

Praktische Informatik 2

Spezifikation und Verifikation

Thomas Röfer

Cyber-Physical Systems
Deutsches Forschungszentrum für
Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

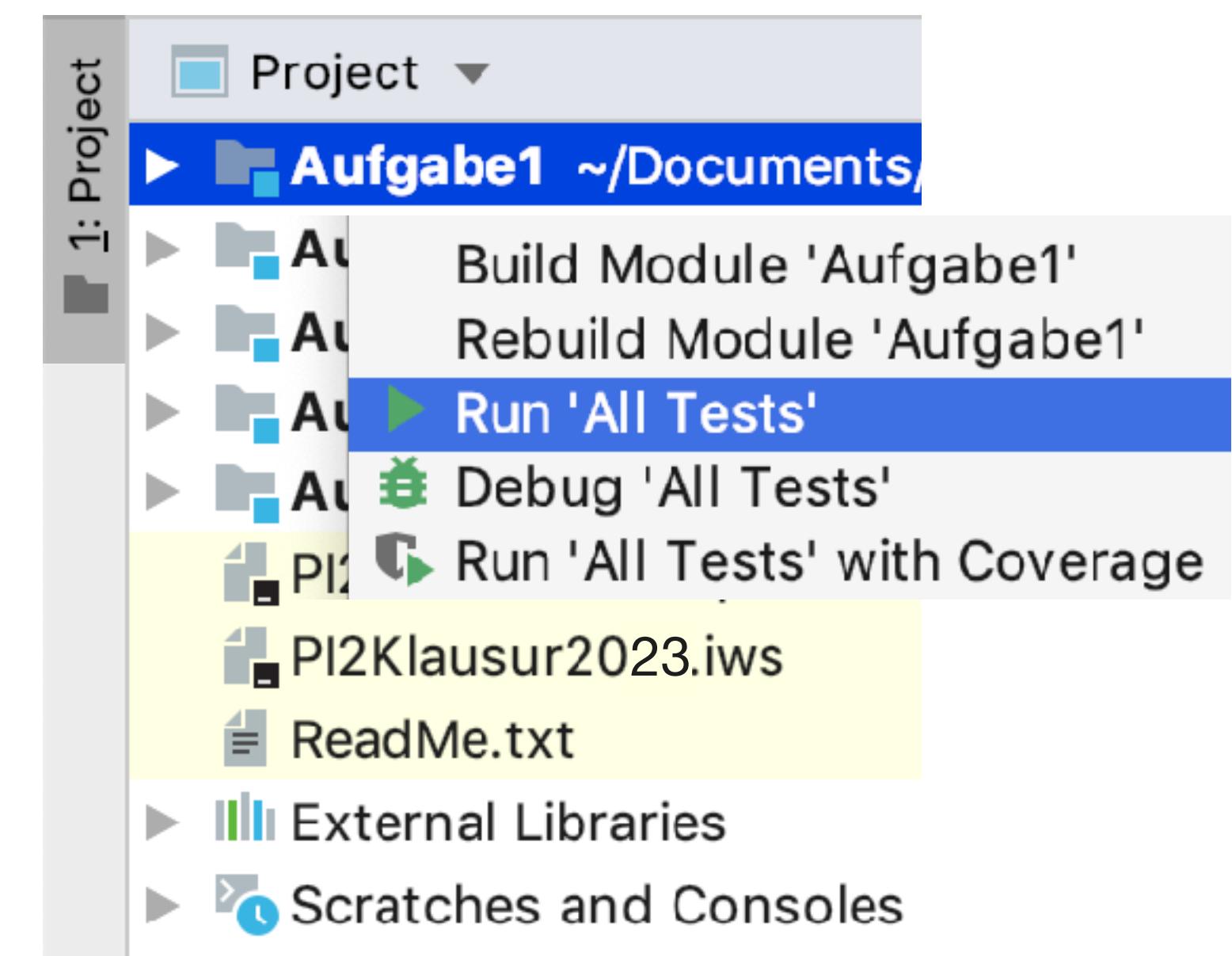
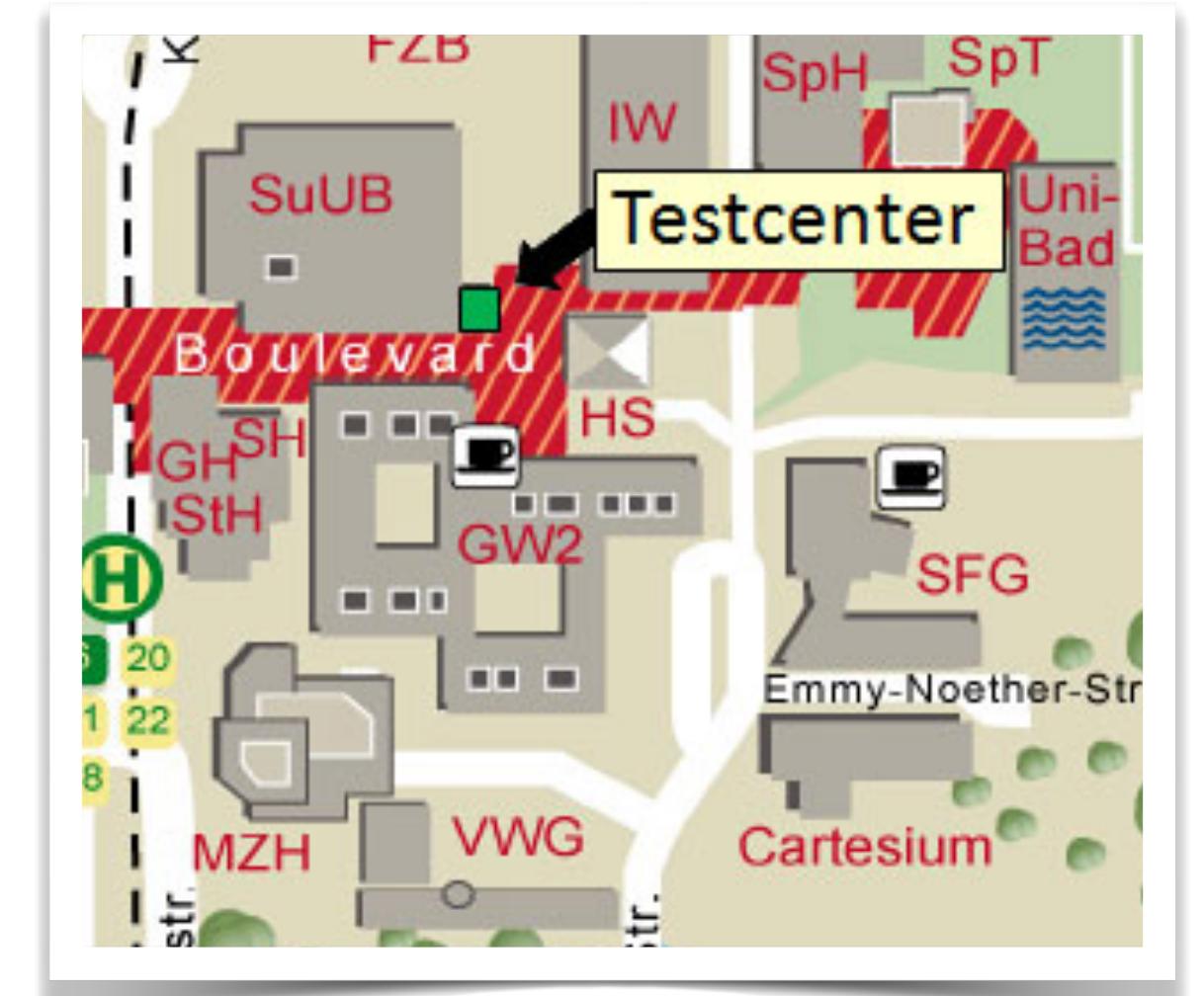


