

# **Informatik**

## **2**

**Hinweise zu den Übungsaufgaben**

**Prof. Dr.-Ing. Holger Vogelsang**  
**B.Sc. Manuel Vogel**

**Sommersemester 2017**

**Don't  
panic**

# Inhaltsverzeichnis

---

<b>1</b>	<b>Einleitung</b>	<b>4</b>
1.1	Organisation der Übungsaufgaben . . . . .	4
1.2	Anerkennung bereits abgegebener Lösungen . . . . .	4
1.3	Bonusaufgabe . . . . .	5
1.4	Programmiertutorium . . . . .	5
1.5	Vorbereitungen . . . . .	5
<b>2</b>	<b>Hinweise zur Arbeit mit Eclipse</b>	<b>7</b>
2.1	Checkstyle . . . . .	7
2.2	FindBugs . . . . .	8
2.3	Kontextsensitive Hilfe . . . . .	9
2.4	Unit-Tests . . . . .	9
<b>3</b>	<b>Hinweise zur Arbeit mit Gitlab</b>	<b>14</b>
3.1	Anlegen eines Projektes im Gitlab . . . . .	14
3.2	Anlegen eines Projekts in Eclipse . . . . .	15
3.3	Konfiguration von Eclipse . . . . .	15
3.4	Erstellen der Lösung der Aufgabe . . . . .	17
3.5	Einfügen ins Git-Repository . . . . .	18
<b>4</b>	<b>Literaturverzeichnis</b>	<b>26</b>
<b>5</b>	<b>Stichwortverzeichnis</b>	<b>27</b>

# Einleitung

---

## 1.1 Organisation der Übungsaufgaben

Jeder von Ihnen muss seine eigene Lösung vorstellen. Zur Erlangung des Scheins müssen alle Pflichtaufgaben korrekt gelöst sein. Wichtig sind:

- korrekte Funktionsweise der Implementierung
- vollständige Kommentierung mit Javadoc (außer bei einfachen Getter- und Setter-Methoden)
- Einhaltung der Code-Konventionen von Oracle: Diese müssen mittels des Plugins für Checkstyle überprüft werden (siehe Abschnitt 2.1.1).
- Prüfen Sie Ihr Programm mit dem Plugin „Findbugs“ auf mögliche Schwachstellen (siehe Abschnitt 2.2.1).

Die Plugins „Checkstyle“ und „Findbugs“ gibt es für die Entwicklungsumgebungen Eclipse, Netbeans und IntelliJ IDEA.

## 1.2 Anerkennung bereits abgegebener Lösungen

Sollte jemand von Ihnen schon in früheren Semestern Lösung abgegeben haben, so werden diese anerkannt, wie in Tabelle 1.1 beschrieben.

**Tabelle 1.1:** Anrechnung

SS 2014	WS 2014	SS 2015	WS 2015	SS 2016	WS 2016	SS 2017
		3.4.5	3.6.5	3.1	1	1
3.1	3.1	3.1	3.1	3.2	2	2
3.2	3.2	3.2	3.2 – 3.3	3.3 – 3.4	3, 4	3, 4
3.3 – 3.5	3.3 – 3.4	3.3 – 3.4	3.4 – 3.6	3.5 – 3.7.4	5, 7, 8, 9	5, 7, 8, 9
3.6	3.5	3.5	3.7	3.8	6	6
3.7	3.6	3.6	3.8	3.9	10	10
3.8	4	4	4	4	4	4

## 1.3 Bonusaufgabe

Sie können die Bonusaufgabe zusätzlich zu den Pflichtaufgaben lösen. Dadurch erhalten Sie fünf Bonuspunkte in der Klausur.

## 1.4 Programmiertutorium

Parallel zu den „regulären“ Übungen, die es laut Studien- und Prüfungsordnung gibt, wird ein Programmiertutorium angeboten, die gerade Programmierneulingen einen leichteren Einstieg in die Software-Entwicklung mit Java ermöglichen sollen. Dazu finden Sie im Ilias separate Aufgaben. Damit Sie diese Aufgaben nicht alleine lösen müssen, wird im Semester eine unterstützende Übungsbetreuung an separaten Terminen angeboten (siehe Stundenplan). Sollten Sie die Bearbeitung der Basisübungen abgeschlossen haben, dann können Sie die Termine als zusätzliche betreute Zeit zur Bearbeitung der Pflichtaufgaben nutzen. Sollten Sie schon Programmierkenntnisse besitzen, aber mit C++ Probleme haben, dann finden Sie in im Ilias einige sehr einfache Aufgaben zu C und C++.

## 1.5 Vorbereitungen

Das Kapitel beschreibt die Installation der notwendigen Programme für das Labor.

### 1.5.1 Installation des JDK unter Windows

Zur reinen Ausführung von Java-Programmen genügt das JRE (Java Runtime Environment). Darin fehlen aber beispielsweise wichtige Werkzeuge wie der Compiler oder das Programm `javadoc`. Hier in der Vorlesung sollen eigene Programme erstellt werden. Daher muss das JDK (Java Development Kit, manchmal auch SDK genannt) in Version 8 verwendet werden. Installation:

- Download des JDK kostenlos unter <http://www.oracle.com/technetwork/java/index.html>
- Doppelklick auf das ausführbare JDK-Archiv, dann das Zielverzeichnis angeben
- Die dringend benötigte Dokumentation muss separat installiert werden. Sie sollte in das Verzeichnis der JDK-Software oder in ein eigenständiges Verzeichnis entpackt werden.
- Wenn das Verzeichnis der ausführbaren Programme nicht im Suchpfad liegt, sollte das Verzeichnis in die PATH-Variable aufgenommen werden. Die Programme können dann von der Kommandozeile aus aufgerufen werden (z.B. die Abfrage der JDK Versionsnummer mit `java -version`). Das ist für die Übungen aber nicht erforderlich.

### 1.5.2 Integrierte Entwicklungsumgebungen

Die Anwendung des „nackten“ JDKs ist nicht sehr komfortabel. Daher bietet sich die Verwendung einer integrierten Entwicklungsumgebung an. Auswahl:

- Eclipse: alle Plattformen, kostenlos unter <http://www.eclipse.org>.
- Netbeans: Oracle, alle Plattformen, kostenlos unter <http://netbeans.org/index.html>

- IntelliJ IDEA: JetBrains, alle Plattformen, kostenlose Community-Edition unter <http://www.jetbrains.com/idea/>. Den Key für eine Vollversion finden Sie im Ilias.
- JDeveloper: Oracle, alle Plattformen, kostenlos unter <http://www.oracle.com/technetwork/java/index.html>

Auf den Poolrechnern ist auf jeden Fall Eclipse installiert. Als Projektverzeichnis sollte ein Verzeichnis auf dem Netzlaufwerk gewählt werden. Das erspart das manuelle Backup! Das später aus Eclipse heraus benötigte Programm `javadoc` wird von Eclipse nicht immer automatisch gefunden. Es muss mit dem Java-Pfad eingebunden werden. Ihr Betreuer hilft Ihnen dabei. Am besten aber wäre es, wenn Sie Ihre Projekte direkt in Ihrem SVN-oder Git-Repository ablegen. Eine Anleitung dazu finden Sie in den Vorlesungsunterlagen.

## Hinweise zur Arbeit mit Eclipse

---

Die folgenden Abschnitte geben Ihnen noch Installationshinweise für Eclipse, die zur Lösung der Aufgaben benötigt werden.

### 2.1 Checkstyle

Alle Lösungen müssen die Code-Konventionen von Oracle einhalten. Die Überprüfung übernimmt das Eclipse-Plugin für „Checkstyle“. Nur wenn keine Warnungen in Form kleiner gelber Ausrufezeichen ausgegeben werden, wird eine Aufgabe abgenommen. In den Pools ist „Checkstyle“ bereits installiert.

