ETH zürich

Java-Einführungskurs Informatik II (D-ITET)

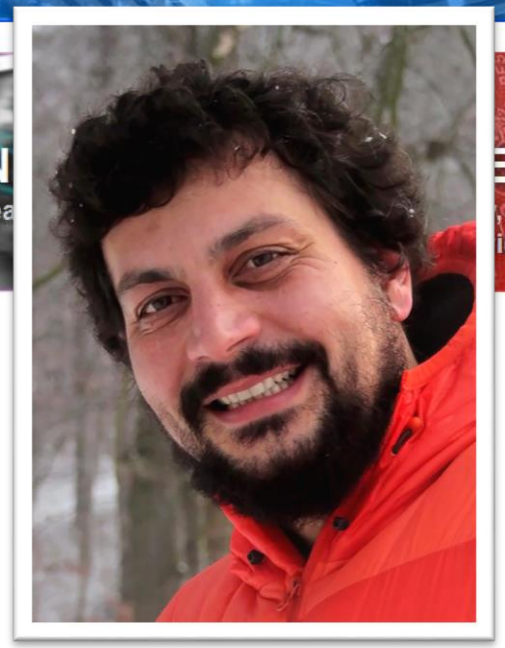
Vincent Becker, *vincent.becker@inf.ethz.ch*

Was haben wir heute vor?

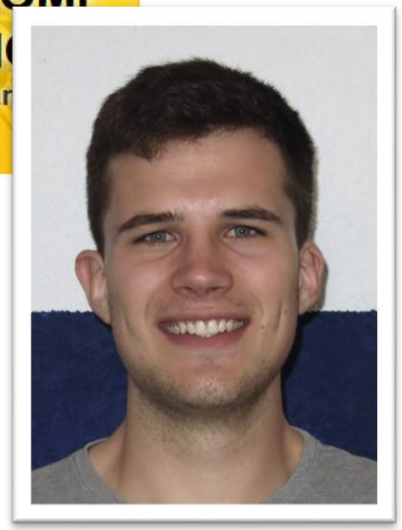
- Vorbereitung auf die Übungen zu Informatik II
  - Vorstellung des Teams
  - Organisatorisches
- Theorie
  - Java-Technologie und Sprache
- Praktisches
  - Erstes Programm auf der Konsole ausführen
  - Eclipse
  - Debugging
  - Testen
  - Javadocs

institute for pervasive computing

RESEARCH GROUP FOR
Distributed Systems



+



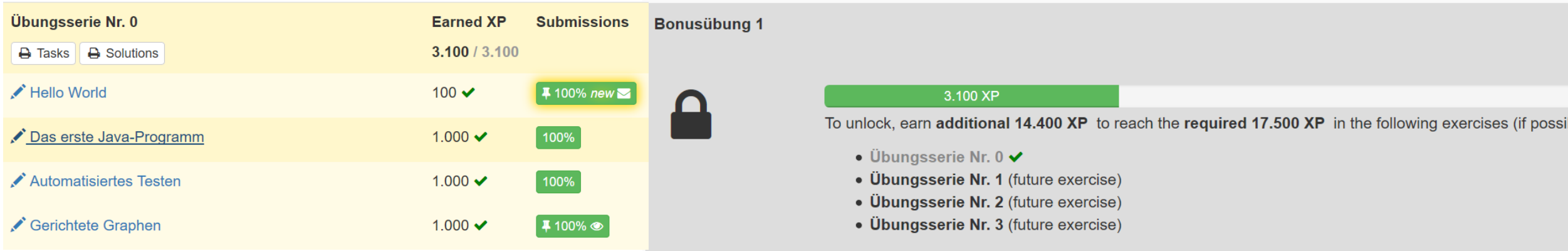
Übungsgruppen

- Mittwochs, 13 – 14 Uhr
- Die Gruppeneinteilung läuft über CodeExpert: https://expert.ethz.ch/enroll/SS19/ifee2
- Anmeldung auf Codeboard.io ist notwendig: https://codeboard.io
- Anwesenheit in den Übungsgruppen ist wichtig
- Ebenso die Bearbeitung der Aufgaben

Fragen & Interaktion!!!

