

Softwaretechnik 1 - 0. Tutorium

Tutorium 17

Felix Bachmann | 30.04.2019

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



- 1 Organisatorisches
- 2 Vorbereitungsblatt
- 3 JUnit4
- 4 Maven
- 5 Git
- 6 Tipps

Name

- E-Post-Adresse: *felix.bachmann*@ewetel.net
- Infostudent im 1. Master-Semester
- Bachelor angefangen im WS 15/16
- im WS18/19 BA am Institut für Telematik
 - software-basierte Netzwerke
- drittes Tutorium
 - SS17: SWT1
 - SS18: SWT1

- Name
- praktische Erfahrung
 - außerhalb der Uni?
 - Programmiersprachen?
 - Erfahrung mit Git/Maven o.Ä.?
- Von dem Tutorium erwarte ich...

- Wann?: ab heute 14-tägig (Di. 11:30-13:00)
- Wo?: Raum -119
- Was?:

- Wann?: ab heute 14-tägig (Di. 11:30-13:00)
- Wo?: Raum -119
- Was?:
 - Feedback letztes Blatt
 - Korrekturzeit i.d.R. < 1 Woche
 - Wiederholung des VL-Stoffs
 - “Rechnen“ von Aufgaben (Altklausuren)
 - Tipps nächstes Blatt

- Wann?: ab heute 14-tägig (Di. 11:30-13:00)
- Wo?: Raum -119
- Was?:
 - Feedback letztes Blatt
 - Korrekturzeit i.d.R. < 1 Woche
 - Wiederholung des VL-Stoffs
 - “Rechnen“ von Aufgaben (Altklausuren)
 - Tipps nächstes Blatt
- Folien gibt's im Ilias und auf www.github.com/malluce/swt1-tut

- Wann?: ab heute 14-tägig (Di. 11:30-13:00)
- Wo?: Raum -119
- Was?:
 - Feedback letztes Blatt
 - Korrekturzeit i.d.R. < 1 Woche
 - Wiederholung des VL-Stoffs
 - “Rechnen“ von Aufgaben (Altklausuren)
 - Tipps nächstes Blatt
- Folien gibt's im Ilias und auf www.github.com/malluce/swt1-tut
- **Fragen stellen !!**

- am 11.06. bin ich nicht da
 - Tut am 11.06. fällt aus
- Alternativen für euch
 - Ersatztutorium
 - eine Woche davor (04.06) oder
 - eine Woche danach (18.06)
 - könnt auch anderes Tut besuchen

Abstimmung: Ersatztermin

- Abstimmung über Ersatztermin
 - <https://forms.gle/kX18JiigVw1mSRPz5>



cool

- **mitdenken**
- **Fragen stellen**
- **Fragen beantworten**
- essen & trinken
- gehen
- schlafen

cool

- **mitdenken**
- **Fragen stellen**
- **Fragen beantworten**
- essen & trinken
- gehen
- schlafen

!cool

- laut sein
- stören
- andere ablenken

- Bestehen des Scheins Voraussetzung zum Bestehen des SWT1-Moduls
 - Modul Voraussetzung für PSE (unter anderem)
- 14-tägige Übungsblätter
- ab 50% der Punkte habt ihr sicher bestanden
- Besprechung der Musterlösung
- Abgaben
 - Theorieaufgaben im 3.Stock \implies Holzkasten
 - Programmieraufgaben auf <http://lez.ipd.kit.edu>

- Theorieaufgaben
 - handschriftlich
 - leserlich
 - Deckblatt (von Vorlage)

- Theorieaufgaben
 - handschriftlich
 - leserlich
 - Deckblatt (von Vorlage)
- Programmieraufgaben (Verstoß = Punktabzug)
 - Git
 - JavaDoc
 - CheckStyle (benutzt „Save Actions“, erspart einige Arbeit)
 - Stil (sinnvolle Namen, Kommentare etc.)

