

Themenübersicht



Das bin ich

mm

40

60

80

100

120

- Felix Bachmann
- Inf_40udent im 4. Semester
- erstes Tutorium
- E-Post-Adresse: felix.bachmann@ewetel.net

60

80

... und ihr?

mm

40

60

80

100

120

- Name
- Studiengang und Semester
- erlernte Programmiersprachen, Lieblingsprogrammiersprache
- Erfahrung mit Git/Maven oder ähnlichen Tools?
- Von dem Tutorium erwarte ich...

60

80

cool

mm

40

60

80

100

120

- mitdenken
- Fragen stellen
- Fragen beantworten
- es 40 r & trinken
- gehen
- schlafen

!cool

60

- laut sein
- stören
- andere ablenken

80

cool

mm

40

60

80

100

120

- mitdenken
- Fragen stellen
- Fragen beantworten
- es 40 r & trinken
- gehen
- schlafen

!cool

60

- laut sein
- stören
- andere ablenken

80

mm

40

60

80

100

120

- Bestehen des Scheins Voraussetzung zum Bestehen des Moduls
- neue Übungsblätter ungefähr alle 2 Wochen \Rightarrow 1+6 Blätter
- ab 50% der Punkte habt ihr sicher bestanden
- ⁴⁰Besprechung der Musterlösung

■ Abgaben

- Theorieaufgaben (handschriftlich!) + Deckblatt im 3. Stock
- Programmieraufgaben auf <http://lez.ipd.kit.edu>
- ⁶⁰Plagiate können teuer werden
- Deadlines sind hart!
- keine Abgabe per Mail!

80

mm

40

60

80

100

120

- Bestehen des Scheins Voraussetzung zum Bestehen des Moduls
- neue Übungsblätter ungefähr alle 2 Wochen \Rightarrow 1+6 Blätter
- ab 50% der Punkte habt ihr sicher bestanden
- ⁴⁰Besprechung der Musterlösung
- Abgaben
 - Theorieaufgaben (handschriftlich!) + Deckblatt im 3.Stock
 - Programmieraufgaben auf <http://lez.ipd.kit.edu>
 - ⁶⁰Plagiate können teuer werden
 - Deadlines sind hart!
 - keine Abgabe per Mail!

80

mm

40

60

80

100

120

- Wann?: ab dem 15.05 14-tägig
- Wo?: Raum -107
- Was?:
 - Wiederholung des VL-Stoffs
 - “Rechnen“ von Aufgaben (Altklausuren)
 - ggf. Tipps für die Übungsblätter
- Folien gibt’s im Ilias und auf www.github.com/malluce/swt1-tut
- Fragen stellen !!

80

Fragen zu Übung(sblätter), Vorlesung

mm

40

60

80

100

120

erst im Forum, auf Google oder Stackoverflow nachschauen, dann

- ne 40 r Forum-Thread anlegen
- falls nicht öffentlich postbar: Mail an mich oder swt1@ipd.kit.edu (nur im Notfall)

60

80

Was ihr bisher getan haben solltet..

mm

40

60

80

100

120

Installation von:

- Eclipse (incl. CheckStyle und EclEmma)

Überblick über:

- Maven
- Git

Tut euch den Gefallen

- Installiert Git manuell!

Probleme mit der Installation? \Rightarrow kommt nach dem Tut nach vorne

80

mm

40

60

80

100

120

- Unittest-Tool für Java-Klassen
- über die pom.xml mit scope "test" einbinden
- Nicht öffentliche Methoden testen
- Konventionen:
 - Für Klasse Hallo Testklasse HalloTest schreiben
 - Methode hallo(Object o) wird z.B. durch die Methode testHalloWithNull() getestet

60

80

mm

40

60

80

100

120

Methoden können mit Annotationen (@XYZ) versehen werden
Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

80

mm

40

60

80

100

120

Methoden können mit Annotationen (@XYZ) versehen werden
Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

80

mm

40

60

80

100

120

Methoden können mit Annotationen (@XYZ) versehen werden
Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

80

mm

40

60

80

100

120

Methoden können mit Annotationen (@XYZ) versehen werden
Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

80

mm

40

60

80

100

120

Methoden können mit Annotationen (@XYZ) versehen werden
Aufbau:

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)
- @Test (vergleichen erwartetes und reales Ergebnis, schlagen ggf. fehl, Ausführung in beliebiger Reihenfolge)
- @After (wird nach jeder Test-Methode einmal ausgeführt)
- @AfterClass (wird am ende einmal ausgeführt)

80

mm

40

60

80

100

120

- `org.junit.Assert` bietet diverse Methoden, um Ergebnis mit Erwartung abzugleichen
- zu jeder Methode kann als erstes Argument ein String mitgegeben werden (wird bei Fehlschlag angezeigt)