Bonusaufgaben

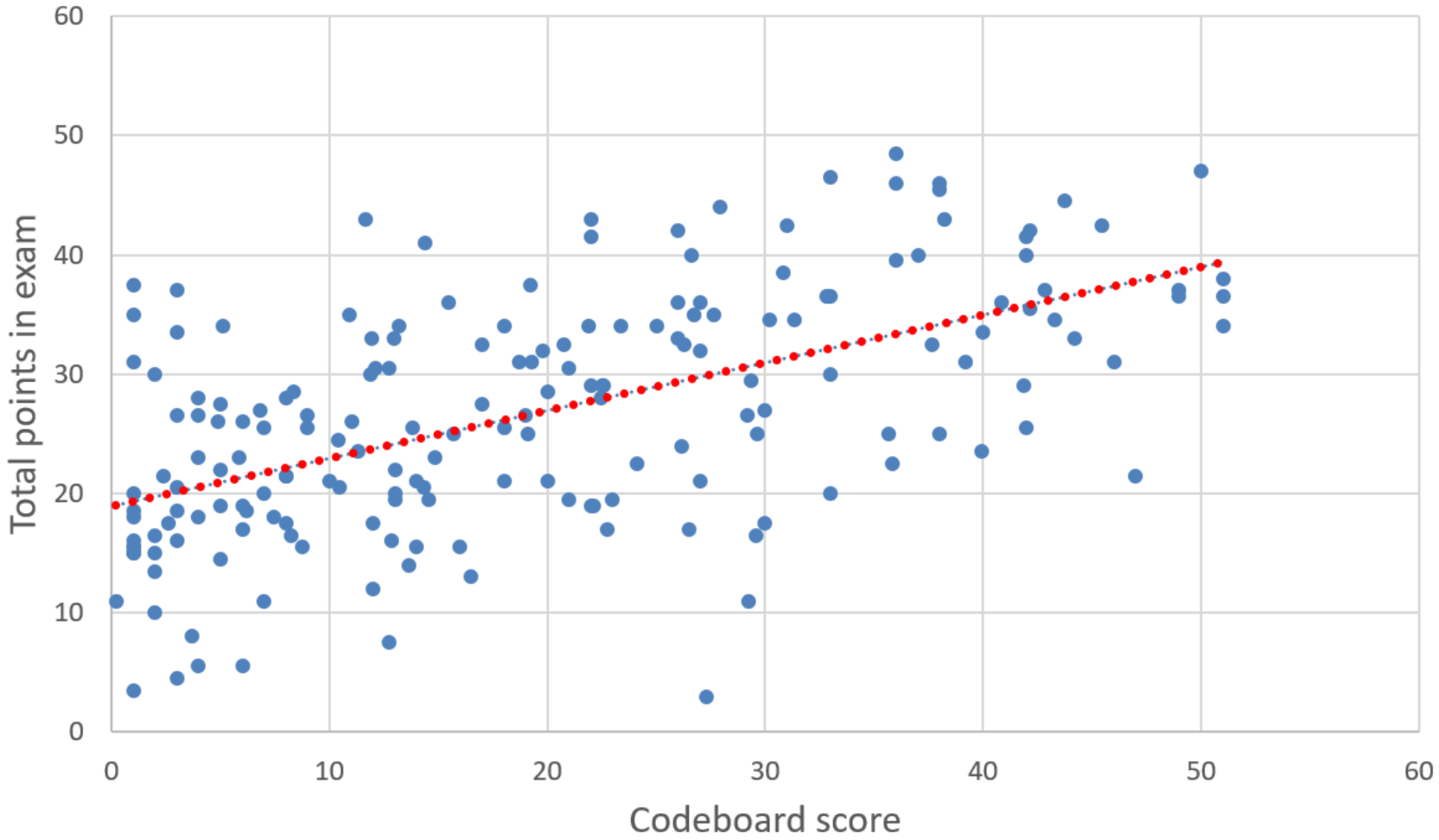
- Es wird wieder spezielle Bonusaufgaben geben (Bonus kann nicht vom letzten Jahr übernommen werden!)
- Um diese freizuschalten, ist das Sammeln von XP in den “normalen” Übungen notwendig