- Theorieaufgaben
 - handschriftlich
 - leserlich
 - Deckblatt (von Vorlage)
- Programmieraufgaben (Verstoß = Punktabzug)
 - Git
 - JavaDoc
 - CheckStyle (benutzt „Save Actions“, erspart einige Arbeit)
 - Stil (sinnvolle Namen, Kommentare etc.)
- keine Abgabe per Mail

- Theorieaufgaben
 - handschriftlich
 - leserlich
 - Deckblatt (von Vorlage)
- Programmieraufgaben (Verstoß = Punktabzug)
 - Git
 - JavaDoc
 - CheckStyle (benutzt „Save Actions“, erspart einige Arbeit)
 - Stil (sinnvolle Namen, Kommentare etc.)
- keine Abgabe per Mail
- harte Deadlines
 - 10 Minuten vor Abgabe in der ATIS das Deckblatt drucken ist unklug

- Theorieaufgaben
 - handschriftlich
 - leserlich
 - Deckblatt (von Vorlage)
- Programmieraufgaben (Verstoß = Punktabzug)
 - Git
 - JavaDoc
 - CheckStyle (benutzt „Save Actions“, erspart einige Arbeit)
 - Stil (sinnvolle Namen, Kommentare etc.)
- keine Abgabe per Mail
- harte Deadlines
 - 10 Minuten vor Abgabe in der ATIS das Deckblatt drucken ist unklug
- Plagiate teuer (ÜBs alleine lösen!)

- Theorieaufgaben
 - handschriftlich
 - leserlich
 - Deckblatt (von Vorlage)
- Programmieraufgaben (Verstoß = Punktabzug)
 - Git
 - JavaDoc
 - CheckStyle (benutzt „Save Actions“, erspart einige Arbeit)
 - Stil (sinnvolle Namen, Kommentare etc.)
- keine Abgabe per Mail
- harte Deadlines
 - 10 Minuten vor Abgabe in der ATIS das Deckblatt drucken ist unklug
- Plagiate teuer (ÜBs alleine lösen!)
- keine Punkte geschenkt \implies früh anfangen

erst im Forum, auf Google oder Stackoverflow nachschauen, dann

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

Forum nicht zum Teilen von Lösungen verwenden → Plagiate!

- dieses Jahr wird in der Übung Java 11 verwendet
- Problem(chen)
 - evtl. Eclipse-Aktualisierung für Java 11-Unterstützung nötig
 - Oracle JDK 11 nur mit Oracle-Account
- Lösungen
 - Eclipse aktualisieren :)
 - Oracle JDK 12 kann man ohne Account beziehen
 - damit kann man auch für Java 11 kompilieren
 - OpenJDK 11 (und 12) nutzen
 - aber kommen als .zip ohne Installer
 - Windows: PATH-Variable manuell um bin/ erweitern

■ Programmieren \Rightarrow SWT1 \Rightarrow PSE

- Programmieren \implies SWT1 \implies PSE
- den Hacker strukturieren

- Programmieren \implies SWT1 \implies PSE
- den Hacker strukturieren
- Tools (Versionsverwaltung, Build-Management) erlernen

Was ihr bisher getan haben solltet..

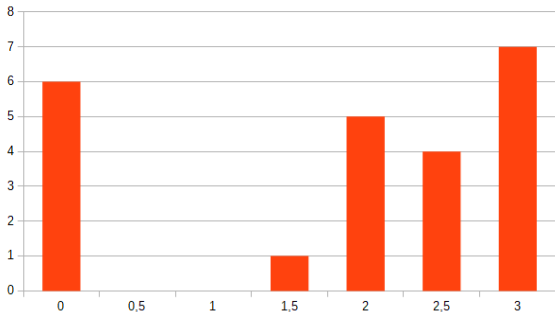
Installation von:

- Eclipse (inkl. CheckStyle und EclEmma)
- Git

Überblick über:

- Maven
- Git

Anmelden in der LEZ!