RoboCup 2023 SPL Finale



Klausur: Inhalt

- Klausur im Testcenter am 05.09.2023, Dauer 120 Minuten
- Anmeldung per Terminvergabe in Stud.IP (ihr müsst auch bei PABO angemeldet sein), wird in vorlesungsfreier Zeit freigeschaltet und per E-Mail angekündigt
- Fragenkatalog (20%): Themen quer durch das Semester
- Programmieraufgaben (5 x 20%)
 - Analoge, kürzere Varianten von (Teilen von) Aufgaben, die bereits auf Übungszetteln gestellt wurden
 - Fünf Aufgaben in einem IntelliJ-Projekt als Module
 - Zu jeder Aufgabe existieren bereits einige Tests

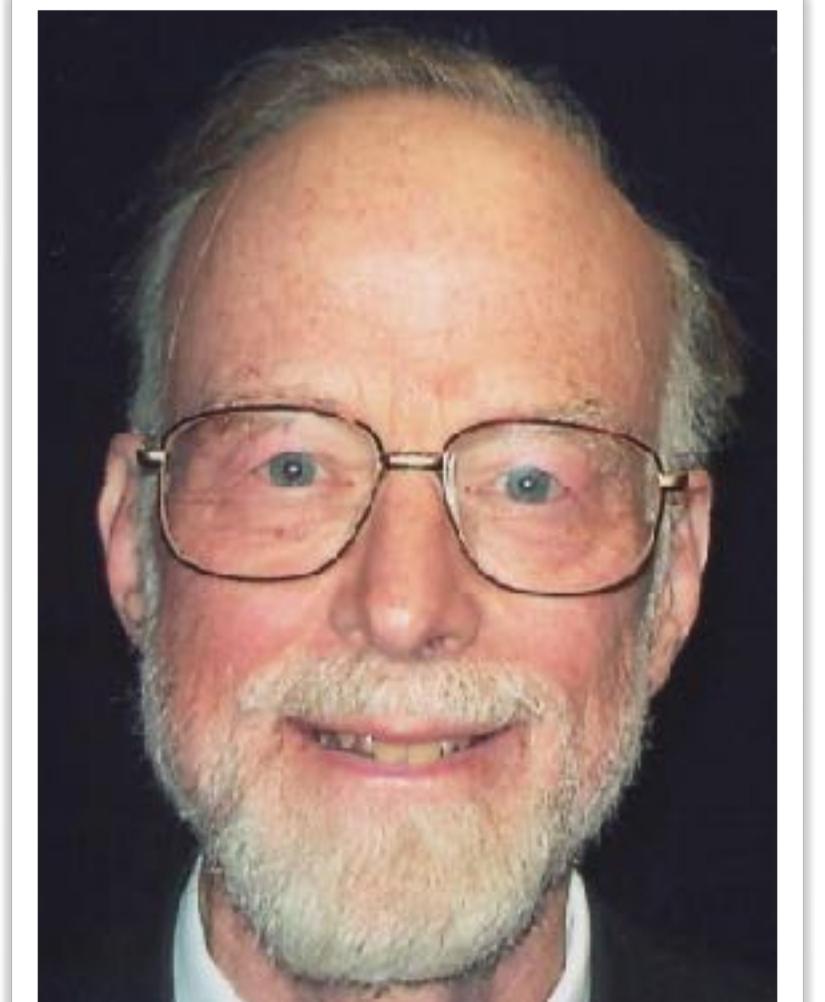


Klausur: Bewertung

- 50% zum Bestehen notwendig
- Programmieraufgaben werden automatisch getestet
- Nur jeweilige Hauptklasse einer Aufgabe wird in vorbereitete Umgebung kopiert, dort kompiliert und getestet
- Übersetzt Quelltext dabei nicht → 0% für diese Aufgabe
- Die Java-Laufzeitbibliothek darf nur verwendet werden, wenn explizit erlaubt (Ausnahme: Testausgaben mit **System.out**. sind ok)
- Testfälle sind andere als in der Klausur

Verifikation mit dem Hoare-Kalkül

- Formales System, um die Korrektheit von Programmen zu beweisen
- Hoare-Tripel: **{P} S {Q}**
 - **S**: Anweisung(en) der Programmiersprache (statement)
 - **P**: Vorbedingung (precondition)
 - **Q**: Nachbedingung (postcondition)
- **P** und **Q** sind logische Aussagen, die Bezeichner des Programms verwenden
- Hoare-Regeln definieren für verschiedene **S**, wie **P** und **Q** zueinander in Beziehung stehen