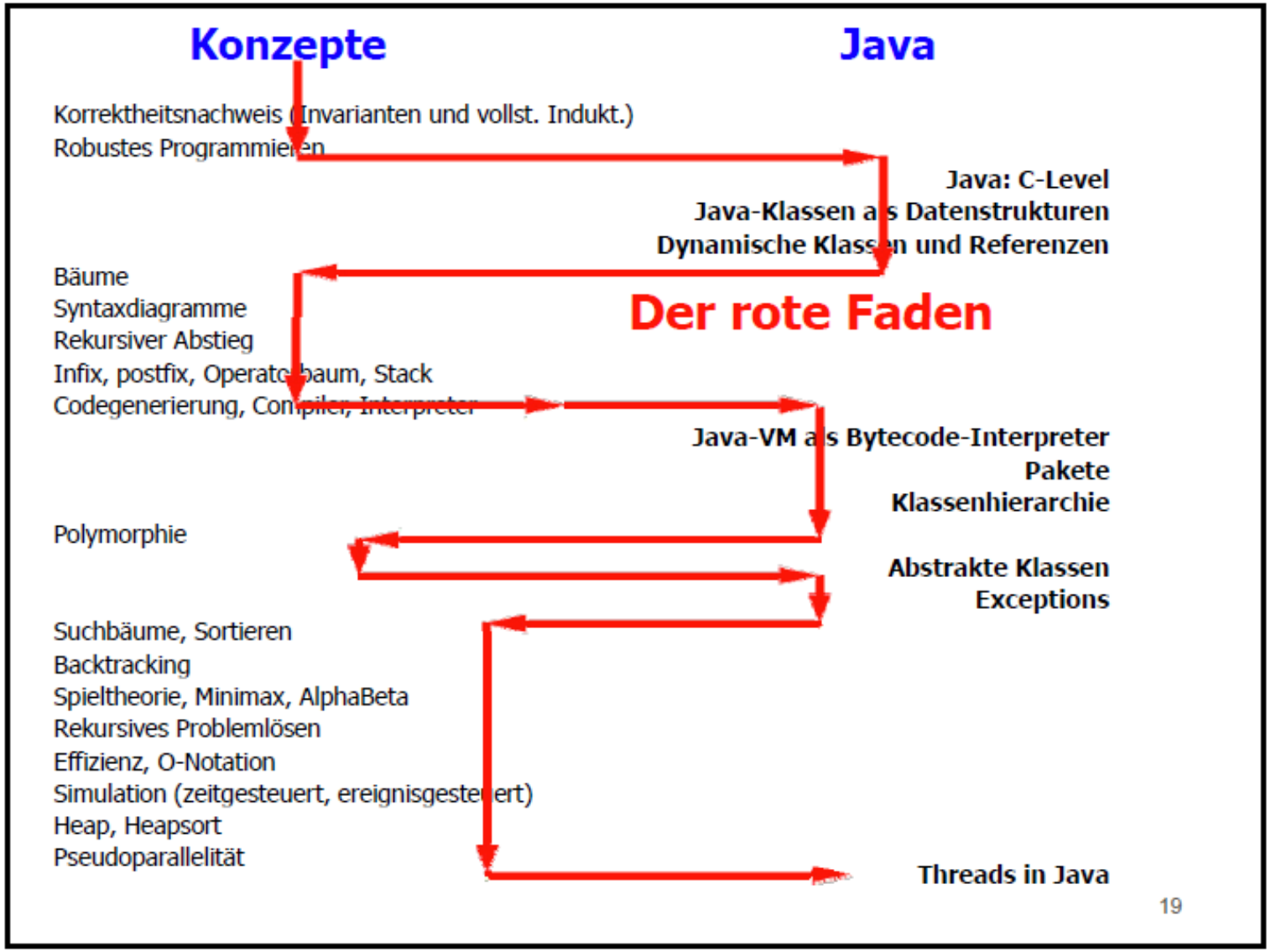
- Die Leistungen aus den Bonusübungen wird proportional in den Bonus von maximal 0.25 Notenpunkten umgerechnet
- Keine Plagiate! Wird automatisch überprüft. Plagiate haben den Verlust des gesamten Bonus aller Beteiligter für das gesamte Semester zur Folge!

Bearbeitung der Übungen führt zu gutem Klausurergebnis?