Kriterien für Punkte

je 1P.:

- Import + Abgabe (pom.xml muss stimmen)
- CheckStyle (+ **sinnvolles** JavaDoc)
- Implementierung (EditMe + EditMeTest)

Achtet zukünftig besonders auf:

- **sinnvolles und vollständiges** JavaDoc (siehe nächste Folie)

Achtet zukünftig besonders auf:

- **sinnvolles und vollständiges** JavaDoc (siehe nächste Folie)
- alte Kommentare (TODOs...) entfernen

Achtet zukünftig besonders auf:

- **sinnvolles und vollständiges** JavaDoc (siehe nächste Folie)
- alte Kommentare (TODOs...) entfernen
- tut etwas in tearDown() \implies Objekte nullen...

Achtet zukünftig besonders auf:

- **sinnvolles und vollständiges** JavaDoc (siehe nächste Folie)
- alte Kommentare (TODOs...) entfernen
- tut etwas in tearDown() \implies Objekte nullen...
- CheckStyle für jede *.java-Datei, insbesondere auch Tests!

Bitte auf Englisch.

```
package tuts.swt1;
```

```
/**
 * This class demonstrates how to use JavaDoc.
 * @author Felix Bachmann
 * @version 1.0
 */
public class JavaDocDemonstration {

    /**
     * Returns a random number in a specific range.
     * @param start the start of the range
     * @param end the end of the range
     * @return a random number in between start and end
     * @throws IllegalArgumentException is thrown if start is 1337
     */
    public int getRandomNumber(int start, int end) throws IllegalArgumentException {
        int random = 0;
        if (start == 1337) {
            throw new IllegalArgumentException("sorry, no leet numbers");
        }
        // very nice calculation
        return random;
    }
}
```

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

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

- @BeforeClass (wird als erstes einmal ausgeführt)

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

- @BeforeClass (wird als erstes einmal ausgeführt)
- @Before (wird vor jeder Test-Methode einmal ausgeführt)

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)

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)

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)

- 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:

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

JUnit4 - eine Testmethode

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

Wie sieht Testmethode aus?

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

Wie sieht Testmethode aus?

Testmethode

```
// imports..  
public class HalloTest {  
    @Test  
    public void testAdd() {  
        Assert.assertEquals(7, Hallo.add(5, 2));  
    }  
}
```


A, B oder C?

Welche Annotation führt dazu, dass die annotierte Methode nach jeder mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- C: @AfterClass

A, B oder C?

Welche Annotation führt dazu, dass die annotierte Methode nach jeder mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- C: @AfterClass

B (Ignore = Methode nicht ausführen, AfterClass = nach Ausführung aller Tests einmal annotierte Methode)

A, B oder C?

Welche Annotation führt dazu, dass die annotierte Methode nach jeder mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- C: @AfterClass

B (Ignore = Methode nicht ausführen, AfterClass = nach Ausführung aller Tests einmal annotierte Methode)

Wahr oder falsch?

Die mit @Test versehenen Methoden werden in der Reihenfolge ausgeführt, in der sie im Quellcode stehen.

A, B oder C?

Welche Annotation führt dazu, dass die annotierte Methode nach jeder mit @Test versehenen Methode einmal ausgeführt wird?

- A: @Ignore
- B: @After
- C: @AfterClass

B (Ignore = Methode nicht ausführen, AfterClass = nach Ausführung aller Tests einmal annotierte Methode)

Wahr oder falsch?

Die mit @Test versehenen Methoden werden in der Reihenfolge ausgeführt, in der sie im Quellcode stehen.

Falsch, „zufällig“

Wahr oder falsch?

Um Ergebnisse von Methodenaufrufen mit dem erwarteten Ergebnis abzugleichen, benutzt man Methoden aus `junit.framework.Assert`.

Wahr oder falsch?

Um Ergebnisse von Methodenaufrufen mit dem erwarteten Ergebnis abzugleichen, benutzt man Methoden aus `junit.framework.Assert`.

Falsch (deprecated, `org.junit.Assert` benutzen!)

