AuDoscore
Automatic Grading of Java or Scala Homework

Norbert Oster, Marius Kamp, Michael Philippsen
Friedrich-Alexander University Erlangen-Nürnberg

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

DEPARTMENT INFORMATIK
LEHRSTUHL FÜR PROGRAMMIERSYSTEME
(INFORMATIK 2)

Agenda

- Hintergrund und Motivation
- Ablauf der Bewertung mit *AuDoscore*
- Testfall-Erstellung aus Sicht des Dozenten
- Technische Umsetzung
- Erfahrungen mit *AuDoscore*

Hintergrund und Motivation

- Modul „*Algorithmen und Datenstrukturen (AuD)*“
  - 300h (10 ECTS) – ca. 25% davon Java-Hausaufgaben

	WS14/15	SS15	WS15/16	SS16	WS16/17
# Studierende	668	198	716	172	733
# Prog.-Aufg.	37	30	38	32	34
# Java-Abgaben	10.478	1.263	12.526	856	11.189
# Java-Dateien	12.685	1.647	17.274	1.167	13.846
# LOC	1.230.394	94.069	1.528.133	80.859	1.215.662

- Modul „*Parallele und Funktionale Programmierung (PFP)*“
  - 150h (5 ECTS) – ca. 15% bzw. 10% Java-/Scala-Aufgaben
  - bis zu 390 Abgaben (600 Scala-Dateien) pro Semester

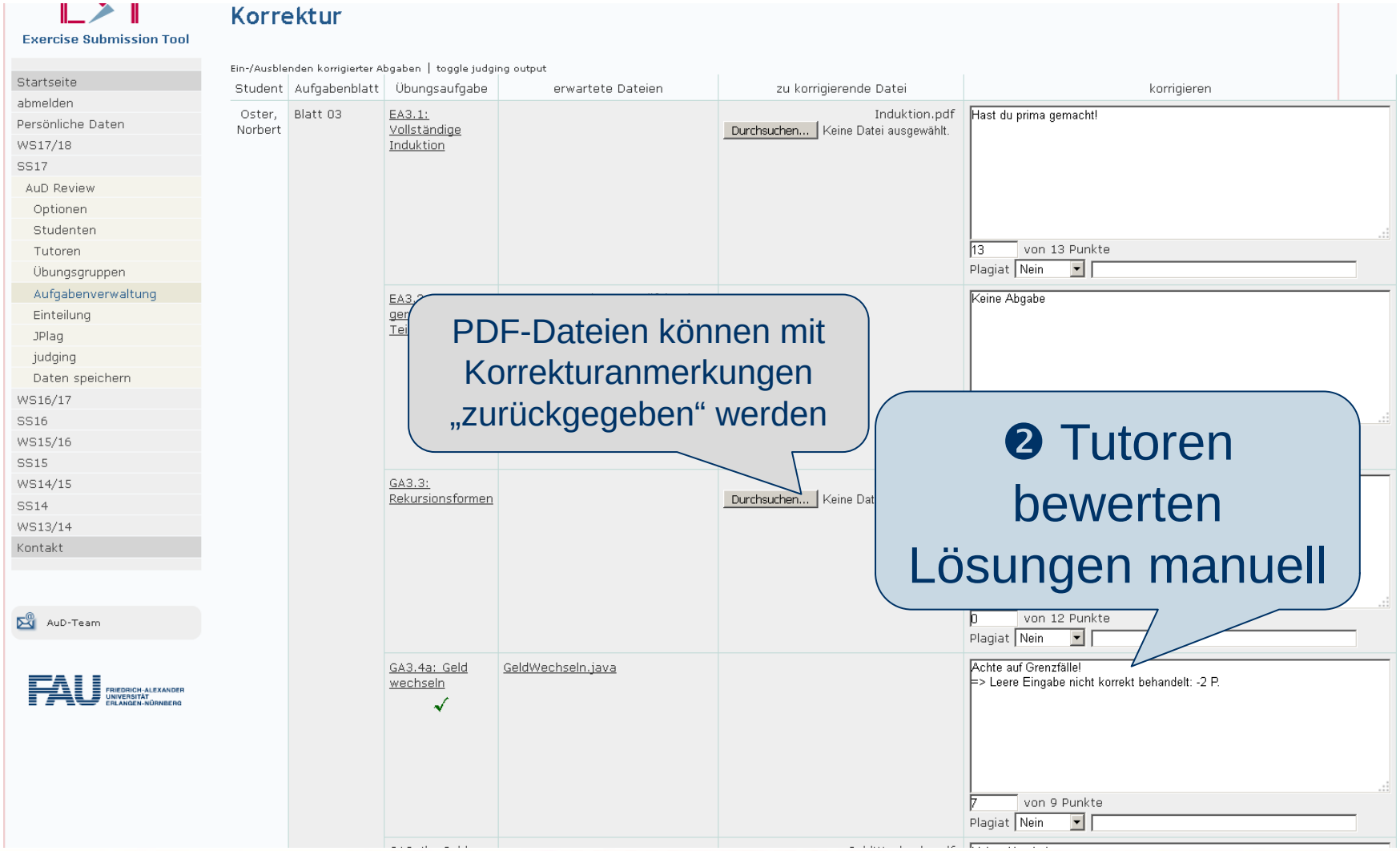
- … und viele weitere, ähnliche Veranstaltungen.