#### 2.1.1 Checkstyle-Installation auf privaten Computern

Sie können das Plugin unter Eclipse mit `Help → Eclipse Marketplace` installieren. Dort geben Sie im Feld `Find` den Namen „Checkstyle“ ein und lassen den gefundenen Treffer installieren.


#### 2.1.2 Globale Checkstyle-Konfiguration

Die Standardkonfiguration von Checkstyle ist relativ „scharf“ eingestellt. Damit Sie nicht komplett verzweifeln, können Sie aus dem Ilias von den Seiten zu den Übungen eine entschärfte Konfigurationsdatei herunterladen (`Checkstyle-Informatik2`) und irgendwo im lokalen Dateisystem ablegen. Weitere Schritte in Eclipse:

- In Eclipse wird das Menü `Window → Preferences` ausgewählt.
- In der linken Baumansicht muss `Checkstyle` selektiert werden. Danach erfolgt ein Mausklick im Dialog auf den Button `New...`
- Im Dialog `External Configuration File` auswählen.
- Die Informatik2-Konfiguration einlesen, indem `Browse` ausgewählt wird.
- Der Konfiguration einen Namen geben (z.B. `Informatik2`).
- Den Dialog mit `OK` schließen und die neue Konfiguration mit `Set as Default` zum Standard erklären.

### 2.1.3 Projekt-spezifische Checkstyle-Konfiguration

In jedem Projekt muss Checkstyle explizit eingeschaltet werden:

- mit der rechten Maustaste auf das Projekt klicken und `Properties` auswählen
- Checkstyle wählen
- im erscheinenden Dialog die gewünschte Konfiguration auswählen
- dann den Haken vor `Checkstyle active for this project` setzen und mit `OK` bestätigen
- Regelverstöße werden im Projekt durch gelbe Markierungen am Rand angezeigt:  


Besitzt ein Projekt solche Verstöße, so werden die betroffenen Klassen auch im Package-Explorer markiert.

## 2.2 FindBugs

FindBugs analysierten Ihren Quelltext und versucht so, mögliche Fehlerquellen anzuzeigen. Nicht jede Markierung muss einen echten Fehler darstellen. Teilweise kann es sich auch nur um Hinweise handeln, dass hier in bestimmten Situationen Fehler auftreten können. Wenn Sie sicher sind, dass solche Situationen bei Ihnen nie auftreten können, dann ignorieren Sie diese Hinweise. Außerdem kann es vorkommen, dass FindBugs Stellen als fragwürdig markiert, obwohl sie absichtlich so erstellt wurden. Ein Beispiel dafür ist der Stringvergleich durch Vergleich der Referenzen auf die String-Objekte, anstatt die `equals`-Methode zu verwenden. Hier kann FindBugs nicht erkennen, ob Sie das absichtlich oder unabsichtlich gemacht haben. Sehen Sie die Meldungen von FindBugs einfach als Anregung, über den Code nachzudenken.

### 2.2.1 FindBugs-Installation

Sie können das Plugin unter Eclipse mit `Help → Eclipse Marketplace` installieren. Dort geben Sie im Feld `Find` den Namen „FindBugs“ ein und lassen den gefundenen Treffer installieren. Sollte mehr als ein Treffer sichtbar werden, dann nehmen Sie denjenigen, bei dem nicht einige Angaben in der Beschreibung `null` sind.

### 2.2.2 Verwendung

Die Anwendung könnte nicht einfacher sein: Nach einem Rechtsklick auf das Projekt wählen Sie `Find Bugs` und dann den Eintrag `Find Bugs`. In der Perspektive `Find Bugs` sehen Sie nach dem Durchlauf die Treffer. Eventuell müssen Sie manuell auf diese Perspektive umschalten. Hinter dem Projektnamen erscheint im „Package Explorer“ in Klammern die Anzahl möglicher Probleme.

### 2.2.3 Konfiguration

FindBugs lässt sich ziemlich gut konfigurieren, so dass Sie bestimmte Fehlerarten ein- oder ausschalten können. Die Standard-Konfiguration ist für uns schon nicht schlecht. Sie können aber unter Umständen unter `Window → Preferences → FindBugs` die Option `Multithreaded Correctness` noch ausschalten, weil das Thema Multithreading erst in der Betriebssystemvorlesung behandelt wird und Sie hier mit möglichen Gefahrenmeldungen in diesem Zusammenhang nichts anfangen können.



### 2.2.4 FindBugs in den Pool-Installationen

Sollte FindBugs auf den PCs bzw. virtuellen Maschinen der Pools keine Fehler melden, dann ist unter Umständen ein kleiner manueller Eingriff in Ihr Projekt erforderlich:

1. Nach einem Rechtsklick auf Ihr zu untersuchendes Projekt wählen Sie `Properties` und dort im Dialog `FindBugs`.
2. Selektieren Sie `Enable Project Specific Settings`.
3. Selektieren Sie alle Einträge in dem eingerahmten Kästchen `Report Visible Bug Categories`.
4. Verschieben Sie den Schieberegler `Minium rank to report` auf den Wert 20.
5. Stellen Sie den Wert von `Minimum confidence to report` auf `Low`.
6. Prüfen Sie mit FindBugs Ihr Projekt erneut.

## 2.3 Kontextsensitive Hilfe

Um die kontextsensitive Hilfe unter Eclipse richtig nutzen zu können, sind einige kleinere Vorarbeiten erforderlich. Normalerweise bietet Eclipse eine kontextsensitive Hilfe an, wenn Sie mit dem Mauszeiger auf eine Methode, eine Klasse usw. deuten. Im Poolraum werden Sie feststellen, dass das leider nicht immer funktioniert, weil der Proxy der Hochschule dieses so nicht zulässt. Es gibt mehrere Möglichkeiten, das zu umgehen. Beispielsweise kann in Eclipse die Verwendung des Proxys eingeschaltet werden. Dazu müssen die Anmeldedaten hinterlegt werden. Leider klappt auch das nicht immer zuverlässig. Sie können aber Eclipse anweisen, eine lokal installierte API-Dokumentation zu verwenden. Unter der URL [http://www.gnostice.com/nl\\_article.asp?id=209](http://www.gnostice.com/nl_article.asp?id=209) finden Sie eine genaue Beschreibung der Vorgehensweise. Den Installationsort der lokalen API-Dokumentation im Pool erfahren Sie von Ihren Betreuern. Wenn Sie eine lokale Eclipse-Installation auf Ihren privaten Computern auch so anpassen wollen, dann müssen Sie neben dem JDK auch die Hilfe-Dokumente installieren. Sie finden diese unter der URL <http://tinyurl.com/355cx3m><sup>1</sup> (siehe „Java SE 8 Documentation“).

## 2.4 Unit-Tests

Bisher wurden Methoden einer Lösung in einer `main`-Methode der Klasse selbst, in einer separaten Klasse oder manuell getestet. Daraus resultieren eine Anzahl von Probleme:

- Wird die Lösung erweitert oder verändert, dann besteht die Gefahr, dass Fehler eingebaut werden.
- Die manuelle Ausführung und Überprüfung ist sehr zeitaufwändig.

Der Lösungsansatz mit `JUnit` besteht darin, Testroutinen in einer separaten Testklasse zu erstellen. Dabei wird je Testfall eine Methode geschrieben. Diese Testmethode ruft die zu prüfende auf. Das Ergebnis des Aufrufs wird automatisch mit dem Sollwert verglichen.

---

<sup>1</sup> <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

### 2.4.1 Erstellung eines Testfalles

JUnit ist bereits in Eclipse integriert. Soll eine JUnit-Testklasse beispielsweise für die Klasse `List` erstellt werden, so genügt ein Rechtsklick mit der Maus auf die Klasse `List`. Im dann erscheinenden Menü werden `New` → `JUnit Test Case` ausgewählt.

