\(int, int\)
```



The screenshot shows the same Java project in the Eclipse IDE. The Main.java file contains the same code as above. The Javadoc comment for the add method is shown in the documentation view:

```
Calculates the sum of two integers.
Parameters:
    a - The first Integer.
    b - The second Integer.
Returns: The sum of a and b.
Throws: IllegalArgumentException - if either a or b is a negative number.
See Also: Math.addExact\(int, int\)
```

# OPEN-API



-  **OpenAPI Specification** (früher Swagger Specification)
-  **Formaler Standard:** Definiert Spezifikationen zur Beschreibung von HTTP-APIs
-  **Maschinen- und menschenlesbar:** Ermöglicht automatische Generierung von Dokumentation, Client-Bibliotheken und Server-Stubs
-  **Werkzeuge:**
  -  **Swagger UI:** Interaktive API-Dokumentation
  -  **Swagger Codegen:** Generierung von Client- und Server-Code
  -  **Swagger Editor:** Online-Editor für OpenAPI-Spezifikationen

# SOFTWARE TESTING



## **WAS IST SOFTWARE TESTING?**

Software Testing ist der Prozess, bei dem Software auf Fehler überprüft wird, um sicherzustellen, dass sie den Anforderungen entspricht und korrekt funktioniert.



## WARUM TESTEN?

**born to  
dilly dally**



**forced to write  
unit tests**



**DHBW**  
Duale Hochschule  
Baden-Württemberg

## WARUM TESTEN?

- 🐛 Frühzeitiges Finden von Fehlern und Vermeidung von Bugs
- 🛡️ Sicherstellung der Code-Qualität und Zuverlässigkeit
- 🤝 Schaffung von Vertrauen in die Funktionalität und Stabilität der Software
- 📚 Lebendige Dokumentation des erwarteten Verhaltens
- 💰 Reduzierung des Wartungsaufwands und langfristige Einsparung von Zeit und Kosten

# ARTEN VON TESTS

## STATISCHE TESTS

- 📋 Überprüfung des Codes ohne Ausführung
- 🔍 Code-Reviews, Linting, Statische Programmanalyse
- 🛡️ Frühe Fehlererkennung und Qualitätsverbesserung

## DYNAMISCHE TESTS

- ▶ Ausführung des Codes zur Überprüfung des Verhaltens
- 🧪 Manuelle Tests, Automatisierte Tests
- 🐛 Finden von Laufzeitfehlern und Validierung der Funktionalität

# STATICHE PROGRAMMANALYSE

Statische Programmanalyse ist eine Methode zur Überprüfung von Quellcode, ohne ihn auszuführen. Sie hilft, potenzielle Fehler, Sicherheitslücken und Code-Qualitätsprobleme frühzeitig zu erkennen.

- ⚙️ Überprüfung des Quellcodes ohne Ausführung
- 🔍 Erkennung von Syntaxfehlern, Stilproblemen und potenziellen Fehlern
- 🛡️ Verbesserung der Codequalität und -wartbarkeit
- 🚀 Automatisierte Überprüfung durch Tools wie Checkstyle, PMD, FindBugs, SonarQube

# MANUELLE TESTS

Manuelle Tests sind Tests, die von Menschen durchgeführt werden, um die Funktionalität und Benutzerfreundlichkeit einer Software zu überprüfen. Sie sind besonders nützlich für explorative Tests und Usability-Tests.

## EIGENSCHAFTEN

- 👤 Durchführung durch Menschen (oft Endanwender)
- ⌚ Zeitaufwändig und fehleranfällig
- 💰 Als Regressionstest sehr teuer

## WANN SINNVOLL?

- 🤖 Bei schwer automatisierbaren Tests
- 💻 UI-Tests (Texte, Layout, Design)
- 🔍 Exploratives Testing (unerwartete Fehler finden)



**DH**BW  
Duale Hochschule  
Baden-Württemberg

# MANUELLE TESTS

## VORTEILE ✓

- ⌚ Flexibel und anpassungsfähig
- 👁️ Gut für UI- und Usability-Tests