- Build-Management-Tool (Automatisierung von möglichst vielen Schritten)

- Build-Management-Tool (Automatisierung von möglichst vielen Schritten)
- Maven ist in jeder Eclipse-Installation integriert
⇒ keine manuelle Installation nötig

- 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)

- 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

- 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

- 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

- 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

- 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
- Phasen = Maven-Befehle (z.B. `mvn package`), in Eclipse in Feld Goal
 - Ausführung einer Phase führt ggf. vorangehende Phasen aus
 - <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>
 - <https://stackoverflow.com/questions/26607834/maven-lifecycle-vs-phase-vs-plugin-vs-goal>

Verzeichnisstruktur:

- src
 - main
 - java
 - resources

Verzeichnisstruktur:

- src
 - main
 - java
 - resources
 - test
 - java
 - resources

Verzeichnisstruktur:

- src
 - main
 - java
 - resources
 - test
 - java
 - resources
- target
 - classes
 - test-classes
 - *.jar / *.war / *.zip ...
 - ...

Verzeichnisstruktur:

- src
 - main
 - java
 - resources
 - test
 - java
 - resources
- target
 - classes
 - test-classes
 - *.jar / *.war / *.zip ...
 - ...
- pom.xml

- pom steht für “Project Object Model“
- konfiguriert euer Maven Projekt im XML-Format (gefüllt durch default-Werte)
 - {Artifact/Group}ID

- pom steht für “Project Object Model“
- konfiguriert euer Maven Projekt im XML-Format (gefüllt durch default-Werte)
 - {Artifact/Group}ID
 - Wo sucht Maven Tests?

- pom steht für “Project Object Model“
- konfiguriert euer Maven Projekt im XML-Format (gefüllt durch default-Werte)
 - {Artifact/Group}ID
 - Wo sucht Maven Tests?
 - Wohin speichert Maven Build-Dateien?

- pom steht für “Project Object Model“
- konfiguriert euer Maven Projekt im XML-Format (gefüllt durch default-Werte)
 - {Artifact/Group}ID
 - Wo sucht Maven Tests?
 - Wohin speichert Maven Build-Dateien?
 - In welches Format soll das Projekt verpackt werden?

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

Wichtige Befehle

```
mvn compile
```

 Quelltexte \Rightarrow .class-Dateien

Wichtige Befehle

<code>mvn compile</code>	Quelltexte \implies .class-Dateien
<code>mvn test</code>	Test-Quelldateien \implies .class-Dateien, führt Tests aus und zeigt Ergebnisse an

Wichtige Befehle

<code>mvn compile</code>	Quelltexte \Rightarrow .class-Dateien
<code>mvn test</code>	Test-Quelldateien \Rightarrow .class-Dateien, führt Tests aus und zeigt Ergebnisse an
<code>mvn package</code>	verpackt Projekt in eine Datei (.war/.jar/.zip)

Wichtige Befehle

<code>mvn compile</code>	Quelltexte \implies .class-Dateien
<code>mvn test</code>	Test-Quelldateien \implies .class-Dateien, führt Tests aus und zeigt Ergebnisse an
<code>mvn package</code>	verpackt Projekt in eine Datei (.war/.jar/.zip)
<code>mvn install</code>	installiert Projekt lokal

Wichtige Befehle

<code>mvn compile</code>	Quelltexte \implies .class-Dateien
<code>mvn test</code>	Test-Quelldateien \implies .class-Dateien, führt Tests aus und zeigt Ergebnisse an
<code>mvn package</code>	verpackt Projekt in eine Datei (.war/.jar/.zip)
<code>mvn install</code>	installiert Projekt lokal
<code>mvn deploy</code>	liefert Projekt (remote) aus

wasserfallartige Ausführung! (Tafel)

Lösungsansätze:

- Rechtsklick auf Projekt \Rightarrow Maven \Rightarrow Update Maven Project
 \Rightarrow Haken bei “Force Update...”
 - Synchronisiert pom.xml mit Projekt, aktualisiert Abhängigkeiten

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

Lösungsansätze:

- Rechtsklick auf Projekt \implies Maven \implies Update Maven Project
 \implies 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)

A, B, C oder D?