- Korrelation Codeboard-Score und Punkten in der Klausur: **+0.57**



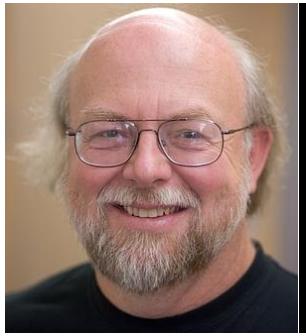
Konzepte sind wichtig!



Which programming language should I use?

(Java ist auch eine Insel)

Warum Java?



"Java is C++ without the guns, knives, and clubs."

- James Gosling

- Objektorientiert
- „Einfacher" als C++
- Umfangreiches Ökosystem: Tools, Bibliotheken, ...
- Virtuelle Maschine: „Compile once – Run everywhere"

Werdegang eines Java-Programms

Program.java — **Quellcode**: Menschenverständlicher Text

↓ `javac Program.java` — Aufruf des **Java-Compilers**

Program.class — **Java-Bytecode**: Maschinenverständlicher Code

↓ `java Program` — Ausführen mittels **virtueller Maschine**

Plattformunabhängigkeit: Java-Bytecode ist ohne Änderung auf jeder Architektur lauffähig, auf welcher eine Laufzeitumgebung installiert ist.




Hello World!

```
/**
 * Ein Programm
 */
public class HelloWorld {

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

}
```

Installation von Java

- Java-Laufzeitumgebung (JRE):
  - Hauptbestandteil ist das Programm `java`
  - Java Virtual Machine (JVM)
  - Standardklassen und weitere Programmbibliotheken

JRE

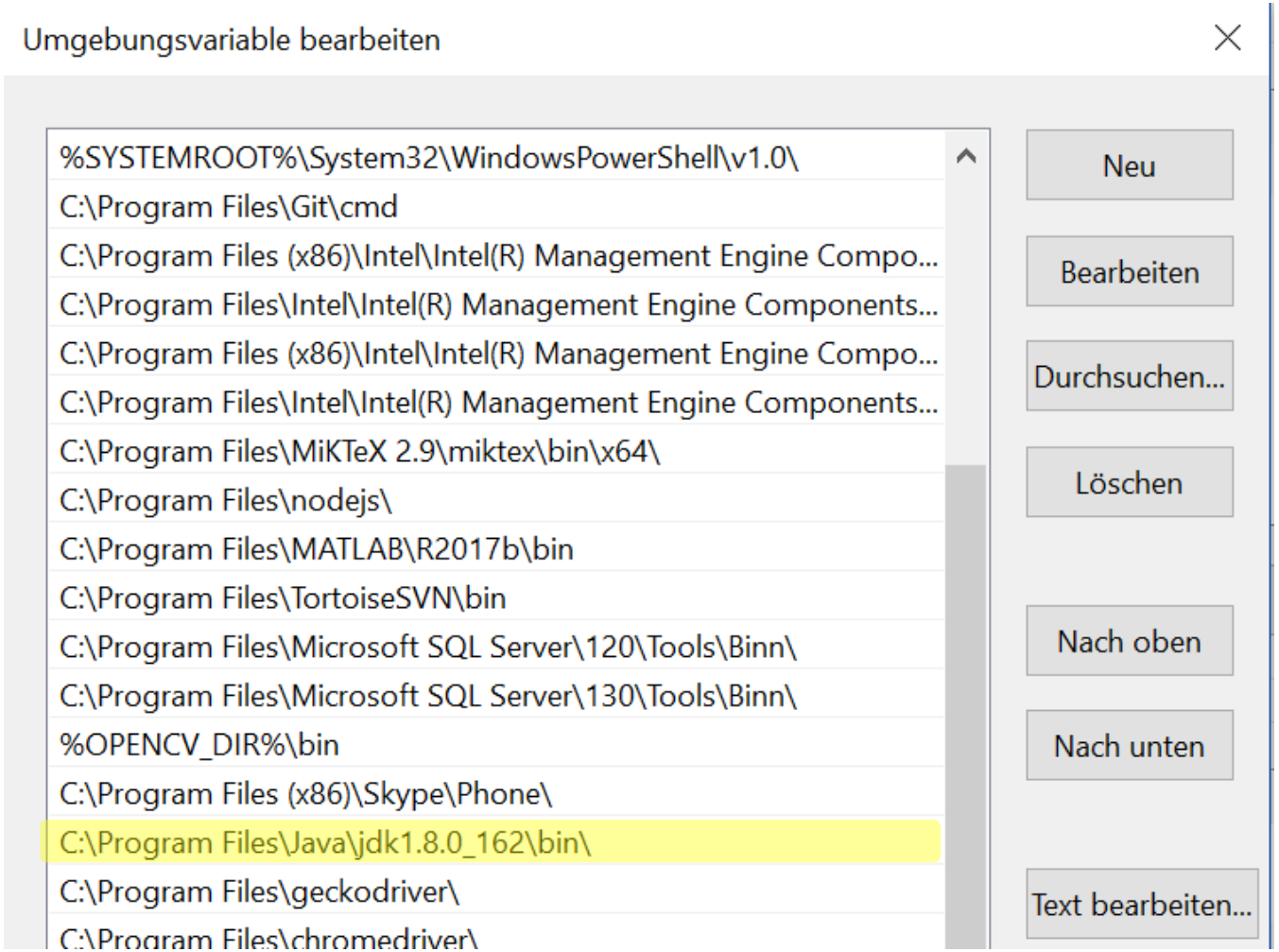
- Java-Entwicklungswerkzeug:
  - Enthält die Programme `java`, `javac` `...`
  - `Enthält die JRE`