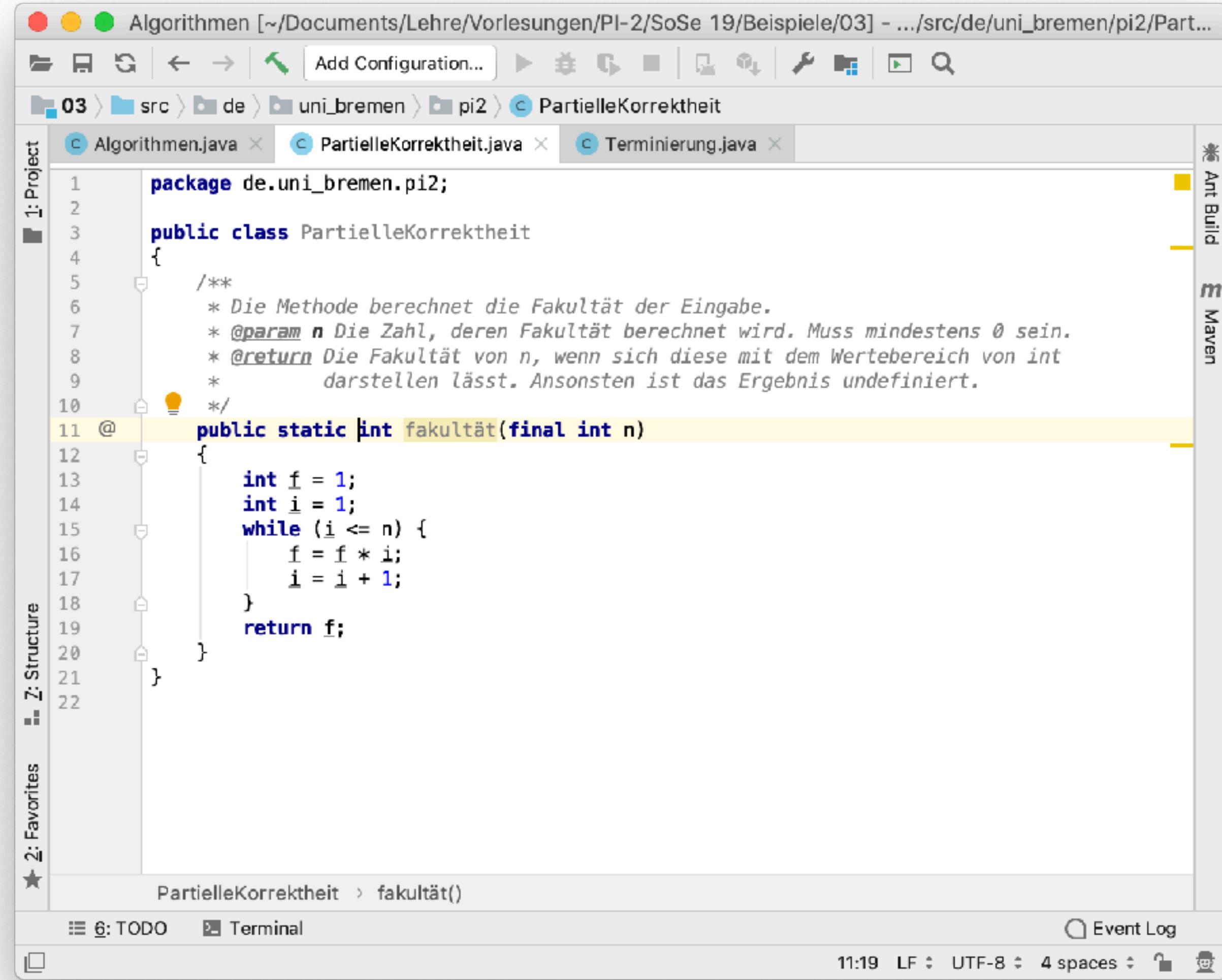


Tony Hoare

Verifikation mit dem Hoare-Kalkül: Grundidee

- Vor- und Nachbedingungen werden auch als Zusicherungen bezeichnet
- Die Vorbedingung der ersten Anweisung enthält üblicherweise Teile der Eingabespezifikation
- Die Nachbedingung der letzte Anweisung sollte die Ausgabespezifikation widerspiegeln
- Ein Beweis ist die Anwendung der Hoare-Regeln von der ersten Vorbedingung zur letzten Nachbedingung
- **Partielle Korrektheit** und **Terminierung** werden getrennt bewiesen