- 🐛 Kann unerwartete Fehler finden

## NACHTEILE ✗

- ⏰ Zeitaufwändig und teuer
- ❗ Fehleranfällig (menschliche Fehler)
- ⌚ Schwer reproduzierbar

# AUTOMATISIERTE TESTS

Automatisierte Tests sind Tests, die mithilfe von Software-Tools und Skripten produktiven Code ausführen, um dessen Funktionalität und Leistung auf Korrektheit zu überprüfen.

## EIGENSCHAFTEN

- 🤖 Durchführung durch Software-Tools
- ⚡ Schnell und wiederholbar
- 💰 Kosteneffizient für Regressionstests

## WANN SINNVOLL?

- ⟳ Wenn sich Anforderungen selten ändern und Tests wiederholt ausgeführt werden müssen
- ↗ Wenn eine hohe Testabdeckung und schnelle Ausführung erforderlich sind
- 🚀 In agilen Entwicklungsprozessen und CI/CD-Pipelines



**DH**BW  
Duale Hochschule  
Baden-Württemberg

# AUTOMATISIERTE TESTS

## VORTEILE ✓

- ⚡ Schnell und wiederholbar
- 🐛 Erleichtert Refactoring
- 📈 Hohe Testabdeckung möglich

## NACHTEILE ✗

- ⌚ Initialer Aufwand zur Erstellung der Tests
- 🔧 Wartungsaufwand bei Änderungen im Code
- ❗ Nicht geeignet für UI- und Usability-Tests

# BLACK- VS. WHITE-BOX TEST

## ● BLACK-BOX TESTING

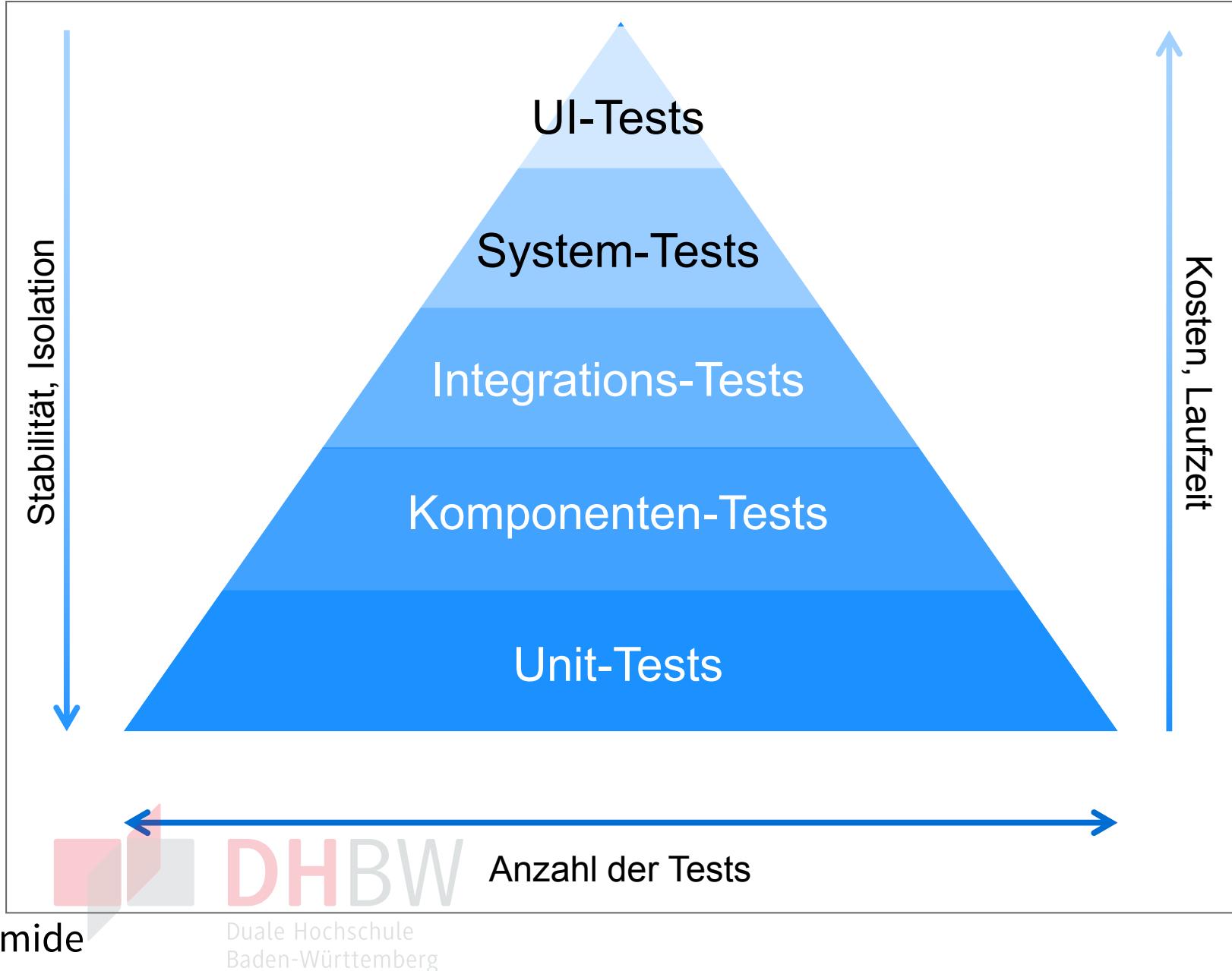
- 🎭 Testen ohne Kenntnis des internen Codes
- 🧩 Fokus auf Eingaben und erwartete Ausgaben
- 👤 Simuliert Benutzerverhalten

## ● WHITE-BOX TESTING

- 🔍 Testen mit Kenntnis des internen Codes
- 🔧 Fokus auf Code-Struktur und Logik
- 💻 Ermöglicht gezielte Tests von Funktionen und Pfaden

# TESTHIERARCHIE





Testpyramide

# TESTHIERARCHIE

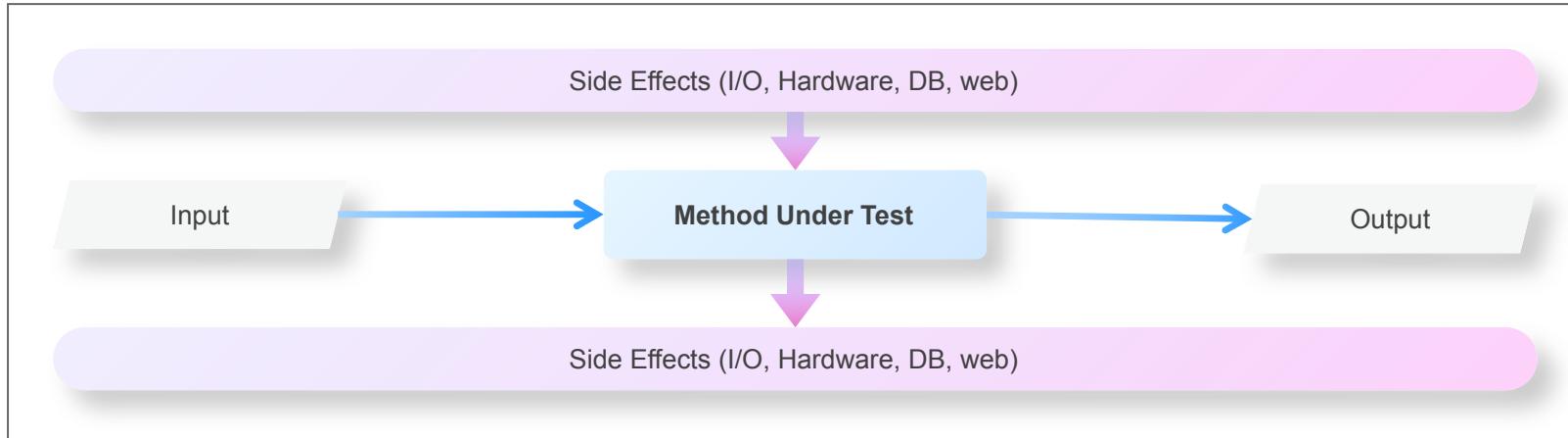
Testart	Beschreibung	Fokus
 <b>Unit-Tests</b>	Testen einzelner Funktionen oder Methoden	Isolation und Korrektheit einzelner Einheiten
 <b>Komponenten-Tests</b>	Testen einzelner Komponenten oder Module	Funktionalität und Zusammenspiel innerhalb einer Komponente
 <b>Integration-Tests</b>	Testen der Interaktion zwischen verschiedenen Komponenten	Zusammenarbeit und Datenfluss zwischen Komponenten
 <b>System-Tests</b>	Testen des gesamten Systems	Funktionalität, Leistung und Stabilität des Gesamtsystems
 <b>UI-Tests</b>	Testen der Benutzeroberfläche	Benutzerfreundlichkeit und Funktionalität der UI

## WEITERE ARTEN VON TESTS

Testart	Beschreibung	Fokus
 <b>Performance-Tests</b>	Überprüfen die Geschwindigkeit und Skalierbarkeit der Software unter Last	Leistung und Effizienz unter verschiedenen Bedingungen
 <b>Sicherheitstests</b>	Identifizieren Sicherheitslücken und Schwachstellen in der Software	Schutz vor unbefugtem Zugriff und Datenverlust
 <b>Usability-Tests</b>	Bewerten die Benutzerfreundlichkeit und das Nutzererlebnis der Software	Benutzerfreundlichkeit und Zufriedenheit
 <b>Regressionstests</b>	Stellen sicher, dass neue Änderungen keine	Stabilität und Konsistenz nach Änderungen

Testart	Beschreibung	Fokus
	bestehenden Funktionen beeinträchtigen	
 <b>End-to-End-Tests</b>	Testen den gesamten Workflow der Anwendung von Anfang bis Ende	Vollständige Abdeckung der Anwendungsfunktionalität
 <b>Smoke Tests</b>	Überprüfen die grundlegende Funktionalität der Software nach einem Build oder einer Bereitstellung	Schnelle Validierung der grundlegenden Funktionen

# TESTING IDEE



## Eingabe

Definieren der Eingabedaten für den Testfall.

