Einführung in EIDI 1
— Informatik, Java und „Hello World“ —

Anne Brüggemann-Klein

Dozentin und Übungsleitung EIDI 1 und PGdP

Prof.Dr. Anne Brüggemann-Klein
Dr. Andreas Reuß
Julian Kranz
Raphaela Palenta

plus ≈ 60 Tutor/innen

Was erwartet Sie heute ?

- Konzept EIDI 1 und PGdP
  - … unter dem Aspekt Informatik
  - … unter dem Aspekt Grundlagen Informatik / Programmierung
  - … unter dem Aspekt Praxis (Coding in Java)
- Organisatorisches
- Einordnung:
  Problemlösung, Rechnerarchitektur und Programmiersprachen
- Erste Schritte mit Java
  - das Java-Ökosystem
  - die ersten Java-Programme („Hello World“)
    - Einlesen und Ausgeben von Text
    - die Klasse String
    - Variablen

Konzept EIDI 1 und PGdP: Aspekt Informatik

- *Computer science is to the information revolution what mechanical engineering was to the industrial revolution.*
  - Design von Artefakten: Eingebettete Systeme (Automotive), Roboter, Steuerung von Flugzeugen
    - Engineering / E-Technik, Mathematik
  - Berechnungen, Algorithmen: Simulationen (Klima, Versicherungen, Epidemien), Routenberechnungen
    - Ökonomie, Mathematik (Numerik)
  - Interaktion Mensch / Computer
    - Kognitionspsychologie
- Klammer *Infor**ma**tik*
  - Problemlösungen mit Hilfe automatisiertem Verarbeiten von Information
  - zentrale Aufgabe: Umgang mit Komplexität
  - EIDI 1 / PGdP: Umsetzung von Problemlösungen in Code

Konzept EIDI 1 und PGdP: Aspekt Grundlagen

Einmalige Kombination:
Tandem von Grundlagen und Praxis des Programmierens

[*a principled approach to programming*]

- Grundlagen der Informatik: konzeptuell, "unplugged"
  - informatisches Denken (Computational thinking) zur Problemlösung, zum Umgang mit Komplexität
    - wesentliches Mittel: Abstraktion (Modelle)
- Grundlagen des Programmierens: konzeptuell, "unplugged"
  - mit Computern ausführbare Strategien und Verfahren zur Problemlösung: Algorithmen
  - Organisation von Information/Daten: Datenstrukturen
  - Konzepte von objekt-orientierten Programmiersprachen
  - Prinzipien und Vorgehensweisen

Konzept EIDI 1 und PGdP: Aspekt Praxis

- Praxis des Programmierens „im Kleinen“
  Formulierung von Problemlösungen / Algorithmen in einer Form, die Computer verstehen und ausführen können
  - Programmierung mit Java
    - konzeptuell reich
    - gut etabliert, besonders im Business-Bereich
    - anschlussfähig
      (z.B. für C++, funktionale Sprachen, Excel / SQL / XSLT)
  - umfassende Praxis: Entwurf, Kodierung, Debuggen, Testen
  - Qualitätskriterien: Kommunikation, Einfachheit, Flexibilität (Kent Beck) und professionelle Praktiken / Patterns dafür
  - Werkzeuge (Entwicklungsumgebung)
  - Entwicklung eigener Handlungsfähigkeit
    „auf den Schultern von Riesen“ (Prinzipien, Bibliotheken)

Konzept EIDI 1 und PGdP: Aspekt Praxis

- Capstone-Projekt im PGdP nach Weihnachten:
  Spiel der Pacman-Klasse mit viel Raum für Phantasie

Konzept EIDI 1 und PGdP: Aspekt Voraussetzungen

- Einführung: Anfangsgründe
  ohne spezielle Voraussetzungen
  - studierfähig
  - interessiert
  - Fähigkeit zu Planung und Organisation
  - systematisches Denken
  - Abstraktionsfähigkeit
  - Genauigkeit
  - Kreativität
  - mathematische Kenntnisse auf Abiturniveau
  - Lernfähigkeit und Anstrengungsbereitschaft
    - 12 von 30 ETCS (2/5)
    - 2 volle Arbeitstage in einer 5-Tage-Woche

Was kommt als nächstes ?