Beweis der partiellen Korrektheit: Demo



The screenshot shows a Java code editor in an IDE. The project structure is visible in the top navigation bar, showing a file tree for '03' > 'src' > 'de' > 'uni_bremen' > 'pi2' > 'PartielleKorrektheit'. The current file is 'PartielleKorrektheit.java'. The code defines a class 'PartielleKorrektheit' with a static method 'fakultät'. The method calculates the factorial of a number 'n'. The code includes Javadoc comments and annotations. The annotation '@param n' is present in the Javadoc comment, and the method signature is annotated with '@' before 'public static int fakultät(final int n)'. The code editor interface includes toolbars, a status bar at the bottom, and various panels for project management and build tools like Ant and Maven.

```
1 package de.uni_bremen.pi2;
2
3 public class PartielleKorrektheit
4 {
5     /**
6      * Die Methode berechnet die Fakultät der Eingabe.
7      * @param n Die Zahl, deren Fakultät berechnet wird. Muss mindestens 0 sein.
8      * @return Die Fakultät von n, wenn sich diese mit dem Wertebereich von int
9      *         darstellen lässt. Ansonsten ist das Ergebnis undefined.
10 */
11 @ public static int fakultät(final int n)
12 {
13     int f = 1;
14     int i = 1;
15     while (i <= n) {
16         f = f * i;
17         i = i + 1;
18     }
19     return f;
20 }
21 }
22
```

Hoare-Kalkül: Axiom der leeren Anweisung

- Generelle Notation: Wenn Ausdrücke über dem „Bruchstrich“ gelten (d.h. bewiesen wurden), dann gilt das Hoare-Tripel unter dem Bruchstrich
- Wenn eine Anweisung keine Variablen verändert, gelten nach der Anweisung dieselben Zusicherungen wie davor
- $\overline{\{P\}S\{P\}}$
- Dies gilt auch, wenn **S** zwar Variablen verändert, diese aber nicht in **P** vorkommen

Hoare-Kalkül: Zuweisungsaxiom

- Nach einer Zuweisung gilt für die geänderte Variable, was vorher für die rechte Seite der Zuweisung galt
- $\overline{\{P[E/x]\}x := E\{P\}}$
- **P[E/x]**: In **P** wurde jedes freie Vorkommen von **x** durch **E** ersetzt
- Beispiel: $\{ i+1 \leq n+1 \} i := i+1 \{ i \leq n+1 \}$

- Diese Regel erfordert Vorbereitung, damit sie überhaupt anwendbar ist
 - Die Vorbedingung muss in eine geeignete Form gebracht werden

Hoare-Kalkül: Kompositions- und Sequenzregel

- Um sequenzielle Programme zu beweisen, muss die Nachbedingung einer Anweisung auch die Vorbedingung der nächsten sein

$$\frac{\{P\}S\{R\}, \{R\}T\{Q\}}{\{P\}S; T\{Q\}}$$

- Beispiel

$$\bullet \{ i+1 \leq n+1 \wedge f \cdot i = (i+1-1)! \}$$

$f := f * i$

$\{ i+1 \leq n+1 \wedge f = (i+1-1)! \}$ Gleichzeitig Vor- und Nachbedingung!

$i := i + 1$

$\{ i \leq n+1 \wedge f = (i-1)! \}$

Hoare-Kalkül: Konsequenzregel

- Vorbedingungen dürfen verstärkt werden, z.B. durch Verschärfung bestehender Aussagen oder Ergänzung zusätzlicher, wahrer Bedingungen
- Nachbedingungen dürfen abgeschwächt werden, z.B. durch Abschwächen bestehender Aussagen und Entfernen von Bedingungen

$$\bullet \frac{P \Rightarrow P', \{P'\}S\{Q'\}, Q' \Rightarrow Q}{\{P\}S\{Q\}}$$

- Beispiel: **{true} x := 0 {x ≥ 0}**, da $\frac{\text{true} \Rightarrow 0 = 0, \{0 = 0\}x := 0\{x = 0\}, x = 0 \Rightarrow x \geq 0}{\{\text{true}\}x := 0\{x \geq 0\}}$
- In Vor- und Nachbedingungen dürfen außerdem beliebige Umformungen vorgenommen werden, solange die Aussagen äquivalent bleiben