## Erwartetes Ergebnis

Festlegen des erwarteten Ergebnisses basierend auf den Eingabedaten.

## Side-Effects

Nebenwirkungen, welche die Ausführung der Funktion beeinflussen können und durch sie auftreten können.

**Überprüfen des tatsächlichen Ergebnisses gegenüber dem erwarteten Ergebnis bei bekannter Eingabe und Minimierung von Side-Effects.**

## FIRST - PRINZIP

- ⚡ **Fast:** Tests sollten schnell ausgeführt werden, um häufiges Testen zu ermöglichen.
- 🧩 **Independent:** Tests sollten unabhängig voneinander sein, sich gegenseitig nicht beeinflussen und Seiteneffekte vermeiden.
- ⟳ **Repeatable:** Tests sollten unter verschiedenen Bedingungen reproduzierbar sein.
- ✓ **Self-Validating:** Tests sollten automatisch überprüfen, ob das Ergebnis korrekt ist.
- 📝 **Timely:** Tests sollten früh im Entwicklungsprozess geschrieben werden, um Fehler frühzeitig zu erkennen.

## WAS SOLLTEN TESTFÄLLE ABDECKEN?

- Positive Testfälle**
  - 👉 Korrekte Werte
- Negative Testfälle**
  - 👉 Inkorrekte Werte
- Äquivalenzklassen**
  - 👉 Wertebereiche mit gleichem Verhalten
- Grenzwertanalyse**
  - 👉 Werte an den Grenzen der Äquivalenzklassen und deren Nähe
- Kombinationen von Eingabewerten**
  - 👉 z.B. durch all-pairs Testing

# WAS SOLLTEN TESTFÄLLE ABDECKEN?

## BEISPIEL

```
boolean isPalindrome(String text) {...}
```

text = ""	true / Exception	Spezifikation: Ist Leerer String ein Palindrom?
text = "a"	true	Ein einzelnes Zeichen ist ein Palindrom
text = "abcd"	false	Kein Palindrom
text = "anna"	true	Klassisches Palindrom
text = "Anna"	true / false	Spezifikation: Groß-/Kleinschreibung beachten?
text = null	Exception	Spezifikation: Wie mit null umgehen?

## ARRANGE - ACT - ASSERT

Die "Arrange - Act - Assert" (AAA) Methode ist ein bewährtes Muster für das Schreiben von Tests, das hilft, Tests klar und verständlich zu strukturieren.

### ARRANGE

-  Vorbereiten der Testdaten und -bedingungen.
-  Beispiel: Erstellen eines Arrays von Zahlen für einen Sortieralgorithmus.

### ACT

-  Ausführen der zu testenden Funktion.
-  Beispiel: Aufrufen der Sortierfunktion mit dem vorbereiteten Array.

### ASSERT

-  Prüfen, ob das Ergebnis den Erwartungen entspricht.
-  Fehler melden, wenn das Ergebnis falsch ist.

# ARRANGE - ACT - ASSERT

## Class Under Test (CUT)

```
1 class CustomMath {  
2     private static double sum = 0;  
3  
4     public static int add(double[]  
5         numbers) {  
6         for(int i=0; i <  
7             numbers.length; i++) {  
8             sum += numbers[i];  
9         }  
10    }  
11    return sum;  
12 }
```

## Test Treiber Klasse

```
1 class TestDriver {  
2     public static void main(String[] args)  
3     {  
4         // Arrange  
5         double[] input = {1.0, 2.0, 3.0};  
6         int expectedResult = 6.0;  
7  
8         // Act  
9         int actualResult =  
10            CustomMath.add(input);  
11  
12         // Assert  
13         boolean testPassed = (actualResult  
14             == expectedResult);  
15         System.out.println(testPassed ?  
16             "OK!" : "Failure!");  
17     }  
18 }
```

Konsolenausgabe:



# PROBLEM BEI DIESEM TEST

## Class Under Test (CUT)

```
1 class CustomMath {  
2     private static double sum = 0;  
3  
4     public static int add(double[]  
numbers) {  
5         for(int i=0; i <  
numbers.length; i++) {  
6             sum += numbers[i];  
7         }  
8         return sum;  
9     }  
10 }
```

## Test Treiber Klasse 2

```
1 class TestDriver2 {  
2     public static void main(String[] args)  
3     {  
4         System.out.println("Test 1:");  
5         double[] input1 = {1.0, 2.0, 3.0};  
6         int expectedResult1 = 6.0;  
7         int actualResult1 =  
8         CustomMath.add(input1);  
9         boolean testPassed1 = (actualResult1  
== expectedResult1);  
10        System.out.println(testPassed1 ?  
11            "OK!" : "Failure!");  
12  
13        System.out.println("Test 2:");  
14        double[] input2 = {6.0};  
15        int expectedResult2 = 6.0;  
16        int actualResult2 =  
17        CustomMath.add(input2);  
18        boolean testPassed2 = (actualResult2  
== expectedResult2);  
19        System.out.println(testPassed2 ?  
20            "OK!" : "Failure!");  
21    }  
22 }
```



Konsolenausgabe:



```
> Test 1:  
> OK!  
> Test 2:  
> Failure!