- Konzept EIDI 1 und PGdP
  - … unter dem Aspekt Informatik
  - … unter dem Aspekt Grundlagen Informatik / Programmierung
  - … unter dem Aspekt Praxis (Coding in Java)
- Organisatorisches
- Einordnung:
  Problemlösung, Rechnerarchitektur und Programmiersprachen
- Erste Schritte mit Java
  - das Java-Ökosystem
  - die ersten Java-Programme („Hello World“)
    - Einlesen und Ausgeben von Text
    - die Klasse String
    - Variablen

Separate Folien

Dieser Teil wurde in der Vorlesung am 14.10. nicht behandelt.

Was kommt als nächstes ?

- Konzept EIDI 1 und PGdP
  - … unter dem Aspekt Informatik
  - … unter dem Aspekt Grundlagen Informatik / Programmierung
  - … unter dem Aspekt Praxis (Coding in Java)
- Organisatorisches
- Einordnung:
  Problemlösung, Rechnerarchitektur, Programmiersprachen

- Erste Schritte mit Java
  - das Java-Ökosystem
  - die ersten Java-Programme („Hello World“)
    - Einlesen und Ausgeben von Text
    - die Klasse String
    - Variablen

Einordnung: Problemlösung

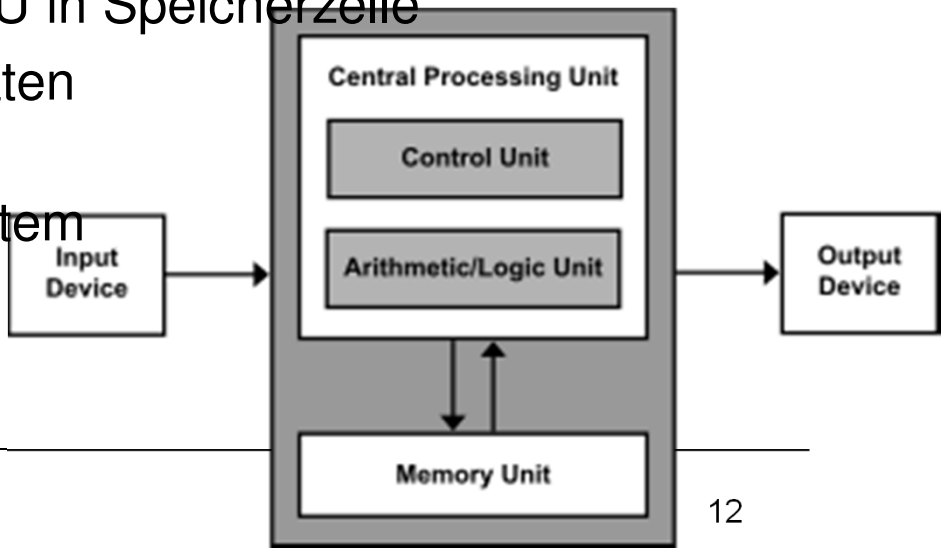
Problemlösung in Phasen

- Verstehen und Formulierung des Problems (Modellierung, Spezifikation)
- Entwicklung eines automatisierten Verfahrens zur Problemlösung
  - genaue, von Computern ausführbare Beschreibung des Ablaufs von einzelnen, einfachen, Schritten: Algorithmus
  - verbunden mit Organisation von Daten: Datenstrukturen
  - ➢ erfordert "technische Empathie": Vorstellung davon, was ein Computer prinzipiell kann
- Umsetzung in Code, in bestimmter Programmiersprache
  - hier: Java
- Ausführung des Codes auf Rechnern

Rechnerarchitektur / Programmiersprachen

Was kann ein Computer prinzipiell ?

- Von-Neumann-Architektur (Stored Program Architecture)
- Sowohl Daten als auch Programme liegen im Hauptspeicher / MU
  (Sequenz von adressierbaren Speicherzellen)
- Die einzelnen Berechnungen erfolgen in der CPU / ALU
- Programm (Maschinensprache): Folge von elementaren Befehlen
  - Lesen von Datum aus Speicherzellen in ALU
  - Schreiben von Datum aus ALU in Speicherzelle
  - Operationen (z.B. +, *) auf Daten in ALU
  - bedingter Sprung zu bestimmtem Befehl, abhängig von Datum in ALU