Bewertung von Programmieraufgaben – vor *AuDoscore* (1)

- EST: **E**xercise **S**ubmission **T**ool

Bewertung von Programmieraufgaben – vor *AuDoscore* (2)

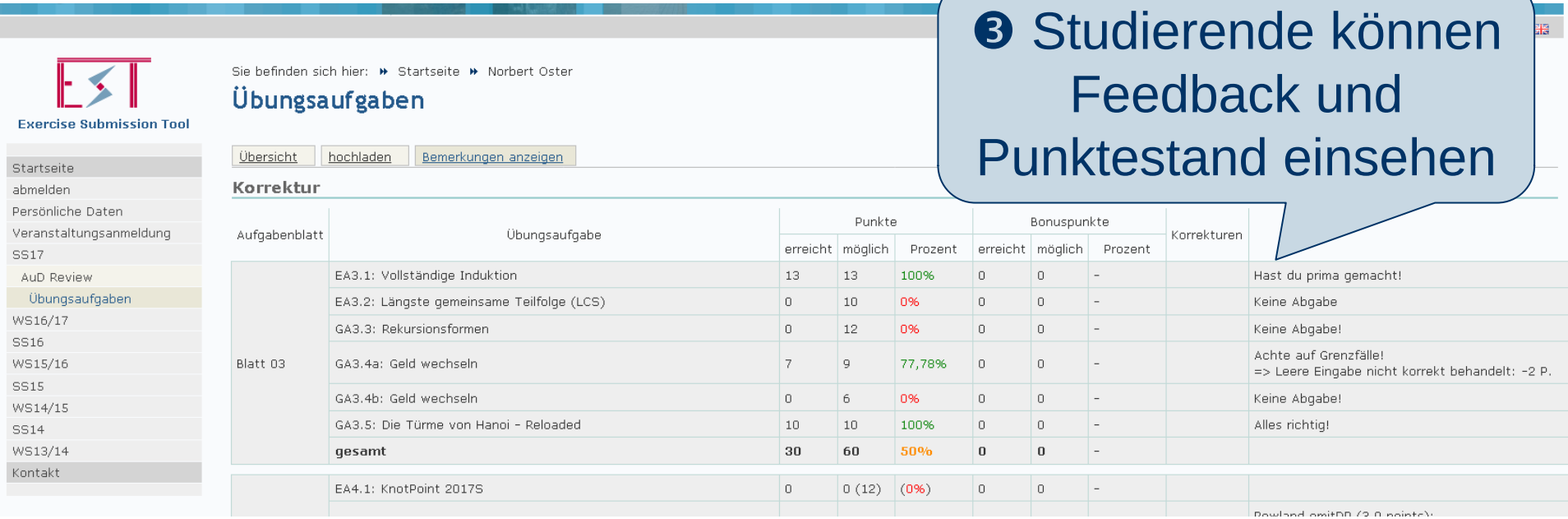
- EST: **E**xercise **S**ubmission **T**ool



Bewertung von Programmieraufgaben – vor *AuDoscore* (1)