```



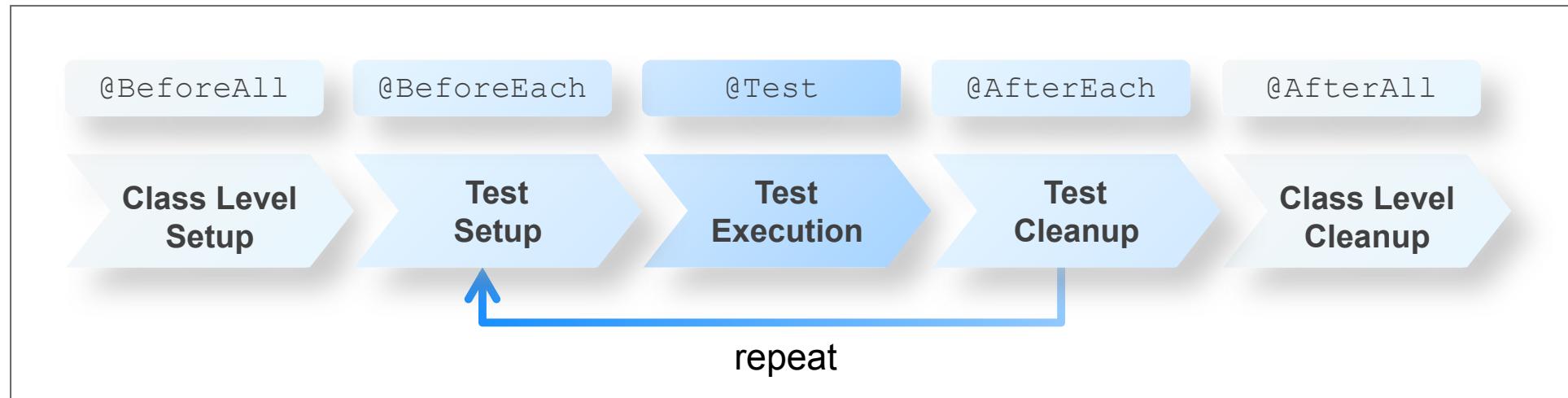
# JUNIT

- ✍ Erleichtert das Schreiben von Tests in Java
- ⚙️ Bietet Annotationen zur Kennzeichnung von Testmethoden
- ✓ Stellt *Assertions* zur Überprüfung von erwarteten Ergebnissen bereit
- ⟳ Lässt sich einfach in Entwicklungsumgebungen (IDEs) integrieren
- 🚀 Unterstützt Test Suites und Test Runner für die Organisation und Ausführung von Tests

# JUNIT - BEISPIEL

```
● ● ●  
1 class CustomMathTest {  
2  
3     @Test  
4     void testAdd() {  
5         // Arrange  
6         double[] input = {1.0, 2.0, 3.0};  
7         int expectedResult = 6.0;  
8  
9         // Act  
10        int actualResult = CustomMath.add(input);  
11  
12        // Assert  
13        assertEquals(expectedResult, actualResult, "The sum should be 6.0");  
14    }  
15 }
```

# JUNIT - TEST LIFECYCLE



Annotation	Zeitpunkt	Beschreibung
@BeforeAll	Einmalig vor allen Tests	Wird einmalig vor allen Testmethoden ausgeführt (Initialisierung).
@BeforeEach	Vor jedem Test	Wird vor jeder Testmethode ausgeführt (Initialisierung).
@Test	Testmethode	Kennzeichnet eine Methode als Testfall.
@AfterEach	Nach jedem Test	Wird nach jeder Testmethode ausgeführt (Aufräumarbeiten).

Annotation	Zeitpunkt	Beschreibung
@AfterAll	Einmalig nach allen Tests	Wird einmalig nach allen Testmethoden ausgeführt (Aufräumarbeiten).

# JUNIT - ASSERTIONS

Assertion	Beschreibung
<code>assertEquals(expected, actual, message)</code>	Überprüft, ob zwei Werte gleich sind.
<code>assertNotEquals(unexpected, actual, message)</code>	Überprüft, ob zwei Werte ungleich sind.
<code>assertTrue(condition, message)</code>	Überprüft, ob eine Bedingung wahr ist.
<code>assertFalse(condition, message)</code>	Überprüft, ob eine Bedingung falsch ist.
<code>assertNull(object, message)</code>	Überprüft, ob ein Objekt null ist.
<code>assertNotNull(object, message)</code>	Überprüft, ob ein Objekt nicht null ist.

## Assertion

---

## Beschreibung

`assertThrows(expectedType, executable, message)`

Überprüft, ob eine bestimmte Exception geworfen wird.

# JUNIT UMFASSENDES BEISPIEL

```
1 public class PalindromeChecker {  
2     public boolean check(String text)  
3     {  
4         if(text == null) {  
5             throw new  
IllegalArgumentException("Cannot  
check for palindrome on null");  
6         }  
7         if (text.isEmpty()) {  
8             return false;  
9         }  
10        if(text.length() == 1) {  
11            return true;  
12        }  
13  
14        String cleanedText = text  
15        .trim()  
16        .toLowerCase();  
17        String reversedText = new  
StringBuilder(cleanedText)  
18        .reverse()  
19        .toString();  
20  
21        return  
reversedText.equals(cleanedText);  
22    }  
23}
```



**DHBW**  
Duale Hochschule  
Baden-Württemberg

```
1 public class PalindromeCheckerTest {  
2     PalindromeChecker  
palindromeChecker;  
3  
4     @BeforeEach  
5     public void setUp() {  
6         palindromeChecker = new  
PalindromeChecker();  
7     }  
8  
9     @Test  
10    @DisplayName("check should throw  
IllegalArgumentException for null  
input")  
11    public void testCheck_NullInput()  
12    {  
13        assertThrows(IllegalArgumentException.class,  
14        () ->  
palindromeChecker.check(null));  
15    }  
16  
17    @Test  
18    @DisplayName("check should return  
false for empty string")  
19    public void  
testCheck_EmptyString() {  
20        assertFalse(palindromeChecker.check(  
21        ""));  
22    }  
23  
24    @Test  
25    @DisplayName("check should return  
true for single character string")  
26    public void
```



```
24  public void
25    testCheck_SingleCharacter() {
26      assertTrue(palindromeChecker.check(
27        "a"));
28
29      @Test
30      @DisplayName("check should return
31      false for non-palindrome string")
32      public void
33        testCheck_NonPalindrome() {
34          assertFalse(palindromeChecker.check(
35            "hello"));
36
37          @Test
38          @DisplayName("check should return
39          true for palindrome string")
40          public void testCheck_Palindrome()
41          {
42            assertTrue(palindromeChecker.check(
43              "madam"));
44
45            @Test
46            @DisplayName("check should return
47            true for palindrome with mixed
48            case")
49            public void
50              testCheck_Palindrome_MixedCase() {
51                assertTrue(palindromeChecker.check(
52                  "Madam"));
53
54                @Test
55                @DisplayName("check should return
56                true for palindrome with leading and
57                trailing spaces")
58                public void
59                  testCheck_Palindrome_LeadingTrailing
```

```
49 Spaces() {
50     assertTrue(palindromeChecker.check("madam"));
51 }
```



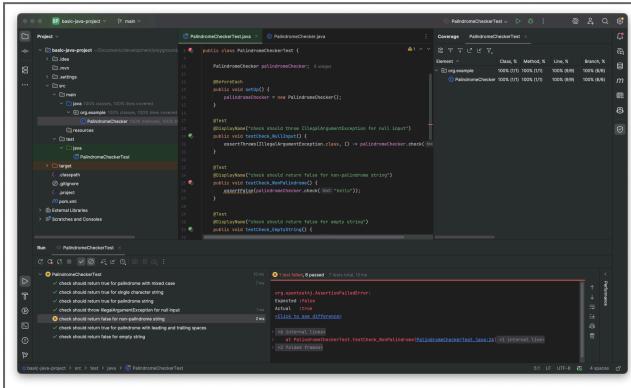
# TEST IN DER IDE

The screenshot shows a Java IDE interface with the following components:

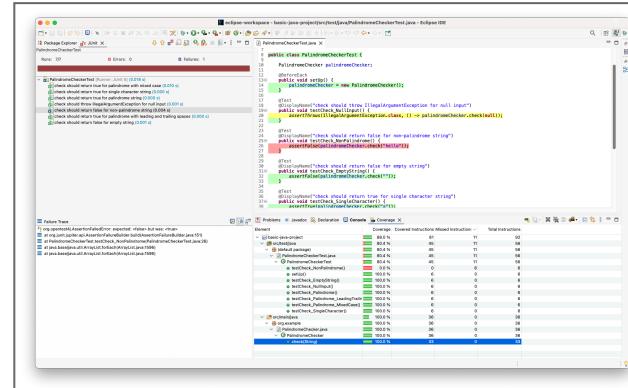
- Left Sidebar:** Includes a "Package Explorer" showing a project named "PalindromeCheckerTest" with 7/7 runs, and a "Failure Trace" section listing errors from org.junit.AssertionFailedError.