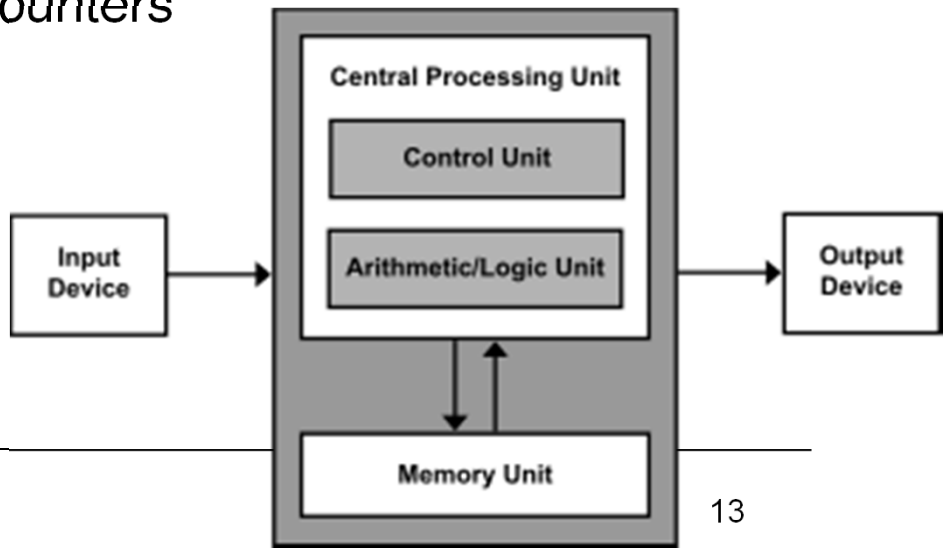


Rechnerarchitektur / Programmiersprachen

- Ausführung des Programms wird von CU gesteuert
- CU verwaltet einen Program counter, der auf den nächsten auszuführenden Befehl zeigt, und steuert den "Befehlszyklus": Was ist zu tun, um den nächsten Befehl abzuarbeiten?
  - Holen und Dekodieren des aktuellen Befehls
  - Laden von Daten aus MU in ALU
  - Ausführen von Operation in ALU
  - Schreiben von Daten aus ALU in MU
  - Aktualisierung des Program counters

!! Akribische Beschreibung von Abläufen in Programmen

!! Computer ist "dumm": führt nur aus, was in sehr kleinteiliger Weise vorgegeben ist



Rechnerarchitektur / Programmiersprachen

- Maschinensprachen als Programmmiersprachen mächtig genug
  - mit Maschinensprachen lässt sich jedes Problem lösen, das überhaupt algorithmisch / maschinell lösbar ist: Berechenbarkeits-Universalität (Turing-Vollständigkeit)
    - verschiedene Formalisierungen von Berechenbarkeit
    - Beweise, dass die alle äquivalent sind
  - es geht nicht um das "was" sondern um das "wie"

➢ Berechenbarkeitstheorie, wird gelehrt im Kontext von "Automaten und Formale Sprachen" oder "Komplexitätstheorie"

Rechnerarchitektur / Programmiersprachen

- Maschinensprachen als Programmiersprachen für die meisten Anwendungen zu primitiv
- Fundamentaler Beitrag der Pionierin Grace Hopper
  - Computer sind für mehr gut als für "number crunching", z.B. für Business Computing
  - Entwickler/innen solcher Anwendungen brauchen komfortablere Sprachen als Maschinencode, um ihre Problemlösungen zu formulieren

➢ "Höhere" Programmiersprachen wie Cobol, Java, C, Python zur Formulierung von Problemlösungen / Algorithmen

➢ Entwicklung von Qualitätskriterien und Praktiken in der Entwicklung wie Dokumentation, Testen, …