JDK

- Wir brauchen das JDK! Wir verwenden Java 8

Hinweis für Windows

- Nach der Installation setzen der Umgebungsvariablen



Aufsführen auf der Konsole

Demo

- Aufruf des Compilers

```
C:\Users\Vincent Becker\Desktop\Java_Intro>javac HelloWorld.java
```

- Ausführen des Programms

```
C:\Users\Vincent Becker\Desktop\Java_Intro>java HelloWorld
Hello World
```

Java-Basics

- Primitive Typen können auf dem Stack angelegt werden, ihre Instanzen sind **keine Objekte**!

```
boolean
byte, char, short, int, long
float, double
```

- Alle anderen Typen sind Objekttypen

Java-Basics: Überall Objekte!

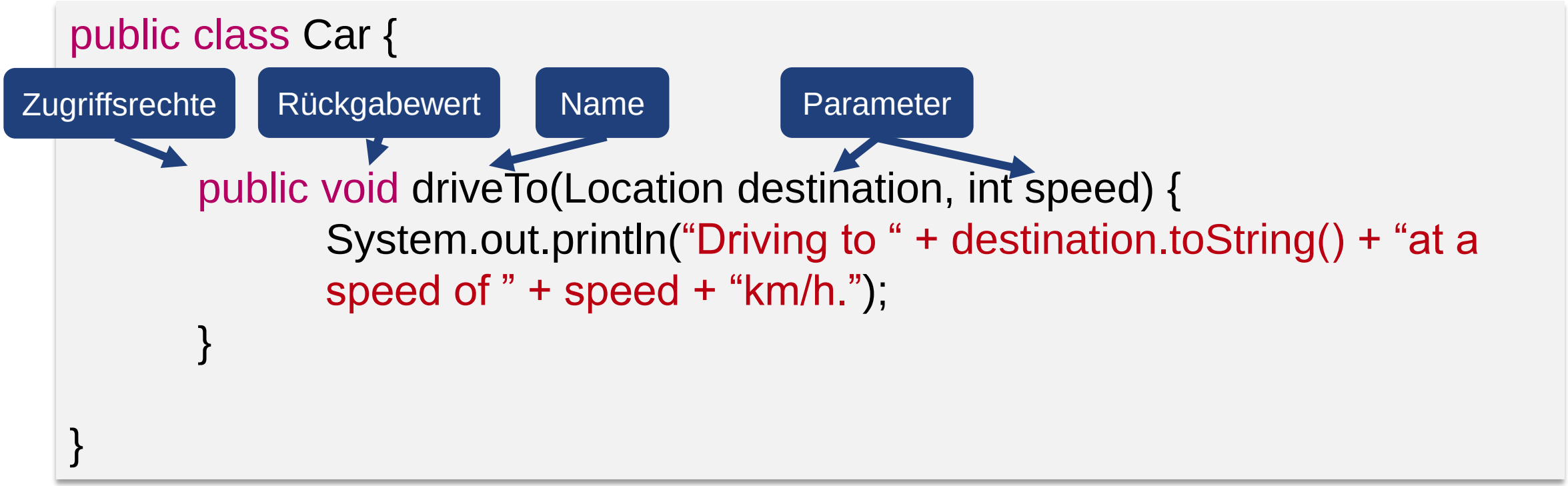
- Objekt: Instanz einer Klasse
- Zugriff ausschliesslich über Referenzen!
- Erzeugung mit `new`

```
Car c = new Car("Porsche", 300);
```