- TEST EXPLORER:** Shows 6/7 tests in the "basic-java-project" package. The "PalindromeCheckerTest" class contains 14ms of tests:
  - testCheck\_NullInput() (check should throw IllegalArgumentException)
  - testCheck\_NonPalindrome() (check should return false)
  - testCheck\_EmptyString() (check should return false)
  - testCheck\_SingleCharacter() (check should return true)
  - testCheck\_Palindrome() (check should return true for "racecar")
  - testCheck\_Palindrome\_MixedCase() (check should return true for "Racecar")
  - testCheck\_Palindrome\_LeadingTrailingSpaces() (check should return true for "racecar")
- TEST COVERAGE:** Shows 100.00% coverage for "PalindromeChecker.java" with methods `<init>` and `check(String)`.
- Code Editor:** Displays the source code for `PalindromeCheckerTest.java`. The code uses JUnit 5 annotations (`@Test`, `@DisplayName`, `assertThrows`, `assertFalse`) to write tests for the `PalindromeChecker` class.
- PROBLEMS:** Shows 1 error related to a failed assertion in the test code.
- TEST RESULTS:** Shows the results of the test run, including test cases and their outcomes.
- Bottom Status Bar:** Shows "Java: Ready" and "SonarQube focus: overall code".
- Bottom Right:** Shows status information like "Ln 23, Col 10" and "Spaces: 4".

# TEST IN DER IDE

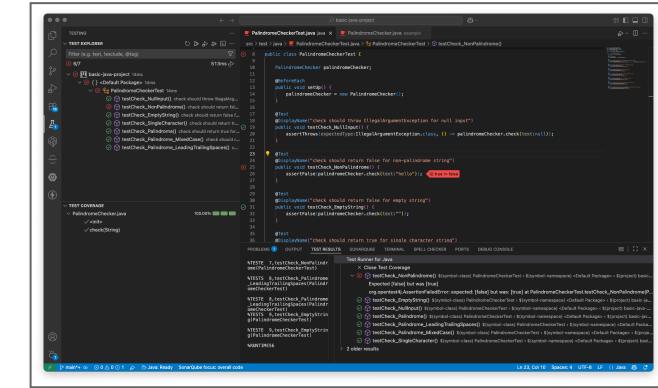
## INTELLIJ IDEA



## ECLIPSE IDE

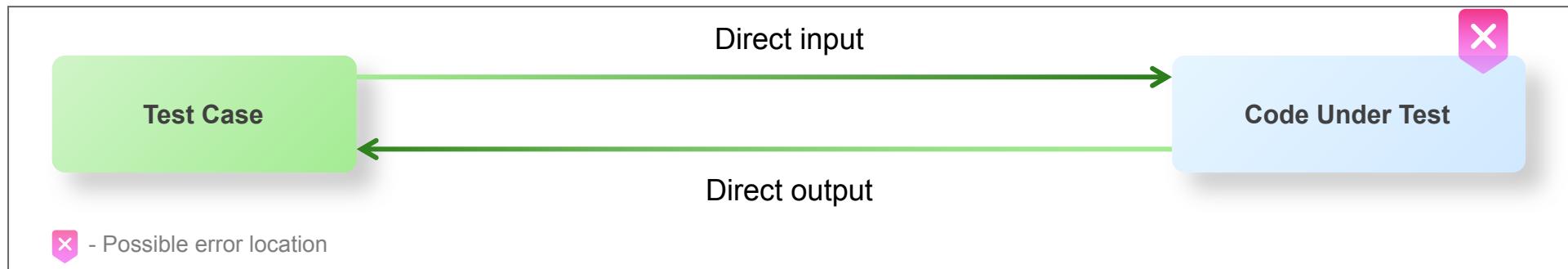


## VISUAL STUDIO CODE



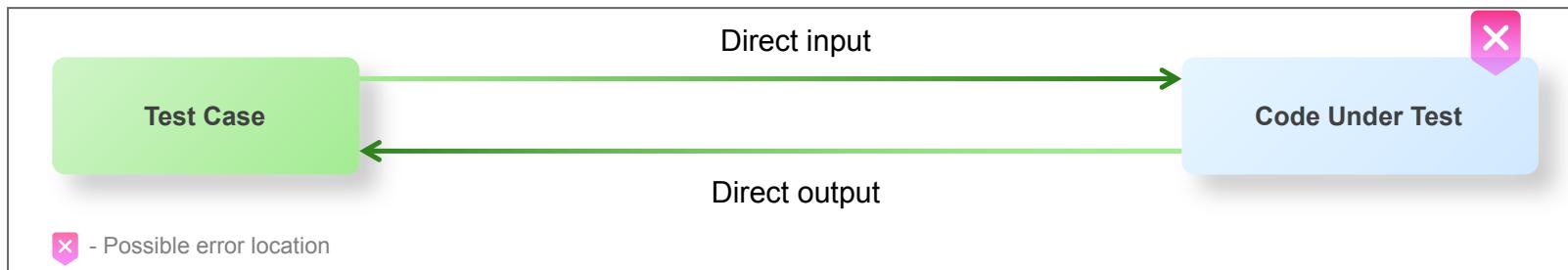
# TEST ISOLATION

## TEST-CODE OHNE ABHÄNGIGKEITEN



# TEST ISOLATION

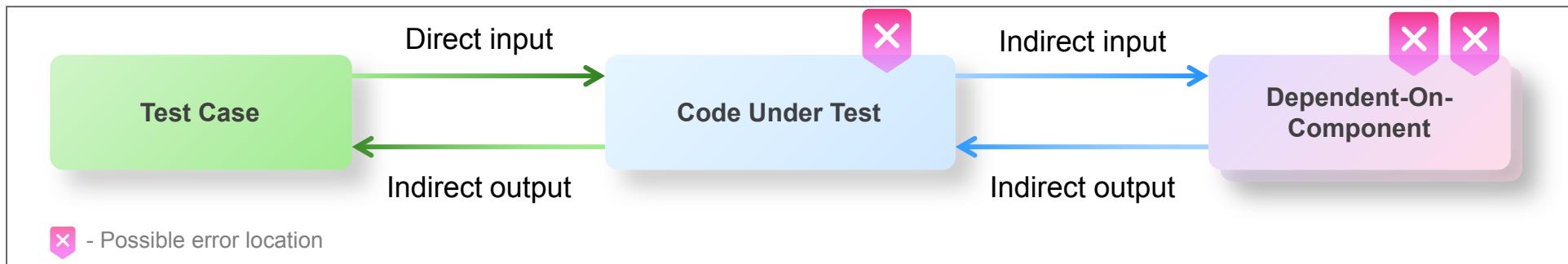
## TEST-CODE OHNE ABHÄNGIGKEITEN



- 🌈 Im Idealfall hat der zu testende Code keine Abhängigkeiten
- ⚡ Tests sind schnell ausführbar
- ⟳ Tests sind unabhängig und wiederholbar
- 🎯 Fehlerquellen können nur im code-under-test liegen

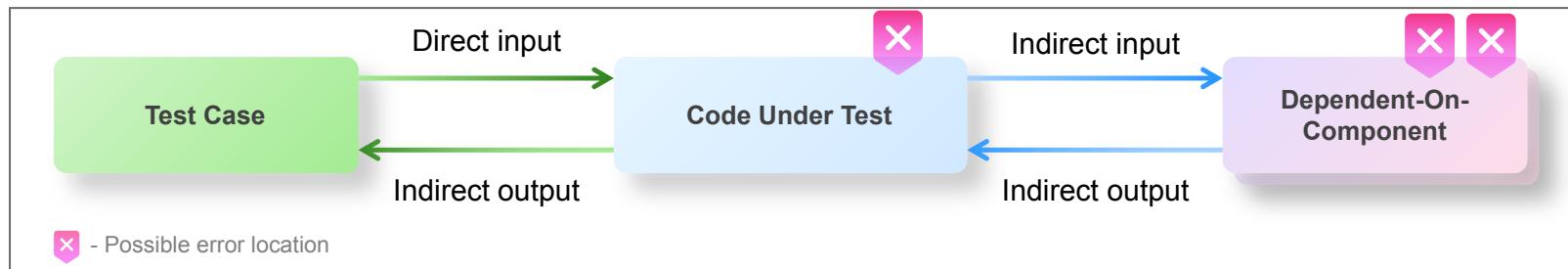
# TEST ISOLATION

## TEST-CODE MIT ABHÄNGIGKEITEN



# TEST ISOLATION

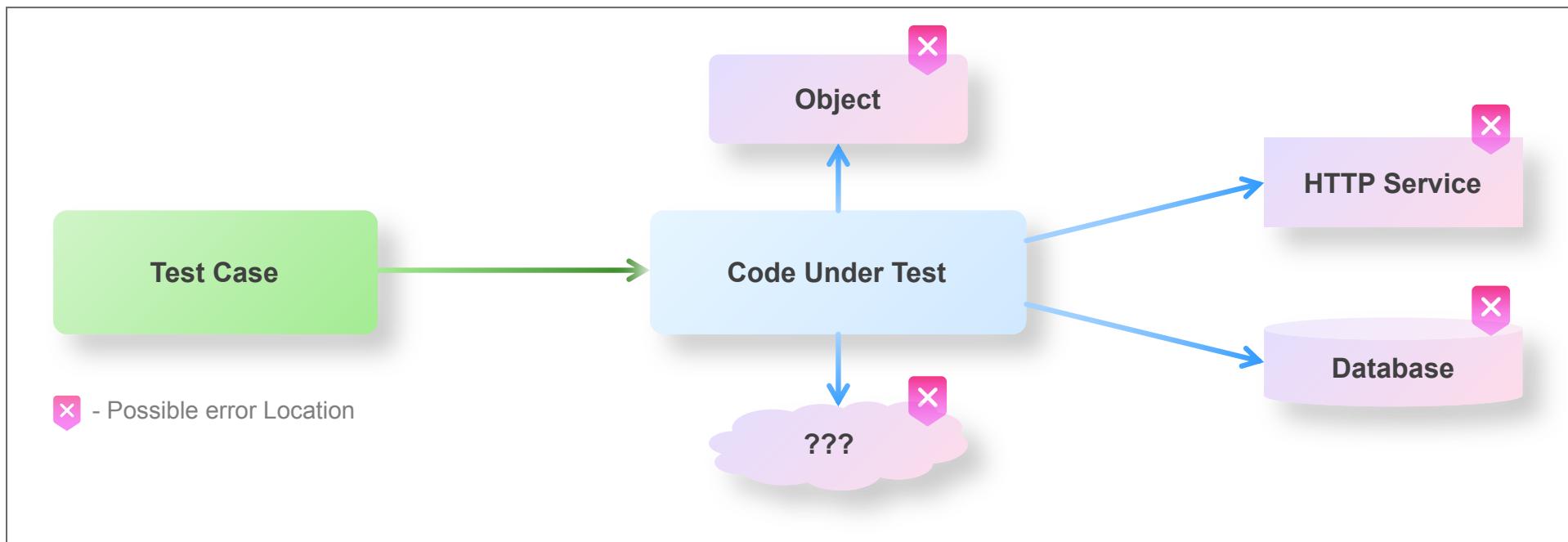
## TEST-CODE MIT ABHÄNGIGKEITEN



- ⚠ In der Praxis hat der zu testende Code oft Abhängigkeiten
- 🐢 Tests können langsam und unzuverlässig werden
- ❗ Fehlerquellen können im code-under-test oder in Abhängigkeiten liegen

# TEST ISOLATION

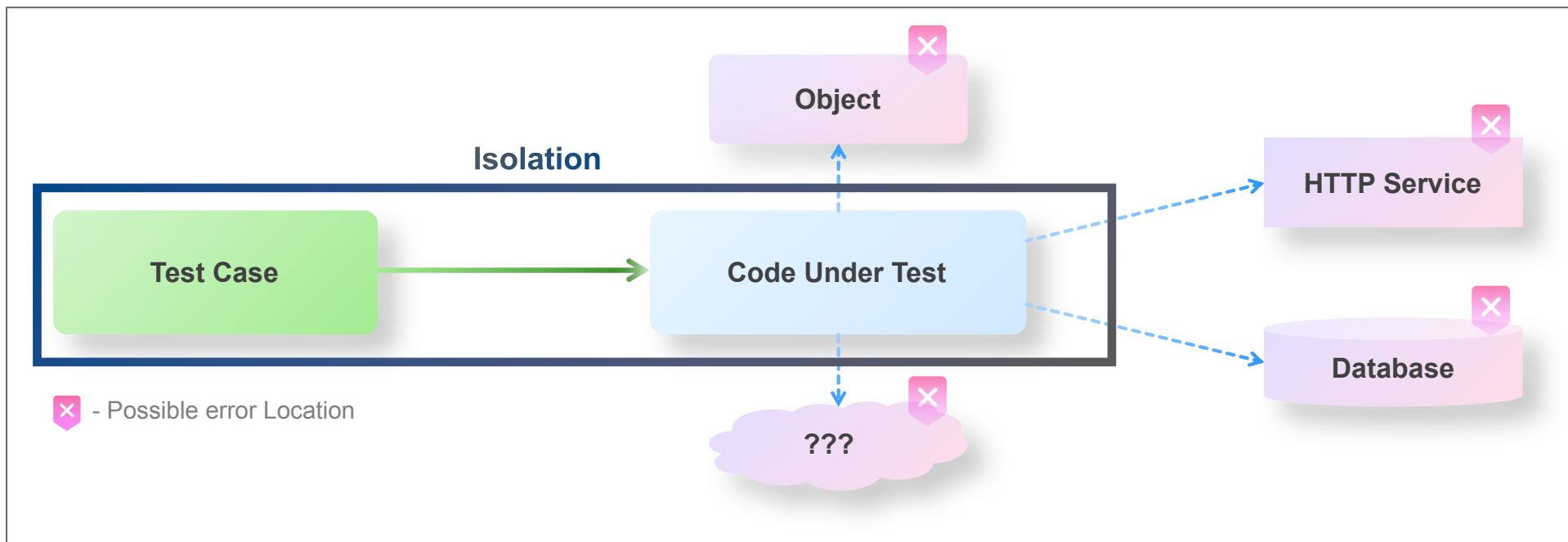
## TEST-CODE MIT ABHÄNGIGKEITEN



👉 Wir wollen die Abhängigkeiten isolieren, um die Tests schnell, unabhängig und wiederholbar zu machen.