Welcher Maven-Befehl kompiliert die Testklassen?

- A: mvn compile
- B: mvn package
- C: mvn test
- D: mvn test-compile

A, B, C oder D?

Welcher Maven-Befehl kompiliert die Testklassen?

- A: mvn compile
- B: mvn package
- C: mvn test
- D: mvn test-compile

B,C,D. A kompiliert nur src-Quelltexte

Wahr oder falsch?

Damit Maven funktioniert, muss die komplette pom.xml manuell ausgefüllt werden.

A, B, C oder D?

Welcher Maven-Befehl kompiliert die Testklassen?

- A: mvn compile
- B: mvn package
- C: mvn test
- D: mvn test-compile














B,C,D. A kompiliert nur src-Quelltexte

Wahr oder falsch?

Damit Maven funktioniert, muss die komplette pom.xml manuell ausgefüllt werden.

Falsch, default-Werte!

Warum Versionsverwaltung?

 final1-09-03(changed split-method)	01.07.2016 17:47	Dateiordner
 final1-12-02	01.07.2016 17:47	Dateiordner
 final1-13-02	01.07.2016 17:47	Dateiordner
 final1-14-02	01.07.2016 17:47	Dateiordner
 final1-15-02	01.07.2016 17:47	Dateiordner
 final1-16-02	01.07.2016 17:47	Dateiordner
 final1-17-02	01.07.2016 17:47	Dateiordner
 final1-20-02(1)	01.07.2016 17:47	Dateiordner
 final1-20-02(2)	01.07.2016 17:47	Dateiordner
 final1-25-02(passed public tests)	01.07.2016 17:47	Dateiordner
 final1-26-02(all commands implemented)	01.07.2016 17:47	Dateiordner
 final1-27-02(version 1.0 - works so far)	01.07.2016 17:47	Dateiordner
 final1-29-02(version 1.1 - finished)	01.07.2016 17:47	Dateiordner

So nicht!



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

Wichtige Befehle - Navigation

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

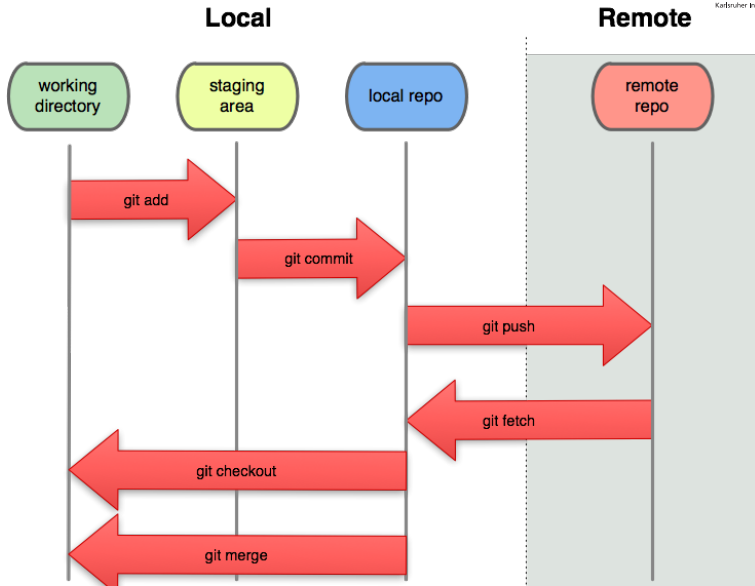
Hacks

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

Wichtige Befehle

<code>git init</code>	Initialisiert ein leeres Git-Repo.
<code>git log</code>	Zeigt alle vergangenen Commits.
<code>git status</code>	Zeigt den Status der Dateien im Repo.
<code>git checkout</code>	Lässt HEAD zwischen Commits springen.
<code>git add</code>	Fügt Datei(en) zur Staging Area hinzu.
<code>git commit -m "message"</code>	Erzeugt einen Commit.
<code>git branch hallo</code>	Erzeugt einen neuen Branch namens hallo.
<code>git merge hallo</code>	Merged den Branch hallo in den aktuellen.