- Entfernung durch Garbage Collector, **kein `delete`!**

Java-Basics: Methoden

- Funktionen heissen in Java Methoden
- Jede Methode hat eine eindeutige Signatur




```
public class Car {

        public void driveTo(Location destination, int speed) {
                System.out.println("Driving to " + destination.toString() + "at a
                speed of " + speed + "km/h.");
        }

}
```

Eclipse (IntelliJIDEA, NetBeans, …)

- Integrierte Entwicklungsumgebungen bieten viele nützliche Tools
  - Direktes Compilieren und Ausführen
  - Syntaxhighlighting
  - Automatische Checks: Syntax, Typen, …
  - Autoformat
  - Debuggen
  - Einfaches Testen
  - Refactoring
  - Anzeige von Javadocs
  - …

- Wir empfehlen die Verwendung von Eclipse, da der Tutor es auf jeden Fall kann
- **WARNUNG:** In der Klausur muss auf Papier programmiert werden!

Installation von Eclipse

- Vorbedingung: JDK installieren
- Eclipse herunterladen: http://www.eclipse.org/downloads/eclipse-packages/
  - Eclipse IDE for Java Developers

- Eclipse ausführen

Ein (bisschen) komplexeres Programm

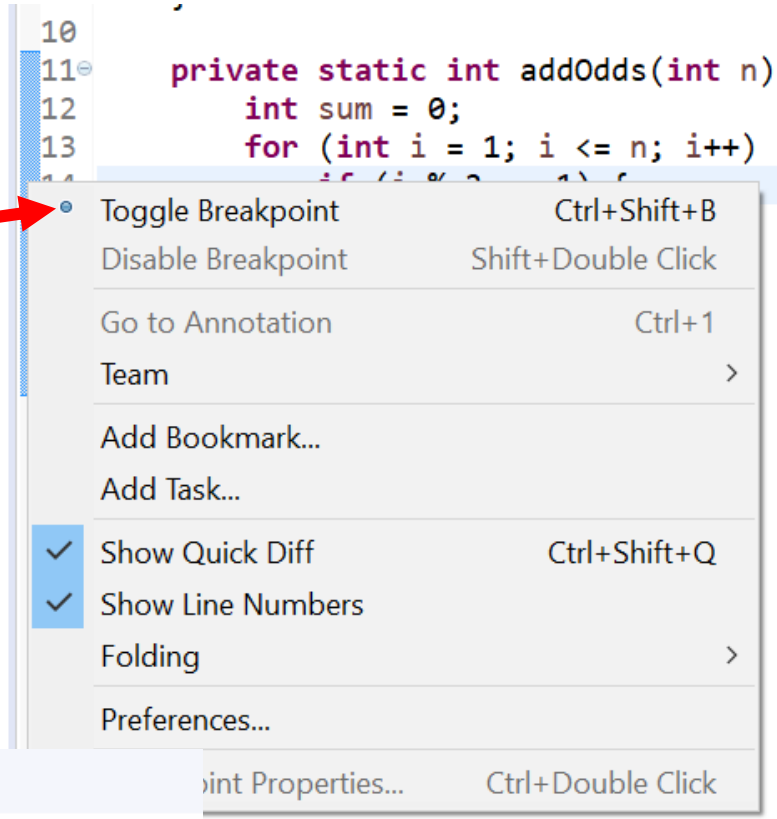
- Aufgabe: Funktion zur Berechnung der Summe aller ungeraden Zahlen bis zur Zahl *n*