➢ Programmier-Paradigmen wie Objekt-Orientierung

➢ Entwicklungsumgebungen

Rechnerarchitektur / Programmiersprachen

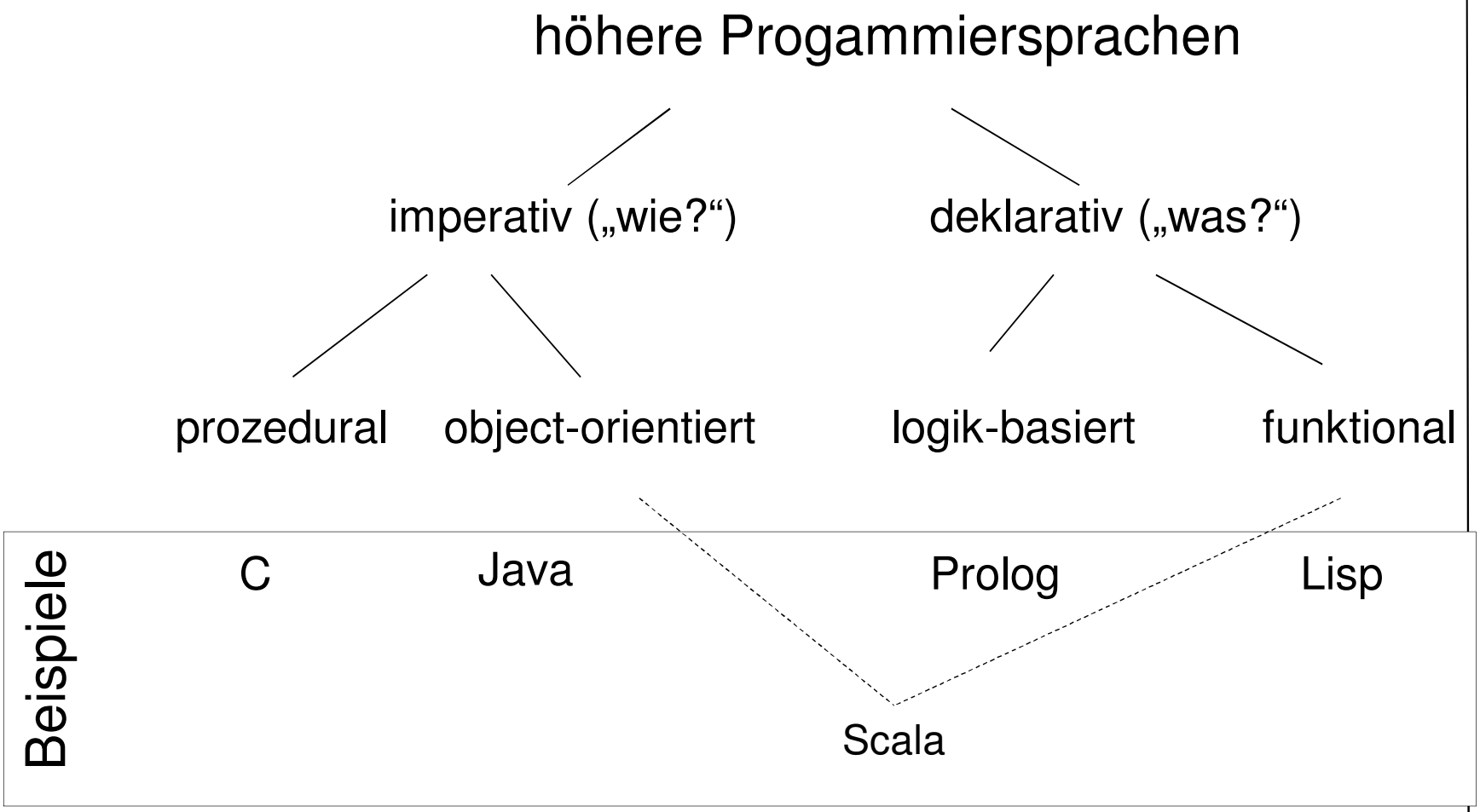
- Umsetzung der Idee "höhere Programmiersprachen" sieht Programme als Sätze in einer künstlichen Sprache
  - strenge Vorgaben zur "Grammatik" dieser Sprachen: Syntax von Programmen (welche Sätze sind gramm. korrekt)
  - genaue Beschreibung des gewünschten Verhaltens von Programmen bei der Ausführung: Semantik
  - spezielle Systeme: Übersetzer / Compiler
    - übersetzen Programm aus höherer Programmiersprache in Maschinensprache ➢ Programm wird vom Compiler in ausführbare Form gebracht
  - alternative Systeme: Interpreter
    - lesen das Programm in höherer Programmiersprache und folgen den Anweisungen des Programms Schritt für Schritt ➢ Programm wird vom Interpreter ausgeführt

Rechnerarchitektur / Programmiersprachen

- "System" Compiler oder Interpreter ist selbst ein Programm
  - in Maschinensprache geschrieben
  - im allgemeinen: selbst in einer einfacheren höheren Programmiersprache geschrieben und in Maschinensprache übersetzt: Bootstrapping
    (sich wie Münchhausen
    am eigenen Schopf aus dem
    Sumpf ziehen)