- EST: **E**xercise **S**ubmission **T**ool

❸ Studierende können Feedback und Punktestand einsehen



Korrektur

Aufgabenblatt	Übungsaufgabe	Punkte erreicht	Punkte möglich	Punkte Prozent	Bonuspunkte erreicht	Bonuspunkte möglich	Bonuspunkte Prozent	Korrekturen	
Blatt 03	EA3.1: Vollständige Induktion	13	13	100%	0	0	-		Hast du prima gemacht!
	EA3.2: Längste gemeinsame Teilfolge (LCS)	0	10	0%	0	0	-		Keine Abgabe
	GA3.3: Rekursionsformen	0	12	0%	0	0	-		Keine Abgabe!
	GA3.4a: Geld wechseln	7	9	77,78%	0	0	-		Achte auf Grenzfälle! => Leere Eingabe nicht korrekt behandelt: -2 P.
	GA3.4b: Geld wechseln	0	6	0%	0	0	-		Keine Abgabe!
	GA3.5: Die Türme von Hanoi - Reloaded	10	10	100%	0	0	-		Alles richtig!
	gesamt	**30**	**60**	**50%**	**0**	**0**	-		
	EA4.1: KnotPoint 2017S	0	0 (12)	(0%)	0	0	-		

- Vorteile:
  - \+ intuitiv

Nachteile:

- Feedback **erst deutlich nach** Abgabefrist
- manuelle Bewertung von Java-Dateien **schwierig und aufwändig**
- Bewertungen ggf. **inkonsistent** („unfair“)

Hintergrund und Motivation

- möglichst leichtgewichtige Erweiterung von JUnit
  - üblicher/gewohnter Testablauf für Dozenten *und* Studierende (Syntax, IDE, Ausführung, …)
  - *frühes* Feedback an Studierende (während Bearbeitungsfrist!)
  - leicht in EST *integrierbar*, aber auch *unabhängig nutzbar*

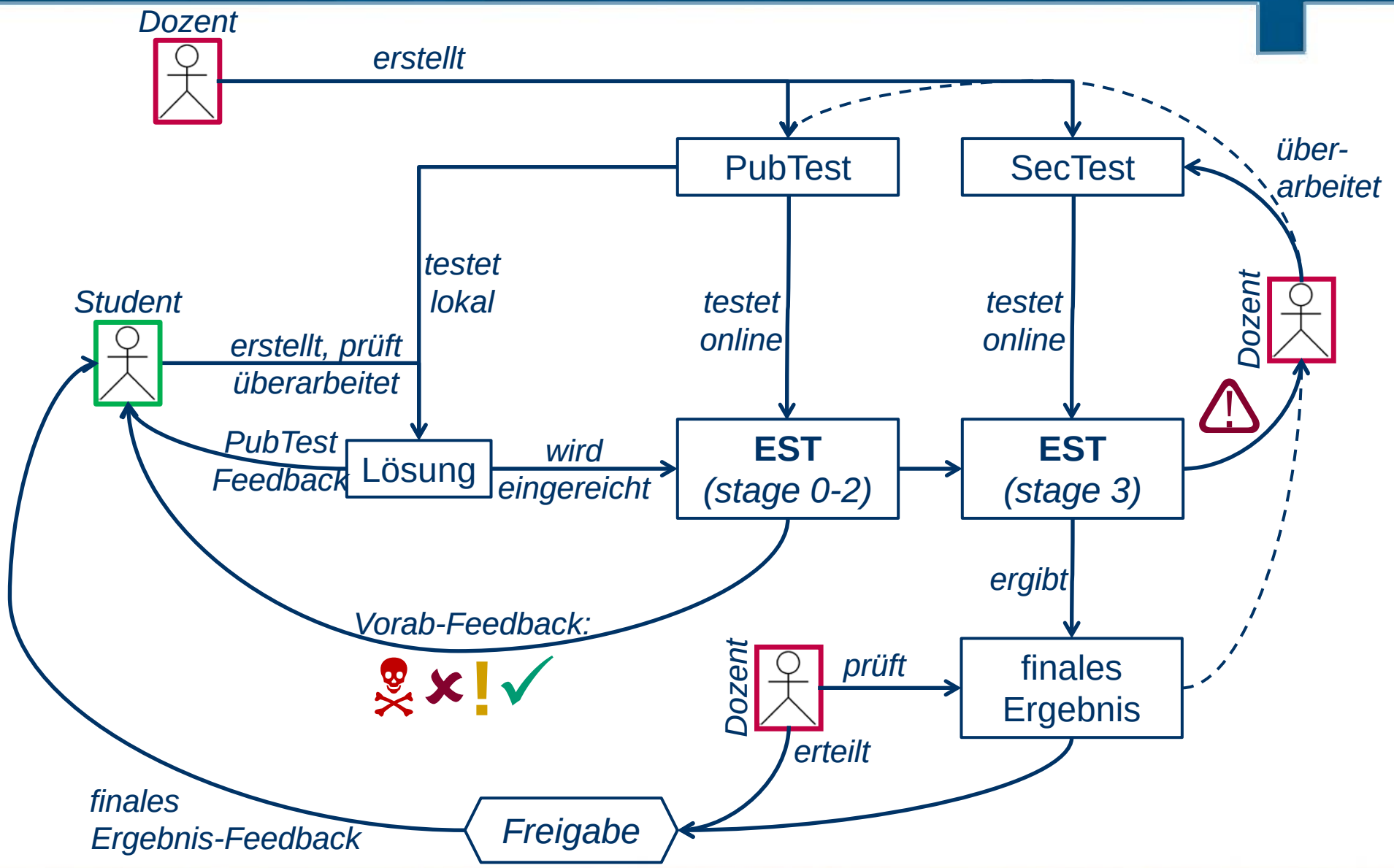
- *AuDoscore*:
  - automatische Bewertung von **Java**-Hausaufgaben
  - https://github.com/FAU-Inf2/AuDoscore (*öffentlich*)
- *ScExFuSS*:
  - Scala Exercise Functional Scoring System
  - nahezu identisch mit AuDoscore – mit **Scala**-Unterstützung
  - https://github.com/FAU-Inf2/ScExFuSS (*noch nicht öffentlich*)