Beispiele:

- `assertArrayEquals(int[] expected, int[] actual)`
- `assertNotNull(Object obj)`
- `assertSame(Object expected, Object actual)`

80

Zu testende Methode in der Klasse Hallo mm 40 60 80 100 120

```
public static int add(int a, int b) {  
    return a + b;  
}
```

40

Testmethode in der Klasse HalloTest

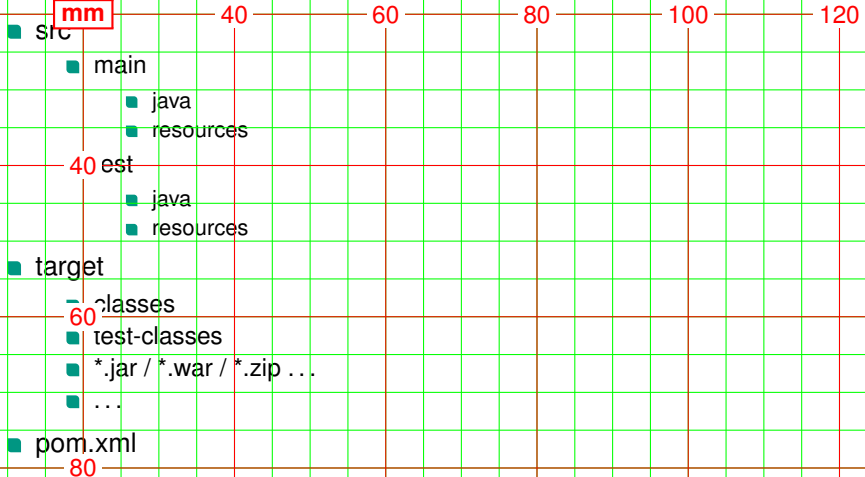
```
@Test  
public void testAdd() {  
    Assert.60assertEquals(7, Hallo.add(5 + 2));  
}
```

(mehr Beispiele später)

80

- Maven ist Juddis... und heißt „Sammler des Wissens“
- Build-Management-Tool (Automatisierung von möglichst vielen Schritten)
- Maven ist in jeder Eclipse-Installation integriert
⇒ keine manuelle Installation nötig
- Aufgaben von Maven
 - Strukturierung (durch vorgegebene Verzeichnisstruktur)
 - Kompilieren
 - Testen
 - Verwalten von Abhängigkeiten
 - Verpacken

Verzeichnisstruktur:



mm

40

60

80

100

120

- pom steht für "Project Object Model"
- konfiguriert euer Maven Projekt im XML-Format (gefüllt durch default-Werte)
 - Wo sucht Maven Tests?
 - Wohin speichert Maven Build-Dateien?
 - In welches Format soll das Projekt verpackt werden?
 - ...
- Eclipse-Plugin bietet GUI

80

mm

40

60

80

100

120

Wichtige Befehle

mvn compile	kompiliert Quelltexte zu .class-Dateien
mvn test	kompiliert Test-Quelldateien zu .class-Dateien, führt Tests aus und zeigt Ergebnisse an
mvn package	verpackt euer Projekt in eine Datei (.war/.jar/.zip)
mvn clear	leert euren target-Ordner

60

80

mm

40

60

80

100

120

Lösungsansätze:

- Rechtsklick auf Projekt \Rightarrow Maven \Rightarrow Update Maven Project \Rightarrow Haken bei "Force Update..."
 - Synchronisiert pom.xml mit Projekt, aktualisiert Abhängigkeiten
- mvn clean
 - vielleicht war der target-Ordner verschmutzt
- C:/Users/MeinName/.m2/ löschen und mvn compile (oder mvn package) ausführen
 - löscht alle Dependencies und lädt sie neu runter (ab und zu lädt man leider korrupte Dateien runter oder Dateien fehlen)

80

Warum Versionsverwaltung?

mm	40	60	80	100	120
final-09-03 (changed split-meth-)			01.07.2016 17:47	Dateiordner	
final-12-02			01.07.2016 17:47	Dateiordner	
final-13-02			01.07.2016 17:47	Dateiordner	
final-14-02			01.07.2016 17:47	Dateiordner	
final-15-02			01.07.2016 17:47	Dateiordner	
final-16-02			01.07.2016 17:47	Dateiordner	
final-17-02			01.07.2016 17:47	Dateiordner	
final-20-02(1)			01.07.2016 17:47	Dateiordner	
final-20-02(2)			01.07.2016 17:47	Dateiordner	
final-25-02(passed public tests)			01.07.2016 17:47	Dateiordner	
final-26-02(all commands implemented)			01.07.2016 17:47	Dateiordner	
final-27-02(version 1.0 - works so far)			01.07.2016 17:47	Dateiordner	
final-29-02(version 1.1 - finished)			01.07.2016 17:47	Dateiordner	