- ➢ Stack von Programmiersprachen von aufsteigender Komplexität

Programmiersprachen-Paradigmen



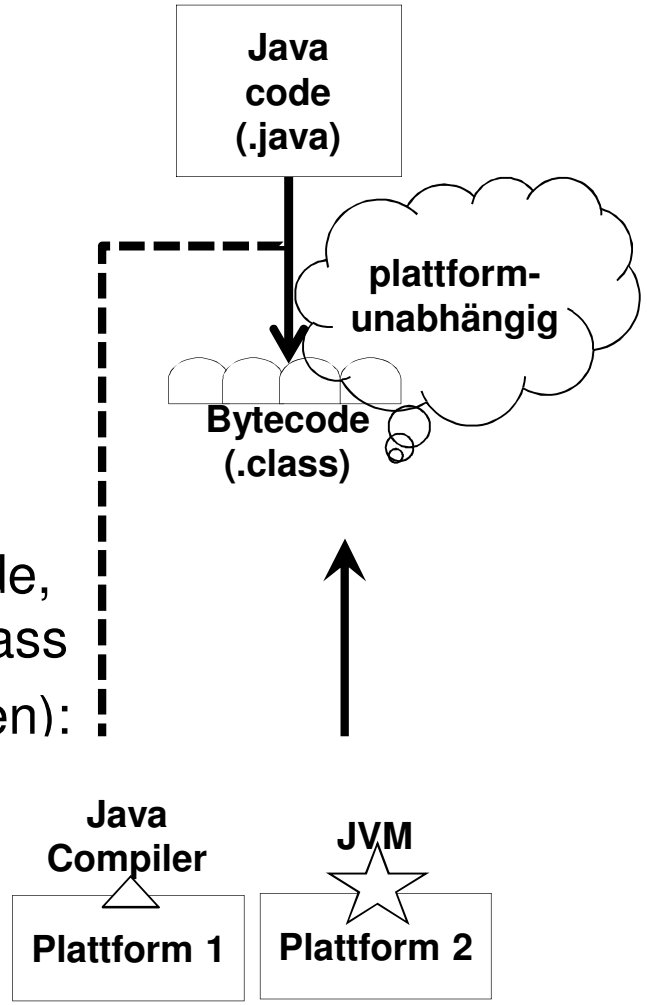
Darstellung nach Georg Groh

Was kommt als nächstes ?

- Konzept EIDI 1 und PGdP
  - … unter dem Aspekt Informatik
  - … unter dem Aspekt Grundlagen Informatik / Programmierung
  - … unter dem Aspekt Praxis (Coding in Java)
- Organisatorisches
- Einordnung:
  Problemlösung, Rechnerarchitektur und Programmiersprachen
- Erste Schritte mit Java
  - das Java-Ökosystem
  - die ersten Java-Programme („Hello World“)
    - Einlesen und Ausgeben von Text
    - die Klasse String
    - Variablen

Das Java-Ökosystem

- „Schreiben“ des Programms mit Texteditor (Codieren): Klasse namens *XXX* in Textdatei Datei *XXX*.java
  - Programm besteht aus Festlegungen (Deklarationen) und Statements, die vom Computer nach bestimmten Ablaufregeln schrittweise ausgeführt werden sollen
- Übersetzen des Programms in einfachere Programmiersprache, in sogenannten Bytecode, mit javac (Compilieren): Ergebnisdatei *XXX*.class
- Ausführen des Bytecode mit java (Interpretieren): Das Programm java startet eine (virtuelle!) Maschine JVM, die den Bytecode in *XXX*.class ausführt, beginnend bei der Methode main() in der Klasse namens *XXX*.



Darstellung nach Idee von Georg Groh

Wird Java-Programm compiliert oder interpretiert ?