Fusion beider Projekte in Planung.

Agenda

- Hintergrund und Motivation
- **Ablauf der Bewertung mit *AuDoscore***
- Testfall-Erstellung aus Sicht des Dozenten
- Technische Umsetzung
- Erfahrungen mit *AuDoscore*

Ablauf der Bewertung mit *AuDoscore*



Agenda

- Hintergrund und Motivation
- Ablauf der Bewertung mit *AuDoscore*
- **Testfall-Erstellung aus Sicht des Dozenten**
- Technische Umsetzung
- Erfahrungen mit *AuDoscore*

AuDoscore aktivieren

- Dozent erstellt
  - Musterlösung und zwei Testfallklassen (Pub/Sec-Test)
  - AuDoscore erweitert JUnit um zusätzliche Annotationen

- Beide Testfallklassen müssen beide JUnit-Rules enthalten:

```
public class ZeichengeometriePublicTest {
    @Rule
    public final PointsLogger pl = new PointsLogger();
    @ClassRule
    public final static PointsSummary ps = new PointsSummary();
```

Sammelt Informationen zu allen Testfällen und deren Ausführung

Erzeugt Zusammenfassung der finalen Bewertung

Öffentliche und Geheime Testfallklasse (PubTest / SecTest)

- **Öffentlicher** Test deklariert Teilaufgaben mit Wertung:

```
@Exercises({  @Ex(exID = "erzeugeZeichenflaeche", points = 2),
              @Ex(exID = "zeichnePunkt", points = 2),
              @Ex(exID = "zeichneLinie", points = 4),
              @Ex(exID = "zeichnePolygon", points = 3),
              @Ex(exID = "zeichneRechteck", points = 5),
              @Ex(exID = "zeichneKreis", points = 4) })
public class ZeichengeometriePublicTest {
```

Absolute Punktzahl (reale Bewertung)

- **Geheimer** Test erhält nur eine Annotation:

```
@SecretClass
public class ZeichengeometrieSecretTest {
```

Einzelner Testfall (Bonus / Malus)

- Bonus-Test:

```
@Test(timeout = 666)
@Points(exID = "zeichneLinie", bonus = 42)
public void testZeichneLinie__Diagonale__check() {
```

  - ✓: Punkte↑ ✗: keine Auswirkung
  - typischer Einsatz: reguläre Funktionalität

- Malus-Test:

Relative Punktzahl (Testfall-Gewichtung)

```
@Test(timeout = 666)
@Points(exID = "zeichneLinie", malus = 666)
public void test_check_if_invariant__gives_student_cheating() {
```

  - ✓: keine Auswirkung ✗: Punkte↓
  - typischer Einsatz: Rekursion, Schnittstellen, Betrugsversuch

Finale Bewertung

- **`bonus`- / `malus`-** Angaben sind relative Gewichte
  - Vorteil: Ergänzen/Löschen von Testfällen einfach
  - Finale Bewertung für einzelne Teilaufgabe τ gemäß:
  - $$grade_\tau = \left\lfloor \max\left(0, \frac{\sum_{TC_\tau}^{passed} |bonus| - \sum_{TC_\tau}^{failed} |malus|}{\sum_{TC_\tau} |bonus|}\right) \cdot |Ex_\tau.points| \right\rfloor_{(0.5)}$$

- Beispiel:

τ $Ex_\tau.points$

```
@Exercises({…, @Ex(exID = "zeichneLinie", points = 11), …})
```

```
@Points(exID = "zeichneLinie", bonus = 4)
@Points(exID = "zeichneLinie", bonus = 8)
@Points(exID = "zeichneLinie", bonus = 10)
@Points(exID = "zeichneLinie", bonus = 12)

@Points(exID = "zeichneLinie", malus = 6)
```

$$\left\lfloor \max\left(0, \frac{(8+10+12)-(6)}{(4+8+10+12)}\right) \cdot 11 \right\rfloor_{(0.5)}$$

$$\Rightarrow 7.5\ Punkte$$

Finale Bewertung – Ansicht für Studierende

Erreichte Punkte (▒▒▒▒▒▒▒▒▒▒)

Aufgabenblatt	Übungsaufgabe	Punkte			Bonuspunkte			Korrekturen	Kommentar
		erreicht	möglich	Prozent	erreicht	möglich	Prozent		
Blatt 07	EA7.1: Zahlen erraten	13,5	18	75%	0	0	-		Zahlenraten - ADAPTER (9.5 points): ✗ 0.00 | test_rate_Spiel_schon_beendet | AssertionError(Der Rueckgabewert fuer rate ist falsch. expected:<3> but was:<1>) ✗ 0.00 | test_rate_ausserhalb_des_Bereichs | NumberFormatException(): ZahlenRatenAdapter.rate(line 63) ✗ 0.00 | test_rate_zu_gross | AssertionError(Der Rueckgabewert fuer rate ist falsch. expected:<8> but was:<6>) ✗ 0.00 | test_rate_zu_klein | AssertionError(Der Rueckgabewert fuer rate ist falsch. expected:<7> but was:<6>) ✓ 0.52 | test_etwasFunktionalitaet__PUBLIC_TEST ✓ 0.52 | test_rate_spackt_ewig__PUBLIC_TEST ✓ 1.04 | test_rate_spackt_genau_1_mal ✓ 1.04 | test_rate_spackt_genau_6_mal ✓ 1.04 | test_rate_spackt_genau_7_mal ✓ 1.04 | test_rate_uninitialisiertes_Spiel ✓ 1.04 | test_starteNeuesSpiel_Wertebereich_ungueltig ✓ 0.52 | test_starteNeuesSpiel_spackt_ewig__PUBLIC_TEST ✓ 1.04 | test_starteNeuesSpiel_spackt_genau_1_mal ✓ 1.04 | test_starteNeuesSpiel_spackt_genau_6_mal ✓ 1.04 | test_starteNeuesSpiel_spackt_genau_7_mal Zahlenraten - API (4.0 points): ✓ 1.00 | test_Adapter_intestines__PUBLIC_TEST__THIS_TEST_IS_VERY_IMPORTANT_IF_IT_FAILS_THEN_YOU_WILL_GET_NO_POINTS_AT_ALL ✓ 2.00 | test_Throwables_intestines__PUBLIC_TEST__THIS_TEST_IS_VERY_IMPORTANT_IF_IT_FAILS_THEN_YOU_WILL_GET_NO_POINTS_AT_ALL ✓ 1.00 | test_special_Throwable_intestines__PUBLIC_TEST
	EA7.2: Branch Coverage	12	13	92,31%	0	0	-		BranchCoverage-Test (12.0 points): ✗ 0.00 | test__BrachCoverage_of__reset__1_3 | AssertionError(Kante nicht ueberdeckt: [1@PriorityQueue.reset][3@PriorityQueue.reset]) ✓ 1.05 | test__BrachCoverage_of__addItem__10_11 ✓ 1.41 | test__BrachCoverage_of__addItem__12_13 ✓ 1.41 | test__BrachCoverage_of__addItem__12_14 ✓ 1.41 | test__BrachCoverage_of__addItem__2_3 ✓ 1.05 | test__BrachCoverage_of__addItem__5_6 ✓ 1.05 | test__BrachCoverage_of__addItem__5_7 ✓ 0.70 | test__BrachCoverage_of__length__2_3__2_4 ✓ 0.70 | test__BrachCoverage_of__reset__1_2 ✓ 0.70 | test__BrachCoverage_of__toString__2_3__2_4 ✓ 0.70 | test__BrachCoverage_of__top__2_4 ✓ 0.35 | test__BrachCoverage_of__top__PUBLIC_TEST ✓ 1.05 | test__FAKE_RESULTS__to_trigger_Assert_failure ✓ 0.70 | test__use_of_Assert__PUBLIC_TEST
	gesamt	**25,5**	**31**	**82,26%**	**0**	**0**	**-**		