Git - Workflow



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

- sinnvolle Commit-Nachrichten
- Dateien ggf. sinnvoll zu Commits zusammenfassen (mit git add)
- Übung: pro Teilaufgabe o.Ä. ein commit

Richtig oder falsch?

Mit `git commit "message"` wird ein neuer Commit erzeugt, dessen Commit-Nachricht `message` ist.

Richtig oder falsch?

Mit `git commit "message"` wird ein neuer Commit erzeugt, dessen Commit-Nachricht `message` ist.

Falsch, es fehlt die Option `-m`

Richtig oder falsch?

Git ist im Gegensatz zu SVN ein zentrales Versionsverwaltungssystem.

Richtig oder falsch?

Mit `git commit "message"` wird ein neuer Commit erzeugt, dessen Commit-Nachricht `message` ist.

Falsch, es fehlt die Option `-m`

Richtig oder falsch?

Git ist im Gegensatz zu SVN ein zentrales Versionsverwaltungssystem.

Falsch, andersrum.

Richtig oder falsch?

`git log` zeigt eine Liste aller bisher getätigten Commits an und zeigt dabei Informationen wie Datum, Zeit, Hashcode und Commitnachricht der jeweiligen Commits an.

Richtig oder falsch?

Mit `git commit "message"` wird ein neuer Commit erzeugt, dessen Commit-Nachricht `message` ist.

Falsch, es fehlt die Option `-m`

Richtig oder falsch?

Git ist im Gegensatz zu SVN ein zentrales Versionsverwaltungssystem.

Falsch, andersrum.

Richtig oder falsch?

`git log` zeigt eine Liste aller bisher getätigten Commits an und zeigt dabei Informationen wie Datum, Zeit, Hashcode und Commitnachricht der jeweiligen Commits an.

Richtig.

- altes Eclipse aktualisieren, ohne Neuinstallation (für Java 11 Support)
 - https://wiki.eclipse.org/FAQ_How_do_I_upgrade_Eclipse_IDE%3F
- Hilfe zu Mockito
 - <https://www.javacodegeeks.com/2012/05/mocks-and-stubs-understanding-test.html>

Aufgabe 1: Altsoftware vorbereiten

- löchriges Kochrezept für Umgang mit Maven, Git, Checkstyle
- bei Fehlern Google + Forum benutzen
- an Maven-Ordnerstruktur erinnern

Aufgabe 1: Altsoftware vorbereiten

- löchriges Kochrezept für Umgang mit Maven, Git, Checkstyle
- bei Fehlern Google + Forum benutzen
- an Maven-Ordnerstruktur erinnern

Aufgabe 2: Modultests

- Aufgaben zum Testen mit JUnit4
- Ordner sollen erstellt werden, wenn sie nicht existieren
- Tests ohne Erwartung sind keine Tests (Asserts oder @expect benutzen)

Aufgabe 3: Testüberdeckung

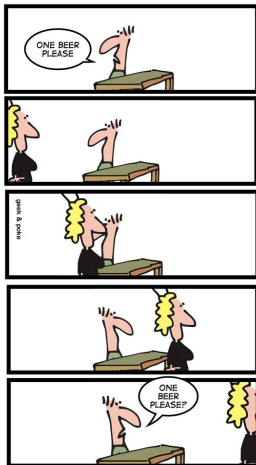
- Testüberdeckung mit EcEmma überprüfen
- Mockito klingt komplizierter als es ist
 - siehe Link, Abschnitt Mock Objekt

Abgabe

- in der LEZ bis zum 08.05, 12:00
- falls ihr schriftliches Feedback wollt, werft das Deckblatt ein

Bis dann! (dann:=14.05.19)

SIMPLY EXPLAINED



.gitignore

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