- C (compiliert)
  - C-Compiler erzeugt Code in Maschinensprache (Code ist spezifisch für Prozessor/Betriebssystem, plattformabhängig)
  - Betriebssystem führt den Maschinencode aus
- Python (interpretiert)
  - Python-Interpreter liest Python-Code und führt ihn aus
- Java mit Bytecode als Zwischensprache (compiliert-interpretiert)
  - Compiler (javac) erzeugt plattformunabhängigen Bytecode (verbreitete Idee der Zwischensprache)
  - JVM (java) interpretiert den Bytecode und führt ihn aus
    - Bytecode als Maschinencode für eine virtuelle Plattform
    - JVM als Betriebssystem-Programm für diese virtuelle Plattform, das Bytecode interpretiert und ausführt

Das Programm "Hello World"

- Informatik-Kult: das erste Programm in neuer Programmiersprache gibt "Hello World" aus
- Kern dazu in Java

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

  - Statement, abgeschlossen mit Semikolon
  - Aufruf der Methode `System.out.println` mit dem Argument `"Hello World"`, einer Zeichenkette (einem Wert vom Typ String, einer Instanz der Klasse String)
  - ➢ Aspekte von Syntax und Semantik

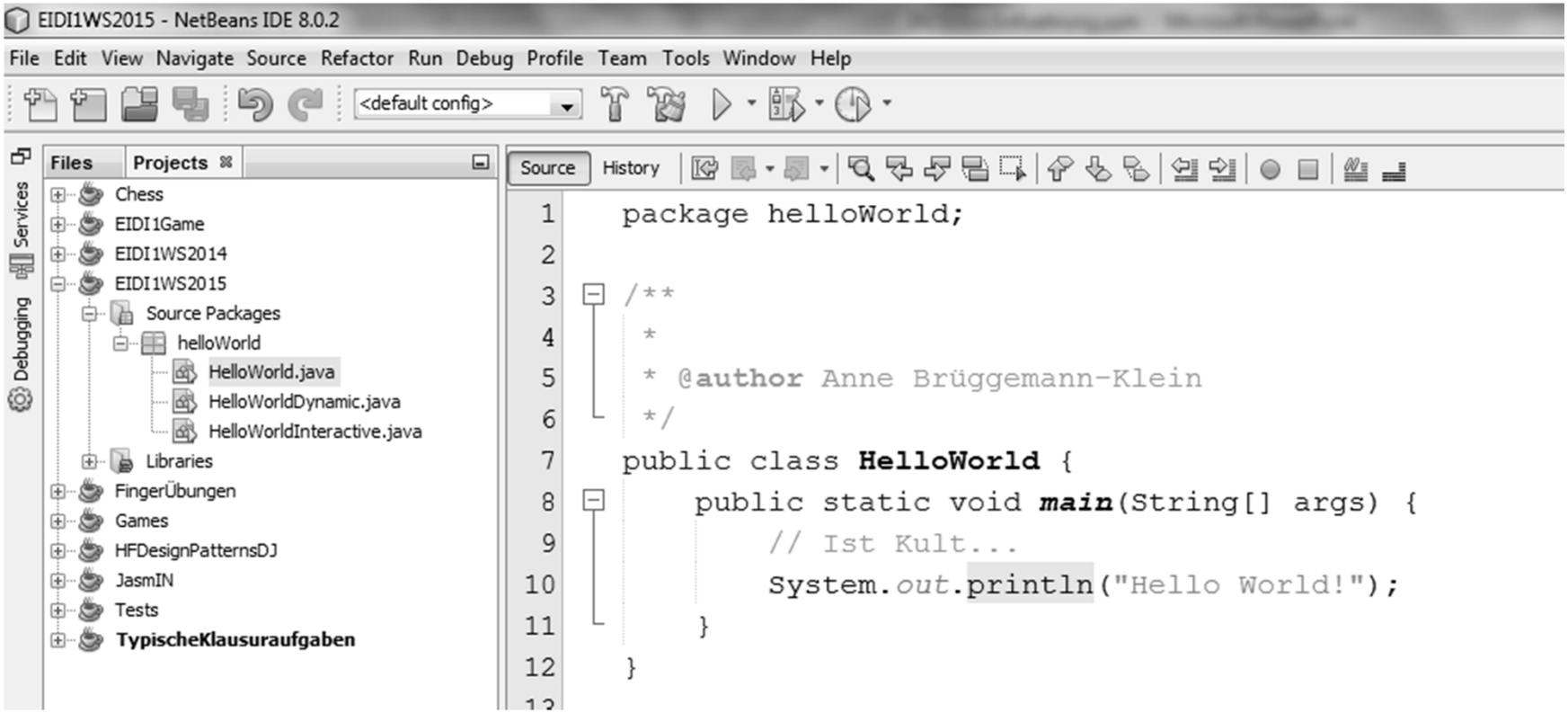
Code-Rahmen in Java

- Template für Klasse (erst mal so verwenden...)

```
package <PackageName>;
/*
 * <Comment to describe purpose of class>
 *
 */
public class <ClassName> {
  public static void main(String[] args) {
    <statement>;
    // more Statements can go here
  }
}
```