  Demo

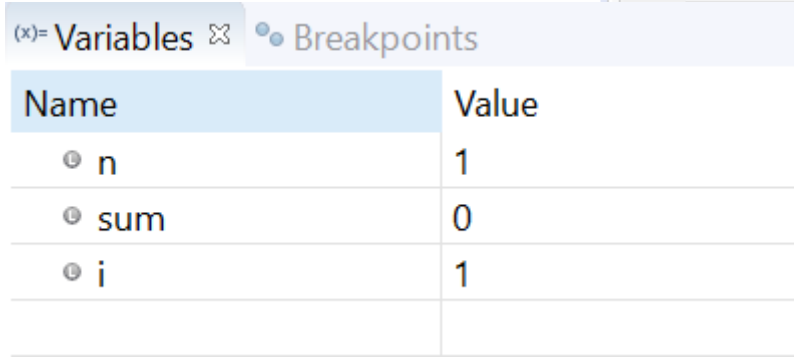
- Achtung: Es gibt viel bessere Lösungen als die naïve, die wir implementieren

Debuggen

- Rechsklick auf den Rand bei der Zeile an der man anhalten möchte
  - Toggle Breakpoint
- Debugger starten
- Step forward / step into (F6 / F7)



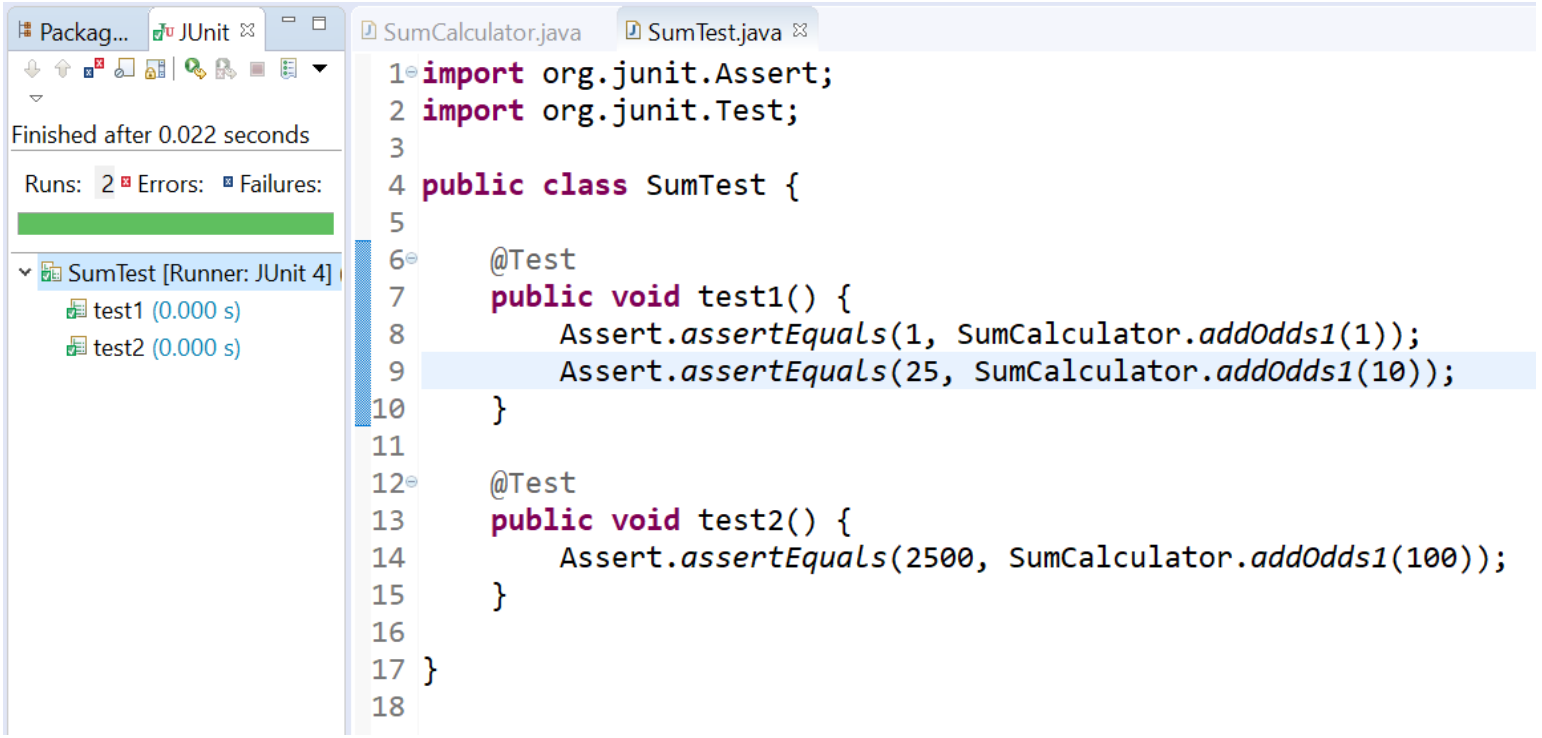

```
15   public static int addOdds1(int n) {
16       int sum = 0;
17       for (int i = 1; i <= n; i++) {
18           if (i % 2 == 1) {
19               sum += i;
20           }
21       }
22       return sum;
23   }
```