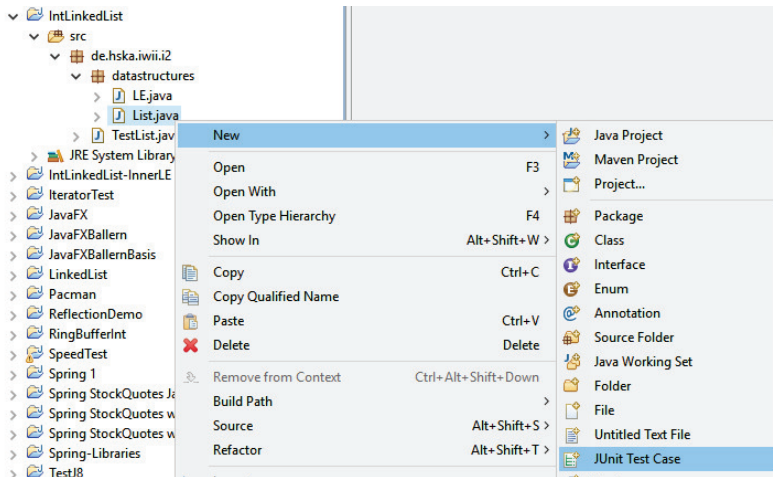


Abbildung 2.1: Testfall erstellen, Schritt 1

Es erscheint der folgende Dialog, in dem im Feld ein Name für die Testklasse vorgeschlagen wird. In der Regel sollten dieser beibehalten werden. Verwenden Sie JUnit 4 für Ihre Tests.

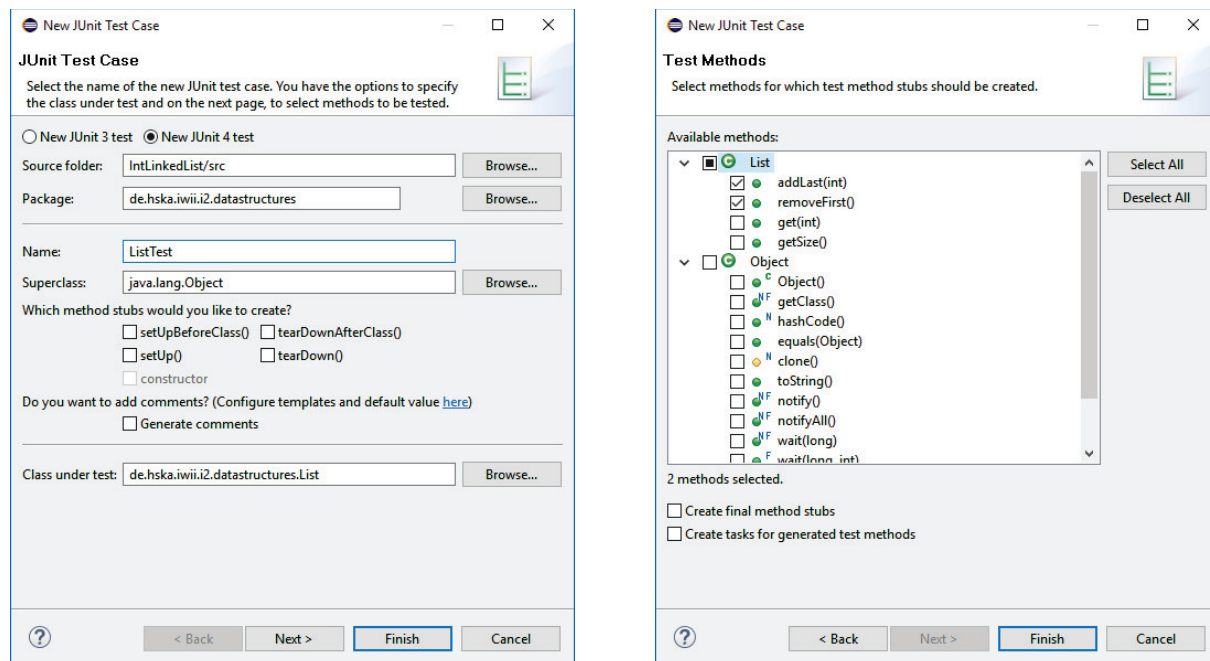
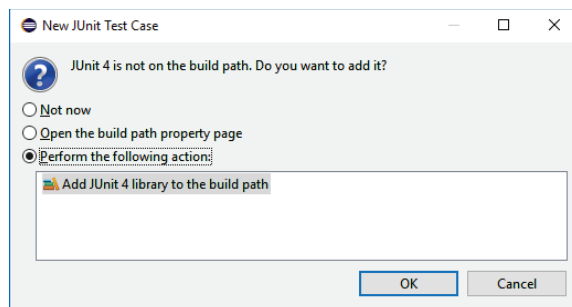


Abbildung 2.2: Testfall erstellen, Schritte 2 und 3

Mit `Next >>` erreicht man weitere Eingabemöglichkeiten. Hier werden die zu testenden Methoden ausgewählt (im Screenshot sind es `addLast` und `removeFirst`).

Die Eingabe wird mit `Finish` abgeschlossen. Eventuell erscheint noch eine Rückfrage, die Sie mit `OK` beantworten sollten.



**Abbildung 2.3:** Testfall erstellen, Schritt 4

Die neu erzeugte Klasse `ListTest` enthält jetzt die zwei noch leeren Testmethoden `testAddLast` und `testRemoveFirst`, die mit den Annotationen `@Test` versehen sind. Annotationen werden im Laufe des Semesters in der Vorlesung genauer behandelt. Die Namen aller Testmethoden beginnen aus alter Tradition mit `test`, gefolgt vom Namen der zu testenden Methode, deren erster Buchstabe groß geschrieben wird. Dieses Namensschema müssen Sie mit JUnit 4 nicht mehr beibehalten. Es stammt noch von älteren Versionen. Wichtig ist lediglich die Annotation `@Test`. Sie müssen aber darauf achten, dass Testmethoden weder Übergabe- noch Rückgabeparameter besitzen und `public` sind. Den Rumpf der Testmethode schreiben Sie natürlich selbst, weil nur Sie wissen, was Sie testen wollen. Er könnte so skizziert werden:

```
@Test
public void testAddLast() {
    // Testcode fehlt hier noch
}
```

Der Test des Ergebnisses erfolgt mit Methoden von JUnit:

- In der Testmethode wird `assertTrue(<Boole'scherAusdruck>)` aufgerufen.
- Ergibt der Ausdruck `false`, so bricht die Ausführung (später) hier ab. Der Testfall ist nicht erfüllt.
- Wenn der Ausdruck `true` wird, wird die Methode weiter ausgeführt, bis sie beendet ist. Der Testfall ist erfolgreich.

Neben `assertTrue` gibt es weitere Methoden, um Bedingungen zu prüfen. Sie finden Sie in der Dokumentation zu JUnit. So würde die Test-Methode aus dem Beispiel nach dem Einfügen der Prüfungen aussehen:

```
@Test
public void testAddLast() {
    List list = new List();
    // Liste muss leer sein.
    assertTrue(list.getSize() == 0);
    // Liste muss ein Element enthalten (es fehlt noch
    // der Test, ob das neue Element auch das letzte ist).
    list.addLast(42);
    assertTrue(list.getSize() == 1);
}
```

Wenn Sie prüfen wollen, ob eine Methode immer eine bestimmte Ausnahme auslöst, dann können Sie die Annotation `@Test` in der folgenden Form verwenden. Dabei beinhaltet der Ausdruck hinter `expected` = die Klasse der erwarteten Ausnahme.

```
@Test(expected = ArithmeticException.class)
public void divisionWithException() {
    int i = 1/0;
}
```

Zum Ausführen der Tests wird die Testklasse im Package-Explorer durch Rechtsklick mit der Maus ausgewählt. Im erscheinenden Kontextmenü muss `Run` → `JUnit Test` angeklickt werden. Dadurch werden alle Testmethoden ausgeführt. Falls alle erfolgreich sind, wird im JUnit-Tab ein grüner Balken angezeigt. Erscheint hier ein roter Balken, so ist mindestens ein Test fehlgeschlagen. Dieser Test ist unter dem Balken rot markiert und kann direkt mit einem Doppelklick ausgewählt werden.