- Begriffe: Name, reservierter Name (public, class, void, …), freie Formatierung mit „Whitespace“ (Einrückungen, Umbruch), Kommentare

"Hello World!" live



Entwicklungsumgebung NetBeans mit Werkzeugen
(u.a. Editor, Compiler, Runtime)

Java-Programm/-Code, Klasse mit Klassennamen (HelloWorld)

Datei (HelloWorld.java), Paket (helloWorld), Projekt (EIDI1WS2015)

Demo Werkzeuge

- Compiler und JVM im Kommandofenster: javac und java
  - Achtung: Systemvariable $PATH muss das JDK „kennen“
  - Pakete und Verzeichnisse müssen übereinstimmen
- NetBeans
  - Vorbereitung: Neues Projekt anlegen
    (z.B. EIDI1WS2015 oder ProjektePGdP)
  - in dem Projekt ein neues Paket anlegen für jede zusammengehörige Gruppe von Programmen
    (z.B. helloWorld)
  - in dem Paket eine Klasse *XXX* in Datei *XXX*.java anlegen
    (z.B. Klasse HelloWorld in Datei HelloWorld.java)
  - Editierfenster,
    Kommando Run (impliziert Compilierung/Build),
    Ausgabefenster

Typische Syntaxfehler

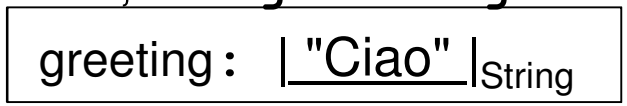
- Vergessene Trennzeichen: " { } ;
- Zeilenumbruch in String-Literalen
- Falsch geschriebene Namen: prntln()
- Fehlerhaftes Template für Java-Klasse, z.B. Vergessen von public static void main(String[] args) {...}
  oder Teilen davon als Container für eigenen Programm-Code
- Keine Übereinstimmung zwischen Dateinamen und Klassennamen: Klasse HalloWorld in Datei HelloWorld.java

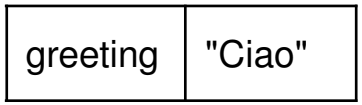
♥ Probieren Sie solche Fehler absichtlich in NetBeans aus

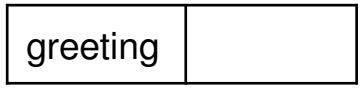
Das Programm "Hello World"

- Sequenz von weiteren Statements mit gleicher Wirkung

```
String greeting;
greeting = "Ciao"; greeting = "Hello";
String toBeGreeted = "World";
String formula;
formula = greeting + " " + toBeGreeted;
formula = formula + "!";
```

  - Variable: benannter Datencontainer, z.B. **`greeting`**
  - Darstellung als benannte Box 

    oder einfach als Tabelle 

  - Deklaration von Variable, z.B. **`String greeting;`**
    - legt Daten-Container an, ohne Wert 

    - legt Typ / Klasse des Containers fest: welche Arten von Werten kann der Container aufnehmen (hier String)
    - nur einmal pro "Scope" (vorerst nur einmal überhaupt)

Das Programm "Hello World"

- Sequenz von weiteren Statements mit gleicher Wirkung

```
String greeting;
greeting = "Ciao"; greeting = "Hello";
String toBeGreeted = "World";
String formula;
formula = greeting + " " + toBeGreeted;
formula = formula + "!";
```

  - Namentabelle für Variablen: Entwicklung während der Ausführung

greeting	

greeting	"Ciao"

greeting	"Hello"

greeting	"Hello"
toBeGreeted	"World"

greeting	"Hello"
toBeGreeted	"World"
formula	

greeting	"Hello"
toBeGreeted	"World"
formula	"Hello World"

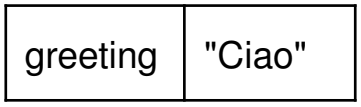
greeting	"Hello"
toBeGreeted	"World"
formula	"Hello World!"