Name der Testmethode oder wahlweise expliziter Kommentar

Beschränkungen der API

- Szenario:
  - Algorithmen oder Datenstrukturen selbst implementieren – auch wenn schon in der API ähnlich vorhanden

- Beispiel: eigene Streutabelle

Verhindert, dass Studierende **`java.util.HashMap`** nutzen

```
@Forbidden({ "java.util.", "java.lang.Math" })
@NotForbidden({ "java.lang.Math.pow" })
public class StreutabellePublicTest {
```

  - Prüfung als Teil der gesamten Ausführungssteuerung über Linux-Shell-Skript: Suche im Output des javap-Disassemblers

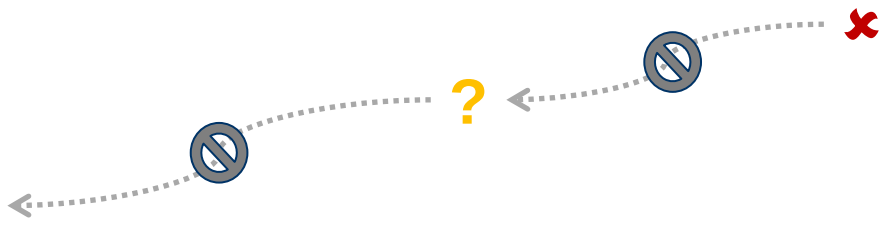
Folgefehler verhindern

- Szenario:
  - aufeinander aufbauende Methoden mit getrennten Testfällen

- Beispiel: QuickSort:
  - Aufrufe: **sort** $\xrightarrow{ruft}$ **partition** $\xrightarrow{ruft}$ **choosePivot** $\xrightarrow{ruft}$ **swap**
  - Code:
  - Tests:


```
@Replace({ "QuickSort.swap", "QuickSort.choosePivot" })
public void test_partition {
```

  - **test_swap:**
  - **test_…**
  - **test_partition:**



Agenda

- Hintergrund und Motivation
- Ablauf der Bewertung mit *AuDoscore*
- Testfall-Erstellung aus Sicht des Dozenten
- **Technische Umsetzung**
- Erfahrungen mit *AuDoscore*

Technische Umsetzung: `PointsLogger` & `PointsSummary`

- *AuDoscore* ist Open Source:
  - **`https://github.com/FAU-Inf2/AuDoscore`**

- Intuitive Erweiterungen für JUnit:

```
class PointsLogger extends TestWatcher {
```

JUnit liefert Infos zu jedem Testfall: starting, **failed**, skipped, **succeeded**