### 2.4.2 Automatische Initialisierung je Testfall

Prinzipiell lassen sich alle Tests wie oben beschrieben durchführen. Ein Problem wird aber schnell sichtbar: Viele Testmethoden benötigen am Anfang immer einen ähnlichen Code, in dem der Shop erstellt und mit Medien gefüllt wird. Als Lösung bietet sich an, die Initialisierung zu automatisieren. Damit wird von JUnit vor der Ausführung eines Testfalls immer eine neue Instanz von `Shop` erzeugt und gefüllt, dann die nächste Testmethode aufgerufen. Die Initialisierung wird von einer beliebigen Methode durchgeführt, die Sie mit der Annotation `@Before` versehen. Folgende Vorgehensweise empfiehlt sich für das Beispiel:

- Es wird ein `private` Attribut `List list` erstellt.
- Die hinzuzufügende Initialisierungsmethode erzeugt die Liste und trägt die Referenz in das `private` Attribut ein.

Dieses ist ein guter Stil und eine Arbeitserleichterung, da unnötiges, manuelles Kopieren von Code vermieden wird.

Sollten nach Ablauf der Testmethoden noch Aufräumarbeiten erforderlich sein, so können die in einer Methode erfolgen, die mit der Annotation `@After` versehen ist. Für das Listen-Beispiel aus der Vorlesung ist das nicht notwendig. Ergebnis (ohne den Testfall `removeFirst`):

```
public class ListTest {

    private List list;

    @Before
    public void setUp() throws Exception {
        list = new List();
    }

    /**
     * Test method for
     * 'de.hska.iwii.i2.datastructures.List.addLast(int)'.
     */
    @Test
    public void testSell() {
        list.addLast(42);
    }
}
```

```
    // Tests einbauen  
}  
  
// ...  
}
```

### 2.4.3 Automatische Initialisierung je Testklasse

Die Annotationen `@Before` und `@After` werden vor bzw. nach jedem Aufruf einer Testmethode (eines Testfalls) ausgeführt. In vielen Fällen genügt es aber, je Testklasse Methoden vor dem Start und nach der Beendigung aller Testmethoden der Klasse aufzurufen. Hierzu dienen die Annotationen `@BeforeClass` und `@AfterClass`.

### 2.4.4 Weitere Hinweise zu JUnit

Mehr über JUnit ist unter <http://www.junit.org> zu finden:

- neueste Version
- Tutorial

Unter der URL <http://www.vogella.com/articles/JUnit/article.html> finden Sie eine kompakte Einführung in JUnit. Ähnliche Frameworks für automatisierte Einzeltests existieren für C#, C++ und viele andere Programmiersprachen. JUnit ist „Standard“ in der Java-Softwareentwicklung und wird bei vielen Projekten eingesetzt. Unit-Tests erlauben eine komfortable Automatisierung der Tests. Dabei ist es für den Entwickler aber nicht erkennbar, welche Teile seines Codes wirklich in einem Testlauf überprüft werden. Dazu gibt es Tools, die die Testüberdeckung ermitteln. Das soll hier aber nicht näher betrachtet werden.

# 3

## Hinweise zur Arbeit mit Gitlab

---

Sie erhalten für die Übungen bei Interesse Zugang zum Gitlab-Server der Fakultät. Wenn Sie sich dazu bitte an Ihren Übungsleiter. Die folgende Anleitung erläutert die Verwendung des Servers. Ihr persönliches Repository ist unter der URL

<https://iwi-i-gitlab-1.hs-karlsruhe.de:2443/<IZ-USER>>

erreichbar, wobei <IZ-USER> Ihr eigener IZ-Benutzername ist. Das Initial-Passwort entspricht Ihrem IZ-Benutzernamen und kann bei Anmeldung im Gitlab über den Browser verändert werden. Der Server ist nur innerhalb der Hochschule bzw. über VPN erreichbar.

### 3.1 Anlegen eines Projektes im Gitlab

Dieser Abschnitt beschreibt, wie Sie ein neues Projekt im Gitlab anlegen können. Er verwendet dazu als Beispiel die Aufgabe 1 zum Bruchrechnen. Gehen Sie im Gitlab nach Anmeldung auf die Seite

<https://iwi-i-gitlab-1.hs-karlsruhe.de:2443/dashboard/projects> und wählen Sie „New Project“ aus:



**Abbildung 3.1:** Anlegen eines neuen Projekts, Schritt 1

Danach tragen Sie den Namen des Projekts in die Maske ein und legen die Sichtbarkeit als privat fest:

### 3.3 Konfiguration von Eclipse

**New project**  
Create or import your project from popular Git services

**Project path**  
https://lwi-i-gitlab-1.hs-karlsruhe.de:2443/voma0001

**Project name**  
Bruchrechnung

Want to house several dependent projects under the same namespace? [Create a group](#)

**Import project from**

☒ GitHub ☐ Bitbucket ☒ GitLab.com ☐ Google Code ☐ Fogbugz ☐ git Repo by URL

**Project description (optional)**  
Description format

**Visibility Level (?)**  
Visibility Level

☒ Private  
Project access must be granted explicitly to each user.

☐ Internal  
The project can be cloned by any logged in user.

☐ Public  
The project can be cloned without any authentication.

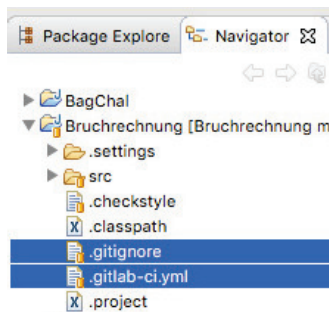
**Create project** **Cancel**

**Abbildung 3.2:** Anlegen eines neuen Projektes, Schritt 2

Erschrecken Sie nicht über die danach aufgeführten Kommandozeilen-Aufrufe. Sie können alle erforderlichen Aufrufe auch aus der IDE heraus absetzen.

## 3.2 Anlegen eines Projekts in Eclipse

Jetzt können Sie das Projekt in Eclipse so anlegen, wie Sie es schon im ersten Semester gelernt haben. Kopieren Sie die Dateien `.gitignore` und `.gitlab-ci.yml` ins Projektverzeichnis:



**Abbildung 3.3:** Projekt in Eclipse

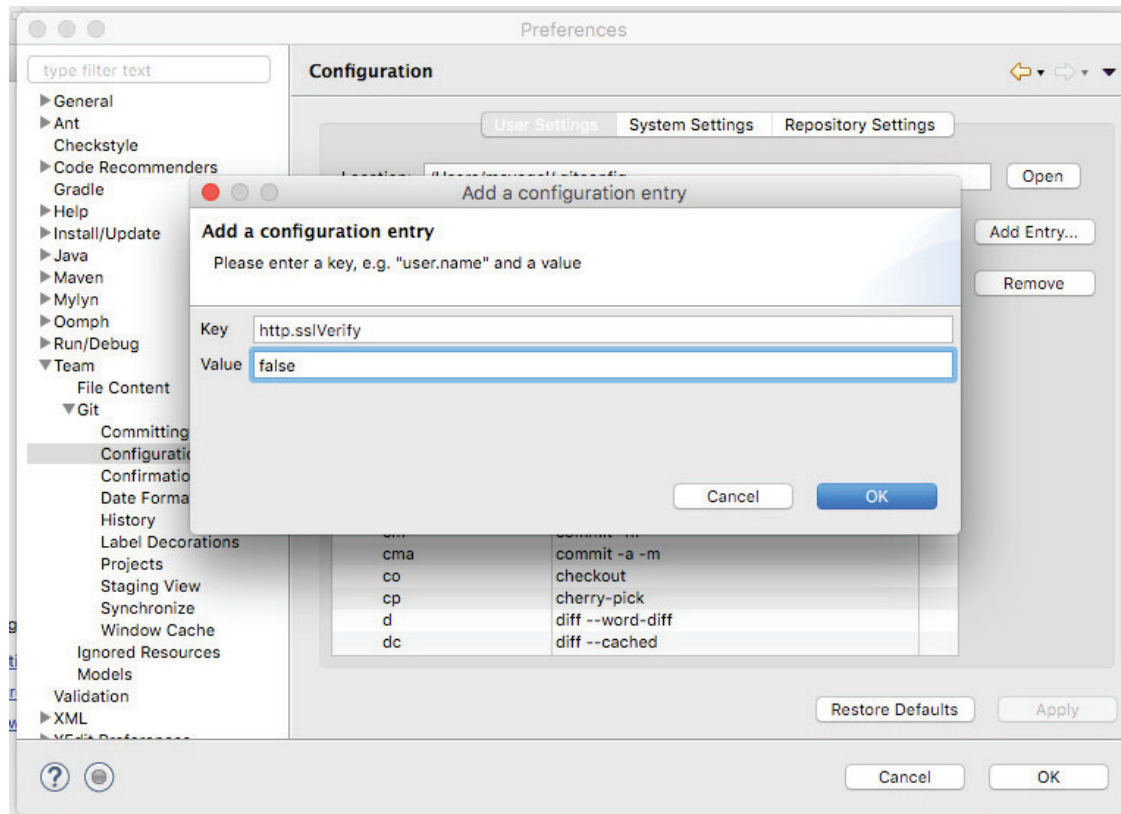
Sie erhalten diese Dateien zum Download dort im Ilias, wo Sie auch die Übungsaufgaben finden. Im Ilias sind sie ohne den führenden Punkt „.“ abgelegt. Benennen Sie nach dem Einfügen in Eclipse die Dateien so um, dass ihre Namen mit dem Punkt anfangen. Verändern Sie nicht den Inhalt der Dateien.

## 3.3 Konfiguration von Eclipse

Eclipse „kennt“ Ihr Gitlab-Repository noch nicht. In diesem Schritt teilen Sie Eclipse mit, wie es auf Ihr Repository zugreifen kann. Nehmen Sie zunächst folgende Einstellungsänderung vor. Gehen Sie im Menü „Preferences“ links im Baum auf „Team“, dann auf „git“ und „Configuration“. Erzeugen Sie mit „Add Entry“ dort einen neuen Eintrag mit folgenden Werten:

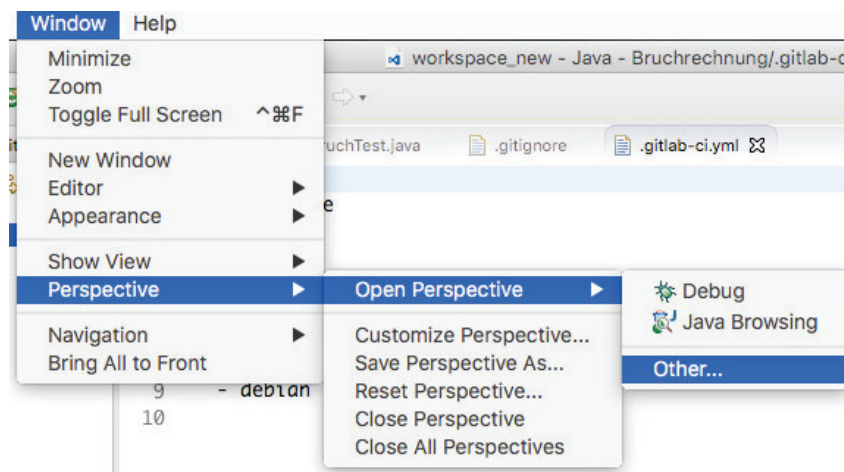


### 3.3 Konfiguration von Eclipse



**Abbildung 3.4:** Deaktivierung der SSL-Prüfung

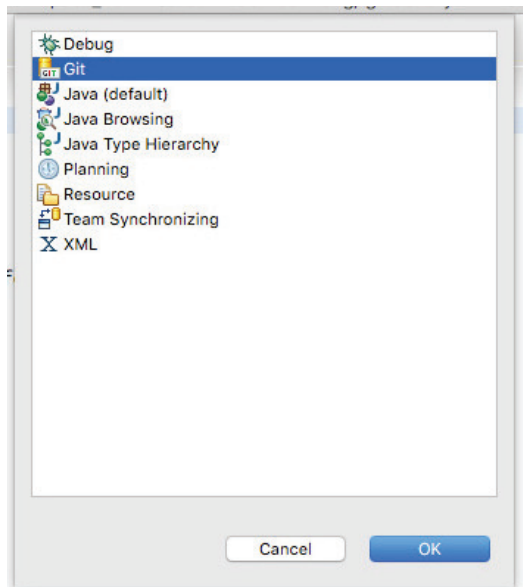
Übernehmen Sie den Wert durch Klick auf „OK“. Jetzt benötigen Sie die Git-Ansicht, um das Repository einzutragen. Gehen Sie dazu im Menü „Window“ auf „Perspective“, „Open Perspective“ und „Other“:



**Abbildung 3.5:** Wechsel in die Git-Ansicht

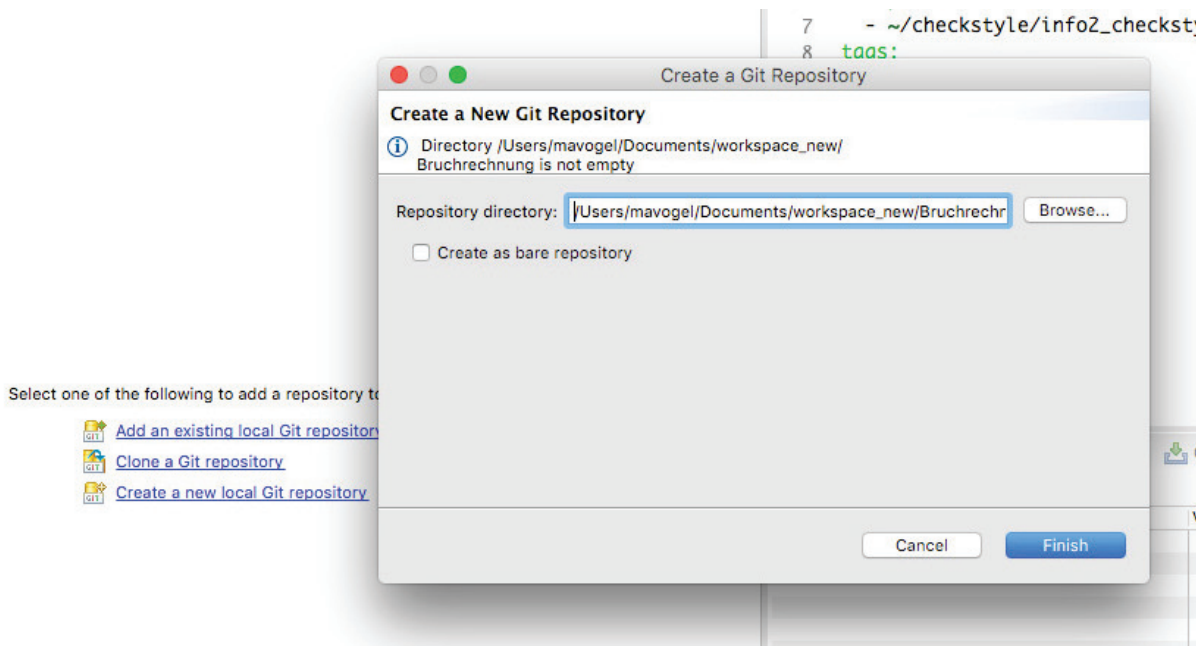
Wählen Sie „Git“ aus:





**Abbildung 3.6:** Auswahl der Git-Ansicht

Klicken Sie links in der Ansicht auf „Create a new local Git repository“ und wählen Sie im jetzt erscheinenden Dialog das Verzeichnis innerhalb Ihres Eclipse-Workspaces ein, in dem das in 3.2 erzeugte Projekt liegt.