# TEST ISOLATION

## TEST-CODE MIT ABHÄNGIGKEITEN

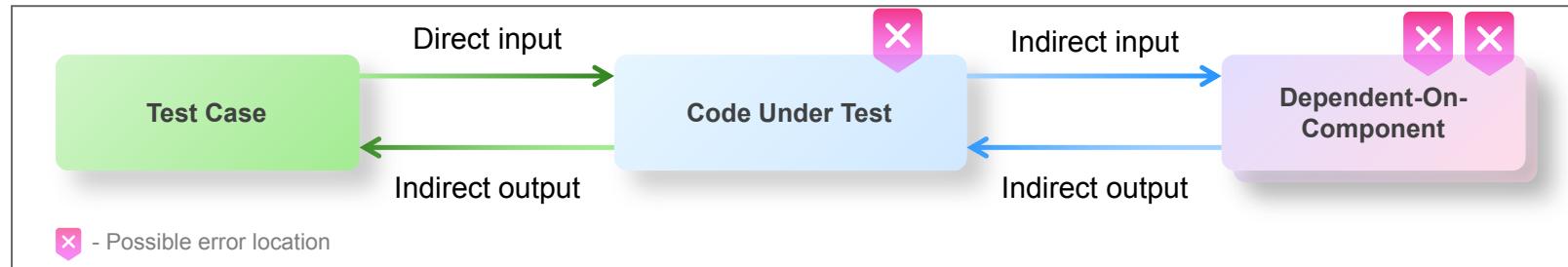


👉 Wir wollen die Abhängigkeiten isolieren, um die Tests schnell, unabhängig und wiederholbar zu machen.

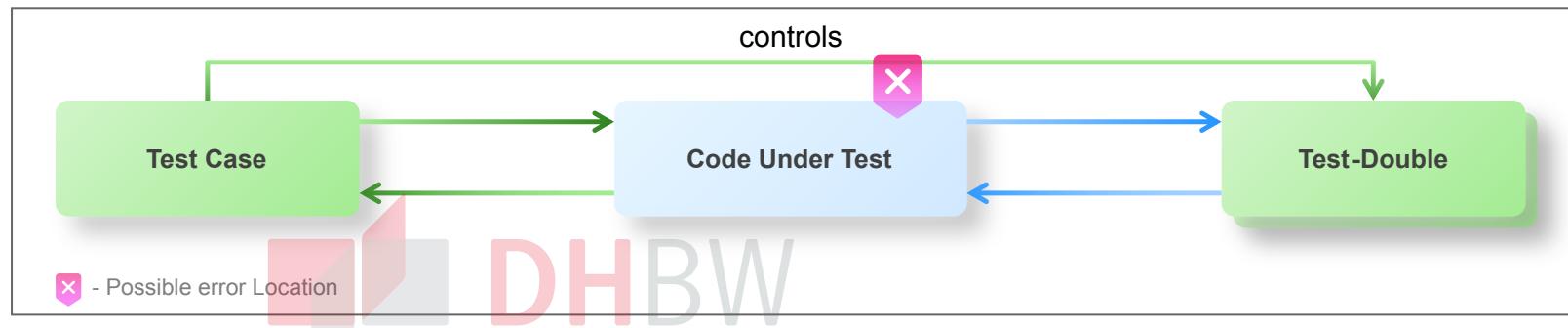
# TEST ISOLATION

## TEST DOUBLES

Vorher:

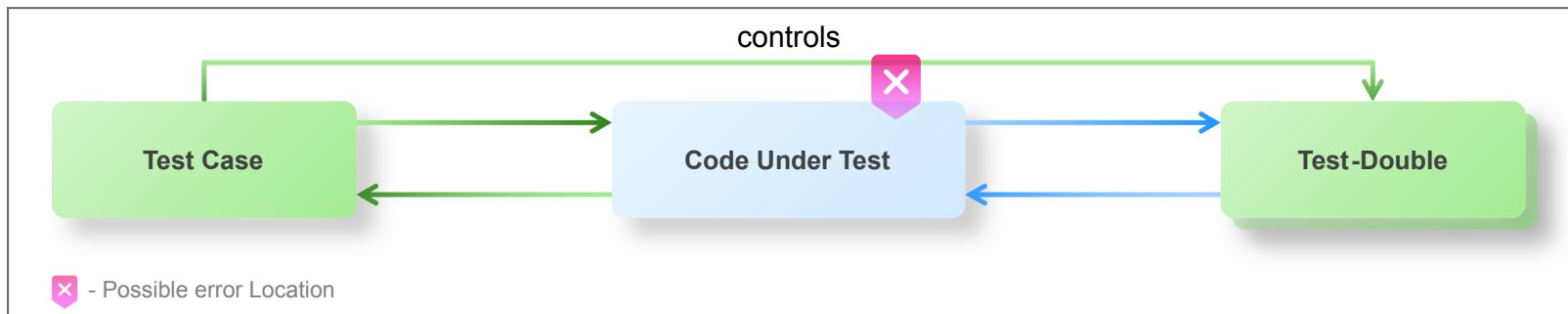


Nachher:



# TEST ISOLATION

## TEST DOUBLES



- 💡 Test Doubles sind Objekte, die die Abhängigkeiten des zu testenden Codes ersetzen
- 🎯 Sie ermöglichen es dem Test, das Verhalten der Abhängigkeiten zu kontrollieren und zu simulieren
- 🎭 Dadurch können Tests isoliert und reproduzierbar durchgeführt werden

# TEST DOUBLES

## ARTEN VON TEST DOUBLES

	 Beschreibung	 Ziel
Dummy	Objekte, die als Platzhalter übergeben werden, aber über kein Verhalten verfügen.	Erfüllung von Abhängigkeiten, ohne tatsächliches Verhalten zu simulieren.