Hoare-Kalkül: Auswahlregel

- Für Verzweigungen mit 2 Alternativen muss die Nachbedingung für beide Alternativen gelten

$$\frac{\{P \wedge B\}S\{Q\}, \{P \wedge \neg B\}T\{Q\}}{\{P\}\text{if } B \text{ then } S \text{ else } T\{Q\}}$$

- Zusätzlich zur Vorbedingung gilt zu Beginn des **then**-Zweiges die **if**-Bedingung, zu Beginn des **else**-Zweiges das Gegenteil
- Beispiel: **{0 ≤ x ≤ 3} if x < 3**
then {0 ≤ x ≤ 3 ∧ x < 3} {0 ≤ x < 3} {1 ≤ x+1 < 4} x := x+1 {1 ≤ x < 4} **{0 ≤ x ≤ 3}**
else {0 ≤ x ≤ 3 ∧ ¬(x < 3)} {x = 3} {} {0 = 0} x := 0 {x = 0} **{0 ≤ x ≤ 3}**
- Fehlt der **else**-Zweig, muss sich **{Q}** aus **{P ∧ ¬B}** ergeben (**T** ist leere Anweisung)

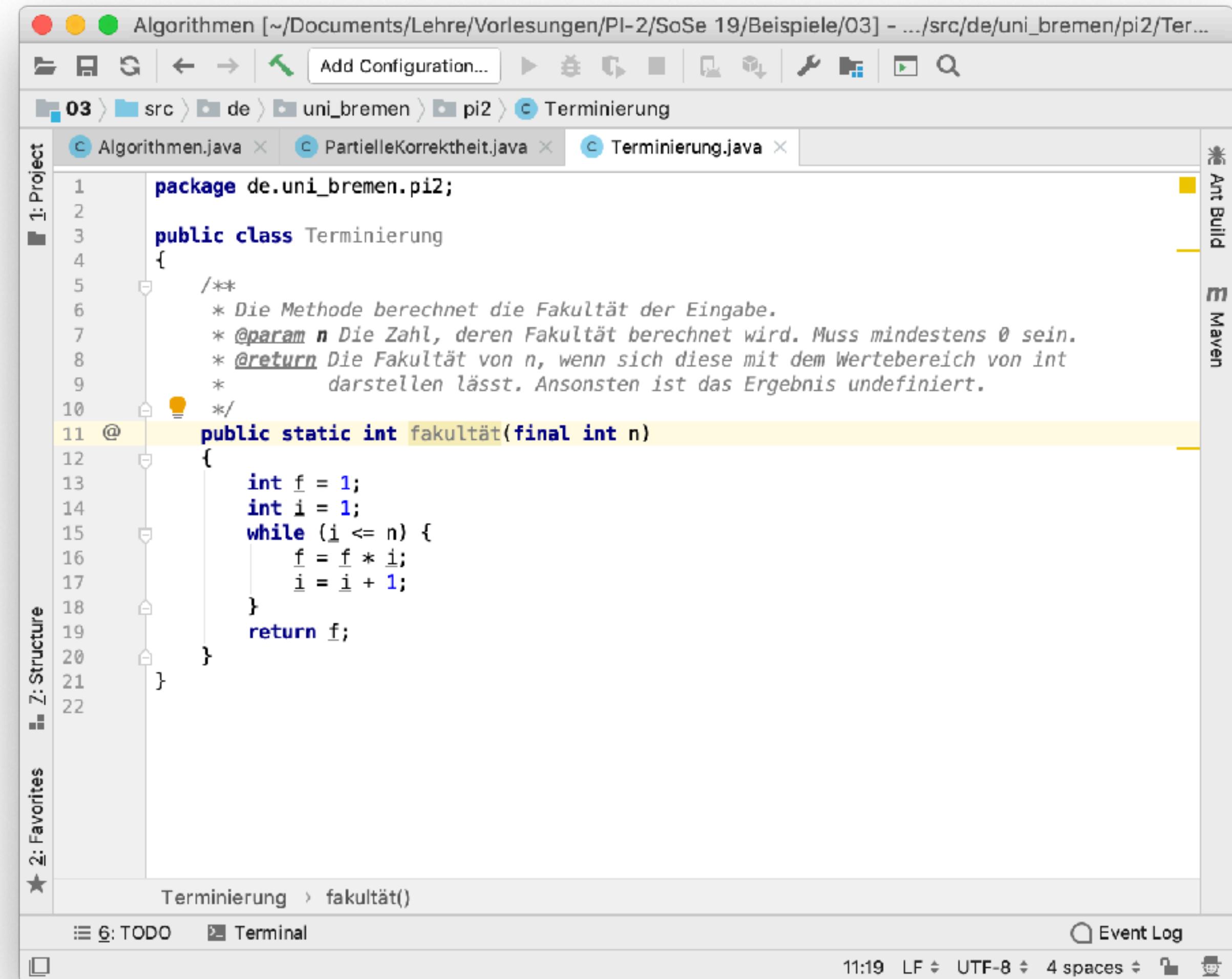
Hoare-Kalkül: Iterationsregel

- Zum Korrektheitsbeweis einer Schleife wird eine **Schleifeninvariante** definiert, die vor der Schleife, am Anfang und Ende des Schleifenrumpfes und nach der Schleife gelten muss

$$\frac{\{I \wedge B\} S \{I\}}{\{I\} \text{while } B \text{ do } S \text{ done} \{I \wedge \neg B\}}$$

- Zu zeigen ist, dass sie am Ende des Rumpfes gilt, wenn sie am Anfang galt
- Die Invariante muss manuell gewählt werden und beschreibt üblicherweise die Grenzen von Laufvariablen und wieviel des Ergebnisses bereits aufgebaut wurde
- Beispiel:
 $\{i \leq n + 1 \wedge f = (i-1)!\}$ while $i <= n$ do
 $\{i \leq n + 1 \wedge f = (i-1)! \wedge i \leq n\} f := f * i; i := i + 1 \{i \leq n + 1 \wedge f = (i-1)!\}$ done
 $\{i \leq n + 1 \wedge f = (i-1)! \wedge \neg(i \leq n)\}$

Beweis der Terminierung: Demo



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

```
1 package de.uni_bremen.pi2;
2
3 public class Terminierung