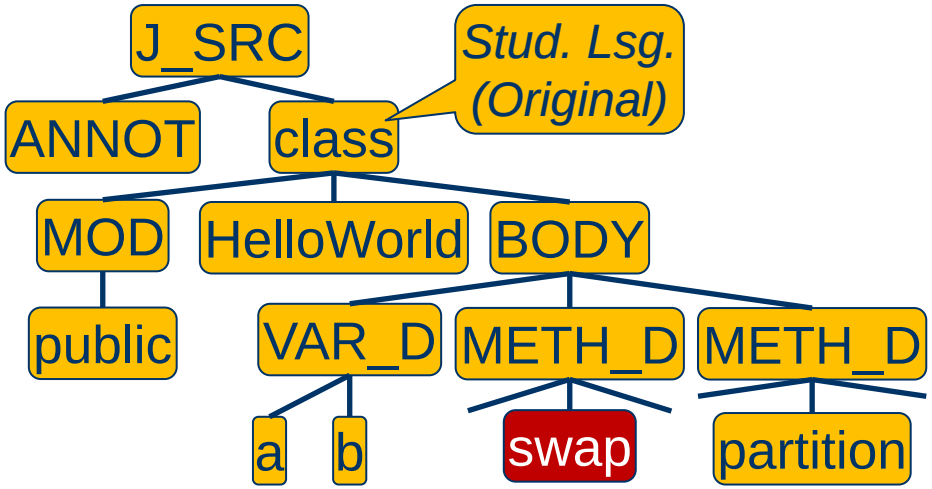
```
class PointsSummary extends ExternalResource {
```

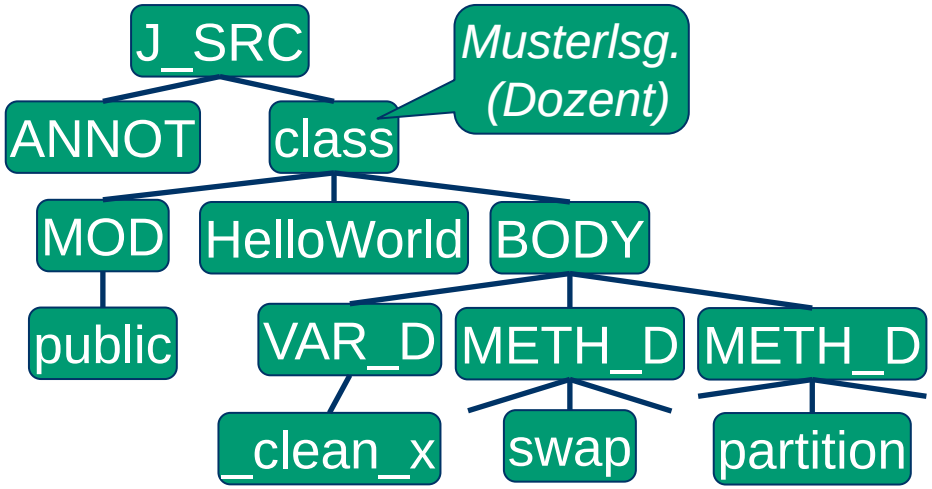
- installiert **`SecurityManager`**
- ersetzt **`stdOut/stdErr`**
- erzeugt Ergebnisbericht

JUnit informiert über gesamte Testsuite: before, **apply**, **after**

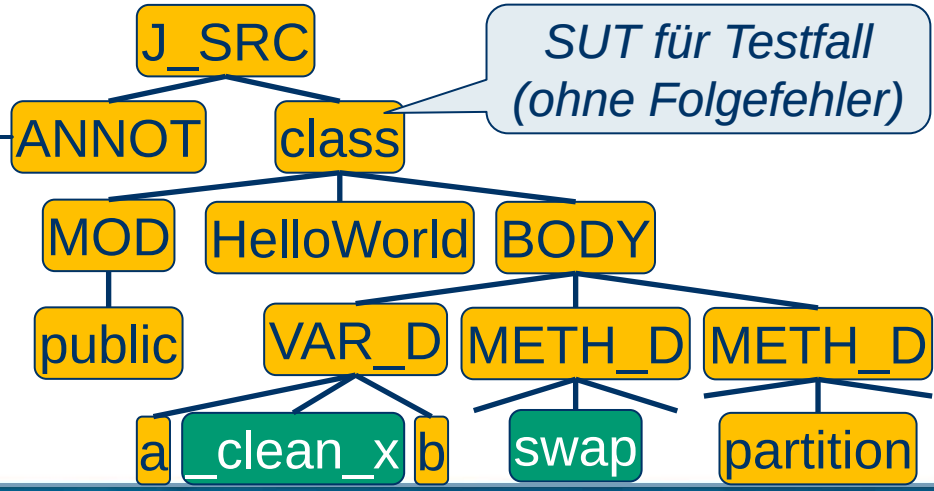
Technische Umsetzung: `@Replace`

- **`@Replace`** nutzt **`com.sun.source.util`** aus JDK:
  - Operationen auf Abstrakten Syntax-Bäumen (AST)






```
@Replace({ "QuickSort.swap",
    "QuickSort.choosePivot" })
public void test_partition {
```