Fake	Objekte mit einer minimalen, dem Testfall genügenden Implementierung.	Ersatz von komplexen Abhängigkeiten durch einfachere Alternativen.
Stub	Liefert vordefinierte Antworten auf bestimmte Aufrufe.	Kontrolle der indirekten Eingaben des zu testenden Codes.
Spy	Protokolliert und überprüft ggf., ob bestimmte Interaktionen stattgefunden haben.	Überprüfung der indirekten Ausgaben des zu testenden Codes.



## Beschreibung

**Mock** Objekte, die vordefinierte Antworten liefern und zusätzlich das erwartete Verhalten überprüfen.

## Ziel

Umfassende Kontrolle und Überprüfung der Interaktionen mit Abhängigkeiten.

# COVERAGE

Woher weiß ich, dass ich genug getestet habe?



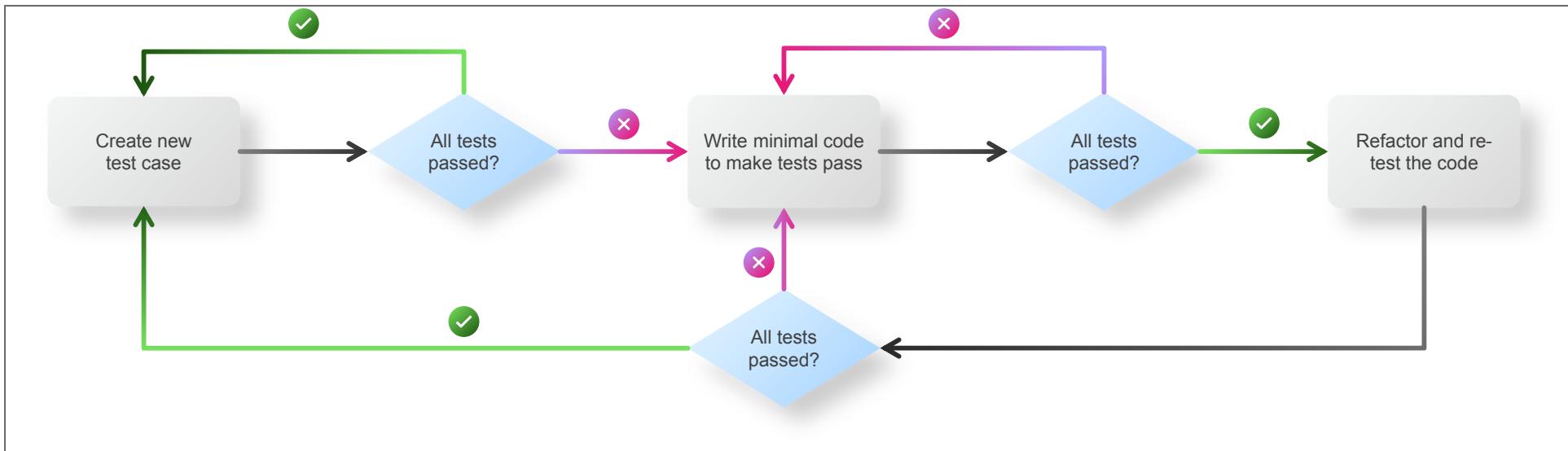
## COVERAGE

- 📊 Code Coverage misst den Anteil des Codes, der durch Tests abgedeckt ist
- ✓ Höhere Coverage bedeutet nicht automatisch bessere Tests
- ⚠️ Wichtiger ist die Qualität der Tests und die Abdeckung verschiedener Szenarien
- 🎯 Ziel ist es, kritische Pfade und Randfälle abzudecken
- 💯 100% Coverage selten erreichbar
- 🏆 Eine Coverage von 80-90% ist ein guter Richtwert, aber wichtiger ist die Qualität der Tests

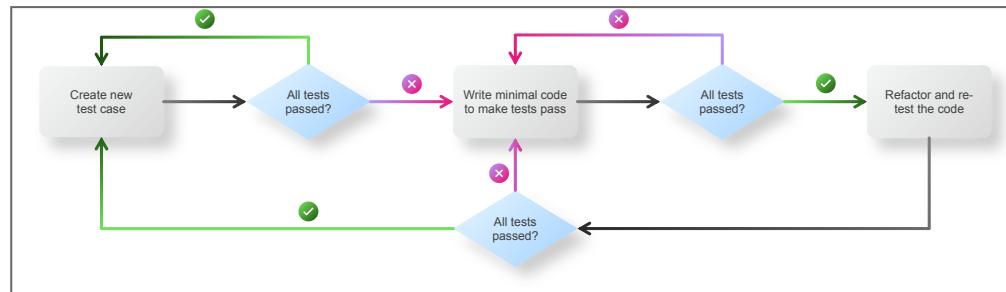
# COVERAGE ARTEN

Art	Beschreibung
 Line Coverage	Misst den Prozentsatz der ausgeführten Codezeilen.
 Statement Coverage	Misst den Prozentsatz der ausgeführten Anweisungen.
 Branch Coverage	Misst den Prozentsatz der ausgeführten Verzweigungen (if, switch).
 Path Coverage	Misst den Prozentsatz der durchlaufenen Pfade im Kontrollflussgraphen.

# TEST-DRIVEN DEVELOPMENT



# TEST-DRIVEN DEVELOPMENT



## VORTEILE

- 💻 Zwingt Entwickler, sich mit allen möglichen Programmsituationen auseinanderzusetzen
- 🧪 Garantiert Existenz von Tests
- 🐛 Verschleppung von Fehlern wird verhindert
- ✅ Es werden genau die Anforderungen erfüllt – und nicht mehr!
- 🚀 Es wird genau nur soviel Produktivcode geschrieben wie benötigt
- ✨ Das Interface-Design wird in der Regel sehr sauber

## NACHTEILE

- 🤔 Akzeptanz von Entwicklern nicht immer gegeben
- 💪 Konsequente Umsetzung notwendig
- 🚧 Wenig geeignet für Prototyping

## Speaker notes