Das Programm "Hello World"

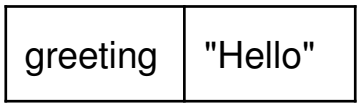
… Sequenz von weiteren Statements mit gleicher Wirkung

```
String greeting;
greeting = "Ciao"; greeting = "Hello";
String toBeGreeted = "World";
String formula;
formula = greeting + " " + toBeGreeted;
formula = formula + "!";
```

– Wertzuweisung an Variable, z.B. **`greeting = "Ciao";`**



- Wert muss zum Typ passen (also hier ein String sein)
- Variable kann nacheinander verschiedene Werte zugewiesen bekommen: im Container greeting wird "Ciao" durch "Hello" ersetzt



- Wert kann als Ausdruck angegeben sein, z.B. als String-Addition in **`greeting + " " + toBeGreeted`**

Das Programm "Hello World"

… Sequenz von weiteren Statements mit gleicher Wirkung

```
String greeting;
greeting = "Ciao"; greeting = "Hello";
String toBeGreeted = "World";
String formula;
formula = greeting + " " + toBeGreeted;
formula = formula + "!";
```

  … Wertzuweisung an Variable, z.B. `greeting = "Ciao";`

    - Regel: erst Ausdruck rechts auswerten, dann den Wert der Variablen zuweisen, z.B. bei `formula = formula + "!";`

  – Kombinierte Deklaration und Wertzuweisung, z.B. `String toBeGreeted = "World";`

Die Klasse String

- Die Klasse String stellt eine Vielzahl von Methoden bereit, um mit String-Objekten zu arbeiten, z.B. **`toUpperCase()`**, **`contains(String)`**
- Übersicht in Java API Documentation http://docs.oracle.com/javase/8/docs/api/
- Aufruf von Methoden auf Objekten mit Punkt-Notation:

```
String toBeGreeted = "World";
toBeGreeted.toUpperCase();
```

- Vorsicht: String-Werte können nicht geändert werden, d.h. **`toBeGreeted.toUpperCase();`** ändert nicht den Wert von toBeGreeted sondern produziert einen neuen Wert, mit dem man etwas machen muss, z.B.:

```
String toBeGreeted;
System.out.println(toBeGreeted.toUpperCase());
```

Einlesen von Strings über die Tastatur

- Googlen führt zur Klasse Scanner
- Genaueres dann in Java API Documentation http://docs.oracle.com/javase/8/docs/api/
- Verwendung in HelloWorldInteractive für typischen Dialog mit User prompting

```
// Declare and initialize Scanner object
// that supports reading of strings from keyboard.
Scanner keyboard = new Scanner(System.in);
// Ask user for greeting.
System.out.print("How do you want to greet?"
  + "   --> ");
greeting = keyboard.nextLine();
```

Was nehmen Sie mit ?

- Dringend NetBeans installieren und für PGdP-Gruppe im Matching-System eintragen
- Zusammenhang EIDI 1 / PGdP: Grundlagen und Praxis
- Java-Ökosystem mit Programm-Code, Compiler, Bytecode, JVM vor dem Hintergrund von Maschinensprache und Von-Neumann-Architektur
  - Plattformunabhängigkeit durch Bytecode
- Einordnen von Java in compilierte / interpretierte Sprachen
  - Prinzip Zwischenformat: Separation of Concerns in Aktion

Was nehmen Sie mit ?

- Aufbau einer Java-Klasse (Einstieg)
  - Rahmen
  - die Klasse String für Zeichenketten (mit Methode)
  - die Klasse Scanner zum Einlesen von Zeichenketten von der Tastatur
- Begriff Algorithmus
  - Dokumentation der Verfahrensweise in Kommentaren

Literatur, Hinweise zum Lernen

- Moodle für EIDI 1 und PGdP:
  https://www.moodle.tum.de/course/view.php?id=22916

- Reges / Stepp: Building Java Programs
- Interaktive Übungen: Practice-it

- MOOC Introduction to Computer Science with Python (edX)
- MOOC Learning how to Learn (Coursera)
- MOOC zu mathematischem Denken (Coursera)

- Aufwand für EIDI 1 und PGdP (Anhaltspunkt)
  - 12 von 30 ETCS (2/5)
  - 2 volle Arbeitstage in einer 5-Tage-Woche