Testen

- Junit Tests: Test einzelner Programm-Komponenten
- Einfach durchführbar in Eclipse
  - New -> JUnit Test Case
  - Run

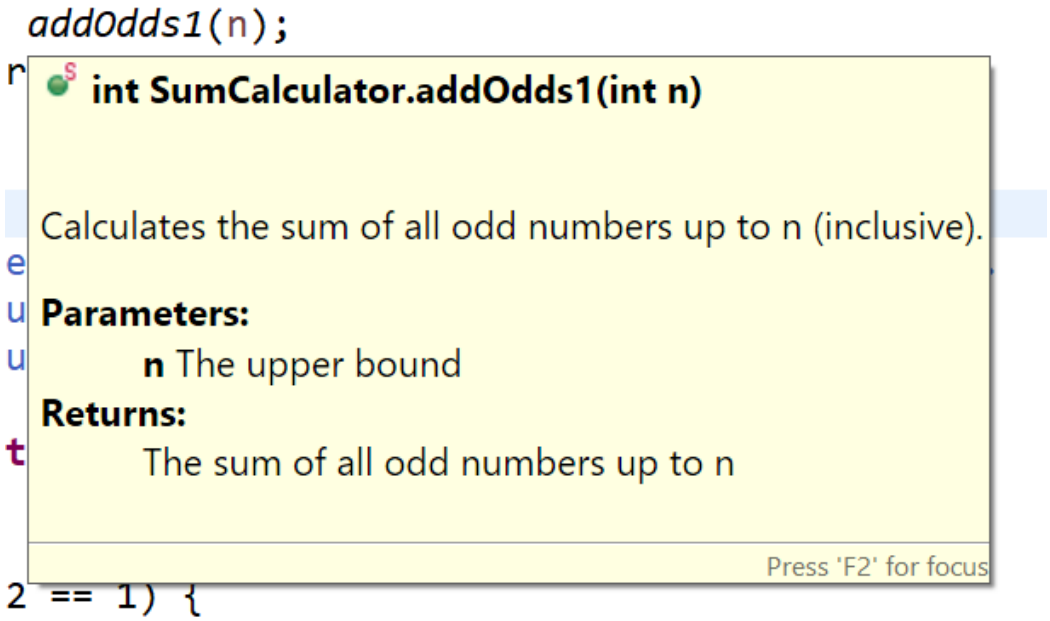
Demo



Javadocs

- Strukturierte Kommentare für Java-Code
- Besonders nützlich in Eclipse

```
/**
 * Calculates the sum of all odd numbers up to n (inclusive).
 *
 * @param n The upper bound.
 * @return The sum of all the odd numbers up to n.
 */
public int addOdds1(int n) {
        …
}
```



Demo

Übungsblatt 0: Aufgabe 1

- `HelloWorld` mit Texteditor
- Ausführen auf der Kommandozeile

- `HelloWorld` in Eclipse
  - Runterladen der Source-Dateien von der Vorlesungswebseite
  - Einbinden in Eclipse

- `HelloWorld` in CodeExpert
  - Generell empfohlen: Bearbeitung in Eclipse, dann zur Abgabe in CodeExpert kopieren

Übungsblatt 0

- Aufgabe 2
  - Erstes Java-Programm: Signum-Funktion
- Aufgabe 3
  - Automatisiertes Testen mit JUnit4
  - Eclipse
  - CodeExpert
- Aufgabe 4
  - Modellbildung

- Fragen?