So nicht!



mm

40

60

80

100

120

- git ist Englisch, bedeutet Schwachkopf, Penner oder Nudelauge (?)
- dezentrales Versionsverwaltungssystem
- wichtig! (universell)

60

80

Umgang mit der Kommandozeile (cmd)

Nötig?

mm

40

60

80

100

120

Wichtige Befehle - Navigation

<code>cd test</code>	Wechselt in das Verzeichnis test.
<code>dir</code> <code>ls</code>	Zeigt Inhalt des aktuellen Ordners an.
<code>.</code>	= aktuelles Verzeichnis
<code>..</code>	= übergeordnetes Verzeichnis

Hacks 60

- Mit den Pfeiltasten können bereits eingegebene Befehle durchgescrollt werden.
- Tabulator = Autovervollständigung

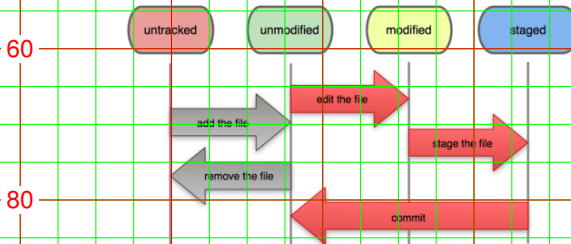
80

Wichtige Befehle

<code>git mm</code>	40	60	80	100	120
<code>git log</code>					
<code>git status</code>					
<code>git checkout</code>					
<code>git add</code>	40				
<code>git commit -m "message"</code>					

Initialisiert ein neues Git Repo.
 Zeigt alle vergangenen Commits.
 Zeigt den Status der Dateien im Repo.
 Lässt HEAD zwischen Commits springen.
 Fügt Datei(en) zur Staging Area hinzu.
 Erzeugt einen Commit.

File Status Lifecycle



mm

40

60

80

100

120

- Datei, die Namen von Pfaden/ Dateien enthält, die von git ignoriert werden sollen (z.B IDE-spezifisches)
- Beispiele:
 - target/
 - *.java
 - dis.like
- # (60) # als Kommentar-Zeichen

80

mm

40

60

80

100

120

Live-Demo

40

60

80

Tipps - 1. Übungsblatt

Aufgabe 1: Altsoftware vorbereiten

- **mm** Lösungsschritt für Umgang mit Maven, Git, Checkstyle - da müsst ihr durch **40** **60** **80** **100** **120**
- Google ist euer Freund (meistens)

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- Als **60** benutzen

Aufgabe 3: Testüberdeckung

- Mockito klingt komplizierter als es ist (schaut mal auf <http://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html>) **80**

Tipps - 1. Übungsblatt

Aufgabe 1: Altsoftware vorbereiten

- **mm** Lösungsschritt für Umgang mit Maven, Git, Checkstyle - da müsst ihr durch **40** **60** **80** **100** **120**
- Google ist euer Freund (meistens)

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- **As 60** ts benutzen !

Aufgabe 3: Testüberdeckung

- Mockito klingt komplizierter als es ist (schaut mal auf <http://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html>) **80**

Tipps - 1. Übungsblatt

Aufgabe 1: Altsoftware vorbereiten

- **mm** Lösungsschritt für Umgang mit Maven, Git, Checkstyle - da müsst ihr durch **40** **60** **80** **100** **120**
- Google ist euer Freund (meistens)

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- **As 60** ts benutzen !

Aufgabe 3: Testüberdeckung

- Mockito klingt komplizierter als es ist (schaut mal auf <https://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html>) **80**

Denkt dran!

mm

40

60

80

100

120

Abgabe

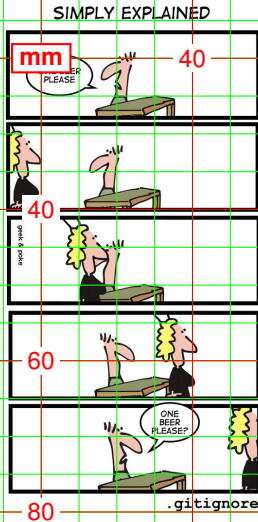
40

- in der LEZ bis zum 10.05, 12:00
- falls ihr ein Feedback wollt, werft das Deckblatt ein

60

80

Bis dann! (dann=15.05.17)



geek-and-poke.com/geekandpoke/2012/11/7/simply-explained.html