4 {
5     /**
6      * Die Methode berechnet die Fakultät der Eingabe.
7      * @param n Die Zahl, deren Fakultät berechnet wird. Muss mindestens 0 sein.
8      * @return Die Fakultät von n, wenn sich diese mit dem Wertebereich von int
9      *         darstellen lässt. Ansonsten ist das Ergebnis undefined.
10     */
11    @public static int faktät(final int n)
12    {
13        int f = 1;
14        int i = 1;
15        while (i <= n) {
16            f = f * i;
17            i = i + 1;
18        }
19        return f;
20    }
21
22}
```

The code is annotated with Javadoc-style comments. The method `faktät` is annotated with `@public` (which is likely a typo for `@param`) and `final`. The IDE interface includes toolbars, a file tree, tabs for other files, and various status indicators.

Hoare-Kalkül: Terminierung

- Für den Beweis der Terminierung einer Schleife wird ein Term $t \geq 0$ definiert (**Schleifenvariante**), der die Anzahl der noch zu erwartenden Schleifendurchläufe abschätzt
- Dieser Term muss am Ende des Schleifenrumpfes kleiner als am Anfang sein

$$\frac{\{I \wedge B \wedge t = z\} S \{I \wedge t < z\}, I \Rightarrow t \geq 0}{\{I\} \text{while } B \text{ do } S \text{ done} \{I \wedge \neg B\}}$$

- Um zu zeigen, dass er am Ende des Rumpfes kleiner ist, wird sein Wert zu Beginn des Rumpfes in einer ansonsten nicht verwendeten Variable **z** gespeichert
- Beispiel: **$t = n+1-i$**
 $\{n+1-i = z \wedge i \leq n\} \{n-i < z\} f = f^*i \{n+1-(i+1) < z\} i = i+1 \{n+1-i < z\}$

Zusammenfassung der Konzepte

- **Hoare-Kalkül**
- **Vorbedingung** und **Nachbedingung**
- **Partielle Korrektheit** und **Terminierung**