AuDoscore für Scala-Code – exakt wie für Java:

```
import tester.annotations._
import Dijkstra._
```

```
@Exercises(Array(new Ex(exID = "Dijkstra.reachable", points = 2),
                 new Ex(exID = "Dijkstra.shortest", points = 2) ))
@Forbidden(Array("scala.collection.mutable"))
@NotForbidden(Array("scala.collection.mutable.StringBuilder",
                    "scala\\$App\\$\\$initCode"))
class DijkstraTestEST extends JUnitWithPoints {
```

Vererbt die Points-Rules
(erspart explizite Deklaration)

```
  @Test(timeout = 2000)
  @Points(exID = "Dijkstra.shortest", bonus = 1,
      comment = "Test shortest() with example from exercise sheet")
  @Replace(Array("Dijkstra.reachable"))
  def testShortestExample: Unit = {
    assertEquals("shortest(reachable(g))(0) returns wrong result",
        Set((0,0),(1,3),(2,5),(3,3),(4,2)),
        shortest(reachable(g))(0).toSet)
  }
```

Agenda

- Hintergrund und Motivation
- Ablauf der Bewertung mit *AuDoscore*
- Testfall-Erstellung aus Sicht des Dozenten
- Technische Umsetzung
- **Erfahrungen mit *AuDoscore***

Erfahrungen mit *AuDoscore* - AuD

- in 5 Semestern automatisch bewertet:
  - 4.149.117 Java-LOC in 46.619 Dateien (36.312 Abgaben)

	WS14/15	SS15	WS15/16	SS16	WS16/17
# Studierende	668	198	716	172	733
# Prog.-Aufg.	37	30	38	32	34
# Java-Abgaben	10.478	1.263	12.526	856	11.189
# Java-Dateien	12.685	1.647	17.274	1.167	13.846
# LOC	1.230.394	94.069	1.528.133	80.859	1.215.662
# Pub/Sec-Tests	160/562	147/737	336/810	143/635	224/804
# Test-Korrektur	26	24	24	27	21
# manuelle Bew.	102	9	723	2	13

Manuelle Überprüfung auf Rekursion

Erfahrungen mit *AuDoscore* – Tipps & Tricks

- Überprüfung auf Rekursion:
  - Vorgabe einer „Call-Back“-Methode, die von den Studierenden im Basisfall der Rekursion aufgerufen werden muss
  - => Rekursionstiefe über Stacktrace im *AuDoscore*-Test prüfen

- Betrugsversuche:
  - Studierende versuchen, die Erwartungen der Testfälle (PubTest) ohne echte Funktionalität zu erfüllen („Mock“!)
  - => „Anti-Cheat-Malus-Test“ notwendig
    - z.B. Diversität der Rückgaben für verschiedene Eingaben zählen

Zukunftspläne…

- Berücksichtigung nicht-funktionaler Aspekte
  - z.B. Codierrichtlinien
- Unterstützung weiterer (JVM-)Programmiersprachen
- Gezielte Unterstützung beim Test paralleler Programme
- Integration externer Bibliotheken
- Verfeinerung von **`@Forbidden`**
  - z.B. auf einzelne Klassen/Methoden

Erfahrungen mit *AuDoscore* - PFP

- in 3 Semestern automatisch bewertet:
  - 18.645 Scala-LOC in 721 Dateien/Abgaben

	WS15/16	SS16	WS16/17
# Studierende	24	390	51
# Prog.-Aufg.	12	7	7
# Scala-Abgaben	1	699	21
# Scala-Dateien	1	699	21
# LOC	14	18.144	487
# Pub/Sec-Tests	24/21	15/54	15/54
# Test-Korrektur	0	6	0
# manuelle Bew.	0	76	0