**Abbildung 3.7:** Auswahl des Projekts

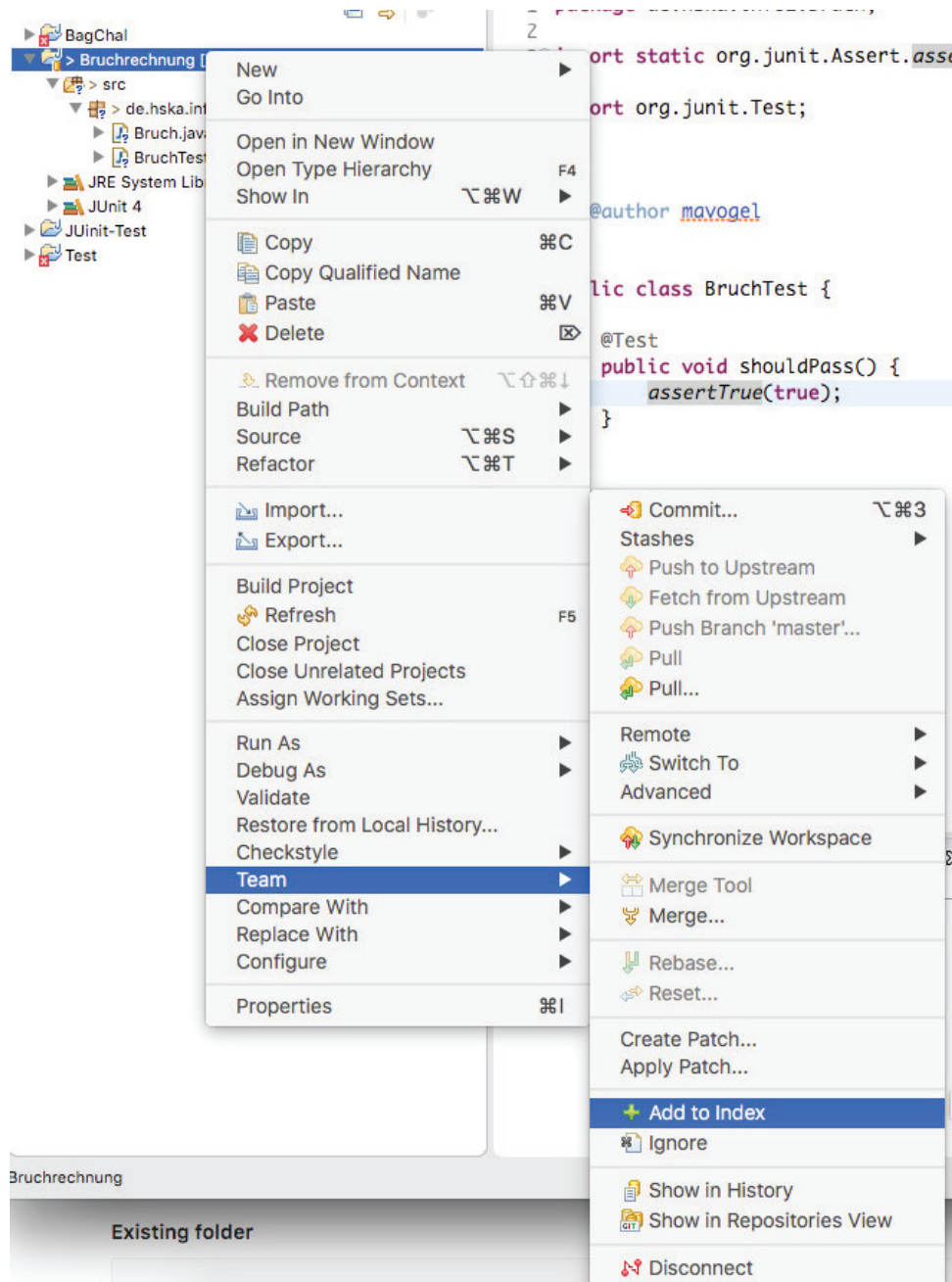
## 3.4 Erstellen der Lösung der Aufgabe

Schreiben Sie jetzt die Lösung der Aufgabe sowie den Test zur Überprüfung der Korrektheit. Wichtig dabei ist, dass Sie sich an die Java-Code-Konventionen halten und dass Ihre Test-Dateien auf `*Test.java` enden (also z.B. `BruchTest.java` für `Bruch.java`),

weil ansonsten Ihre Tests ebenfalls mit Checkstyle überprüft werden (was auch nicht schaden könnte).

### 3.5 Einfügen ins Git-Repository

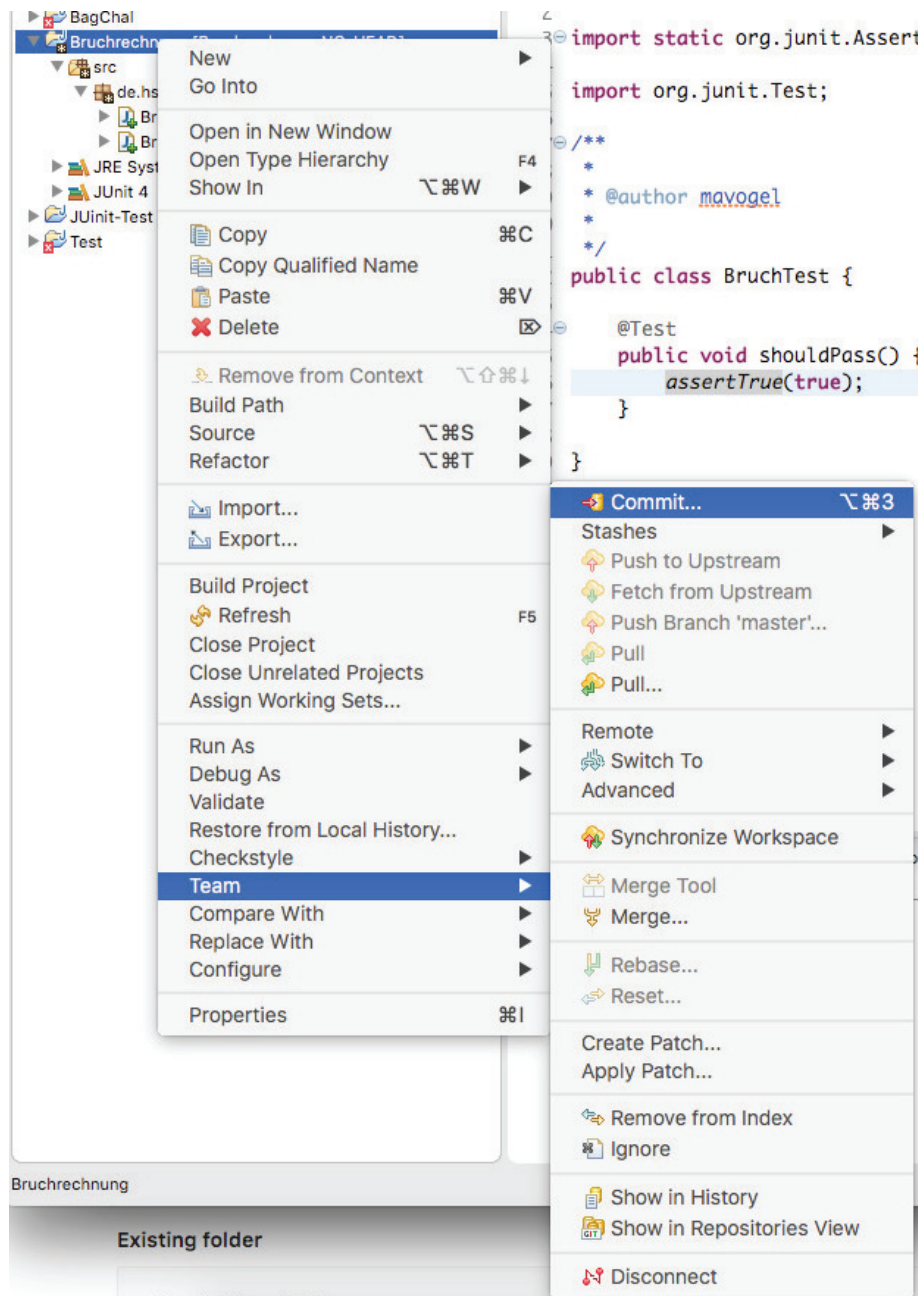
Jetzt können Sie Ihre Lösung in Ihr eigenes Git-Repository übertragen. Dieser Schritt ist natürlich auch schon dann möglich, wenn Sie Ihre Lösung noch nicht komplett fertiggestellt haben. Fügen Sie Ihre Dateien zunächst zum Index hinzu:



**Abbildung 3.8:** Hinzufügen zum Index

Übertragen Sie danach die Änderungen in die Staging-Area (Ihr lokales Repository):

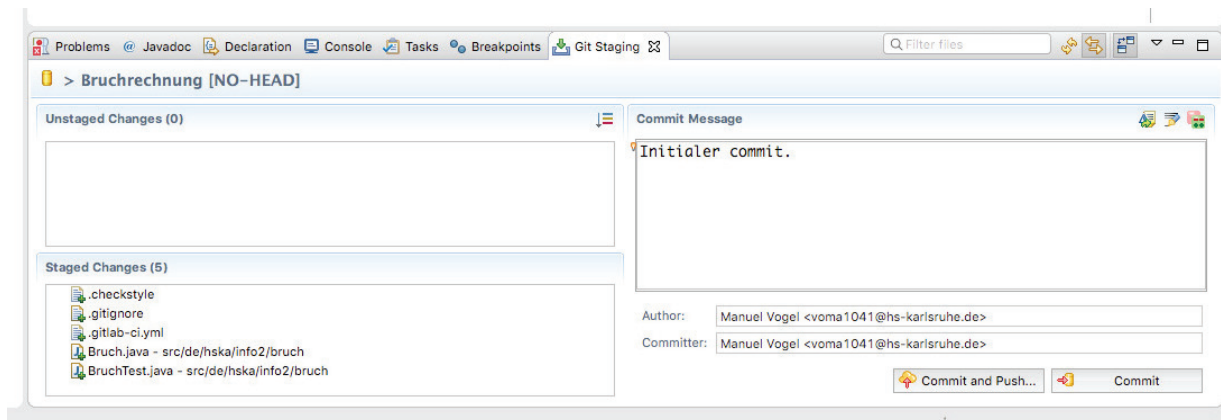
### 3.5 Einfügen ins Git-Repository



**Abbildung 3.9:** Übernahme der Änderungen in die Staging-Area

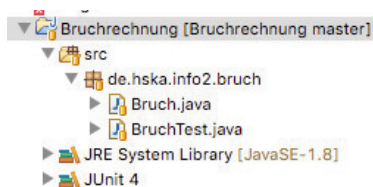
Geben Sie Ihrer Version einen sinnvollen Kommentar und klicken Sie auf „Commit“:

### 3.5 Einfügen ins Git-Repository



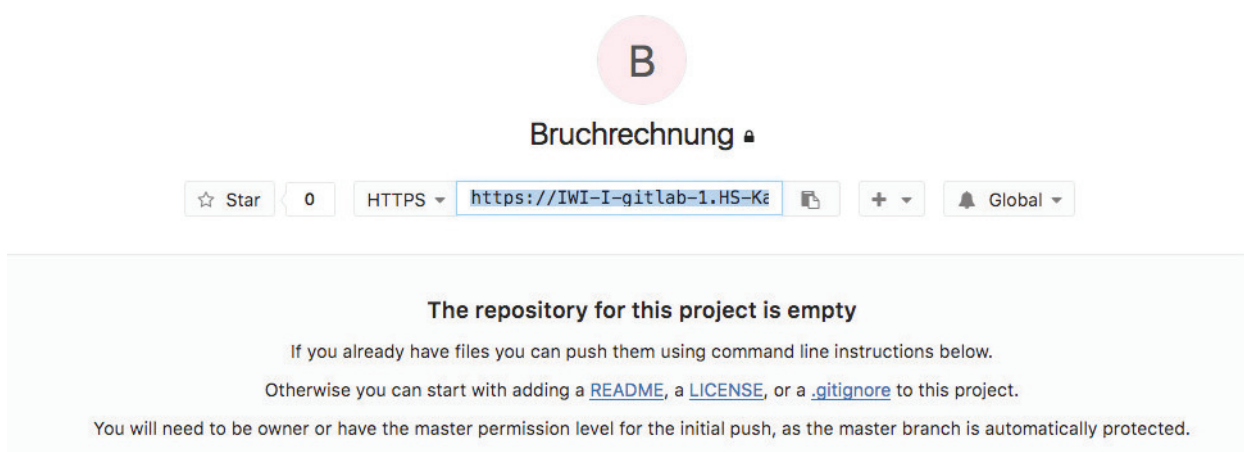
**Abbildung 3.10:** Kommentierung des Commit

Jetzt sollten alle Änderungsmarken (>) vor den Dateien im Projekt-Explorer verschwunden sein:



**Abbildung 3.11:** Quelltextdateien sind aktuell

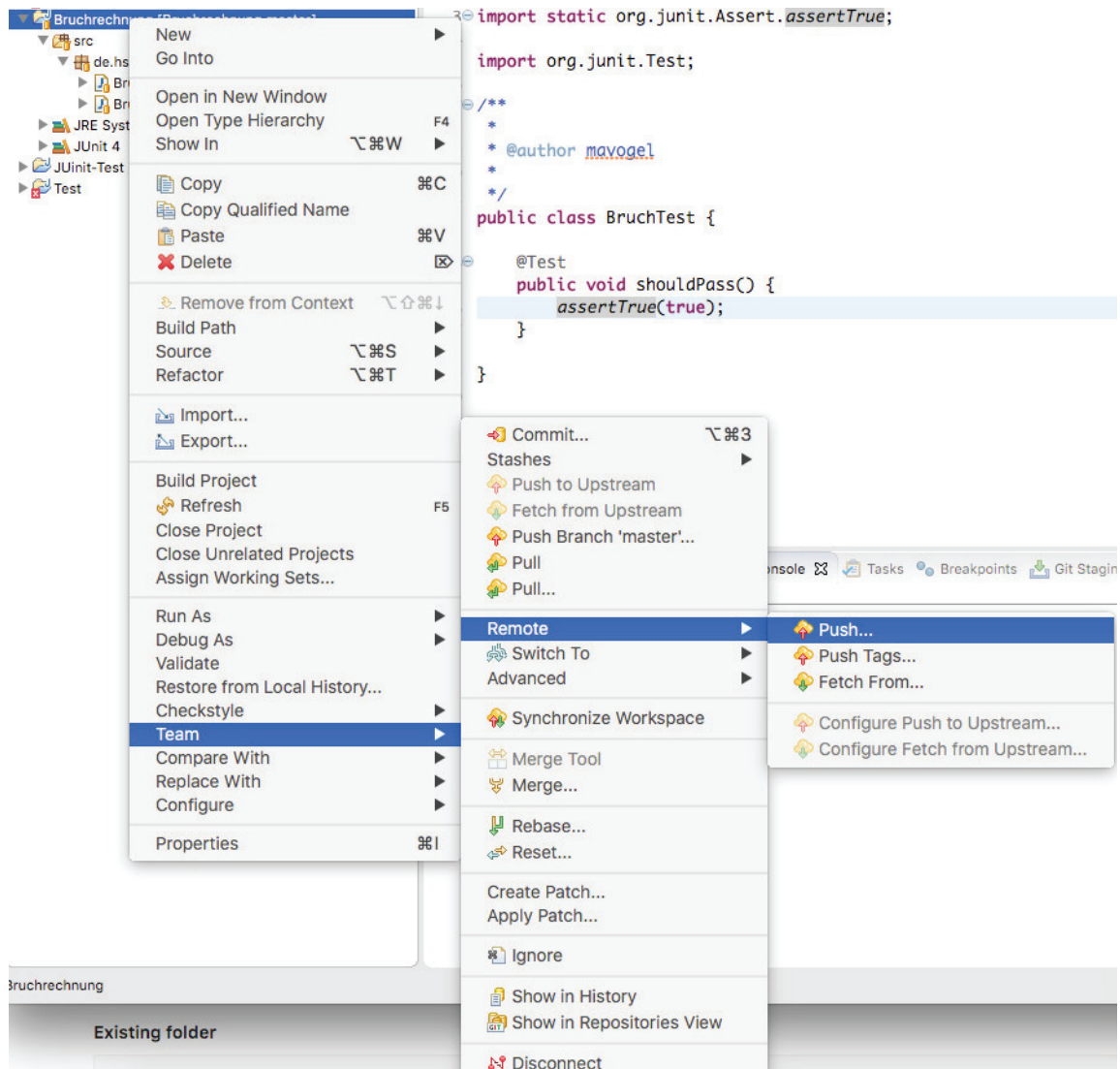
Die Änderungen befinden sich nach dem „Commit“ in Ihrem lokalen Repository und können von dort aus in Ihr Repository auf dem Gitlab-Server übertragen werden. Dazu benötigen Sie die eindeutige URL Ihres Repositories, das Sie von der Web-Seite des Gitlab im eingeloggten Zustand kopieren können. Achten Sie darauf, dass die URL die Endung „.git“ behält:



**Abbildung 3.12:** Ermittlung der Repository-URL

Übertragen Sie die Änderungen aus Ihrem lokalen Repository auf den Server:

### 3.5 Einfügen ins Git-Repository



**Abbildung 3.13:** Pushen des lokalen Repositories

Die IDE fragt Sie nach Ihren Zugangsdaten zum Gitlab:

### 3.5 Einfügen ins Git-Repository

Push to Another Repository

**Destination Git Repository**

Enter the location of the destination repository.

Location

URI:  

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

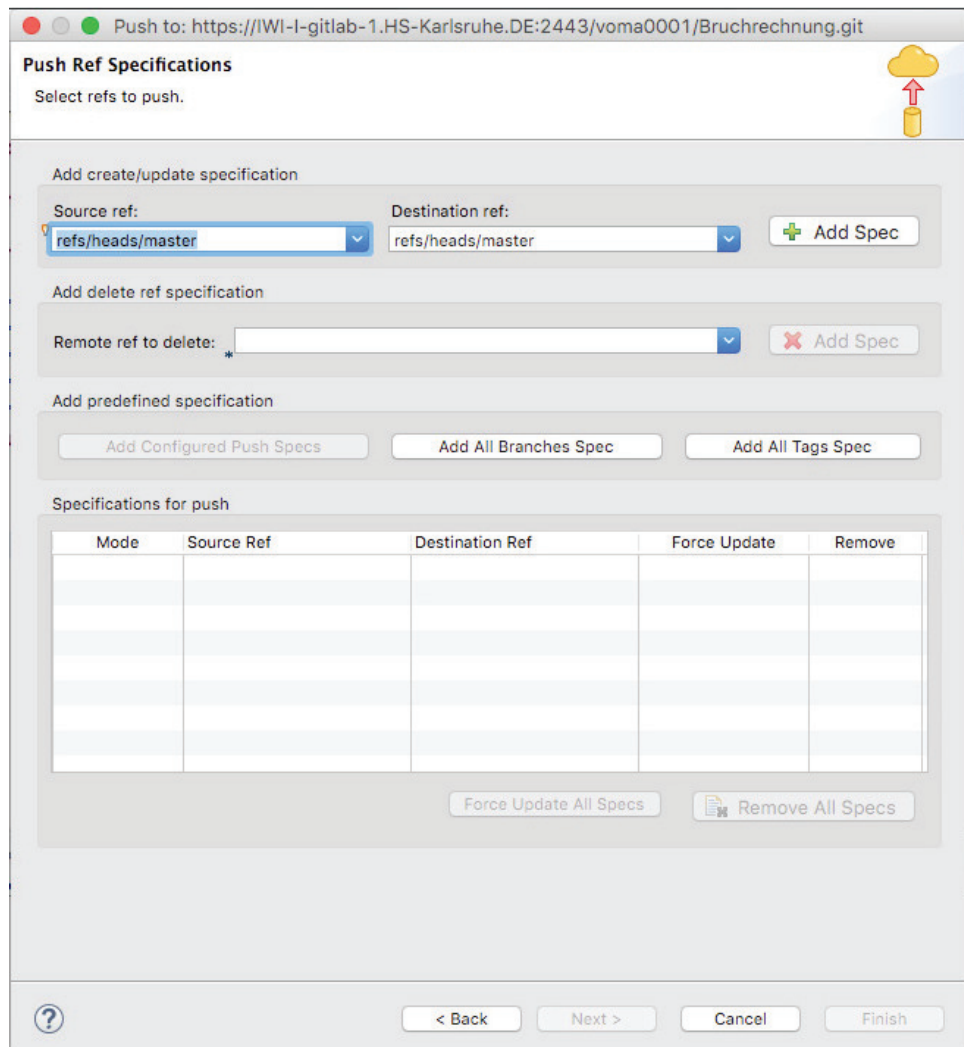
☐ Store in Secure Store

**Abbildung 3.14:** Eintragen der Zugangsdaten zum Gitlab

Auf der Folgeseite können Sie die Einstellungen beibehalten:

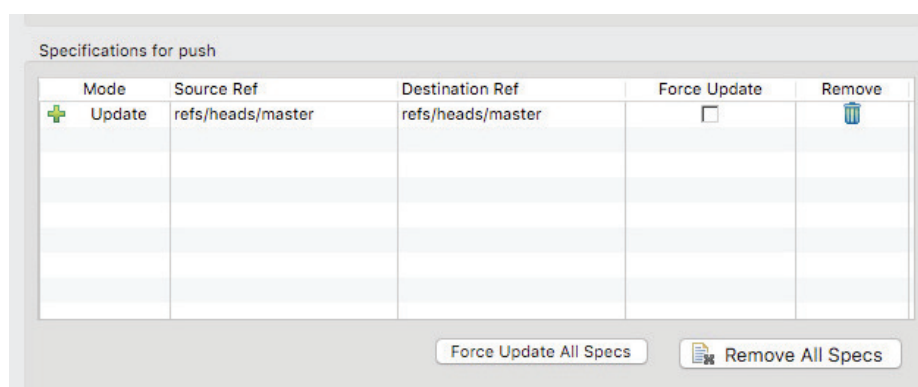


### 3.5 Einfügen ins Git-Repository



**Abbildung 3.15:** Abbildung des lokalen Branches auf das im Gitlab

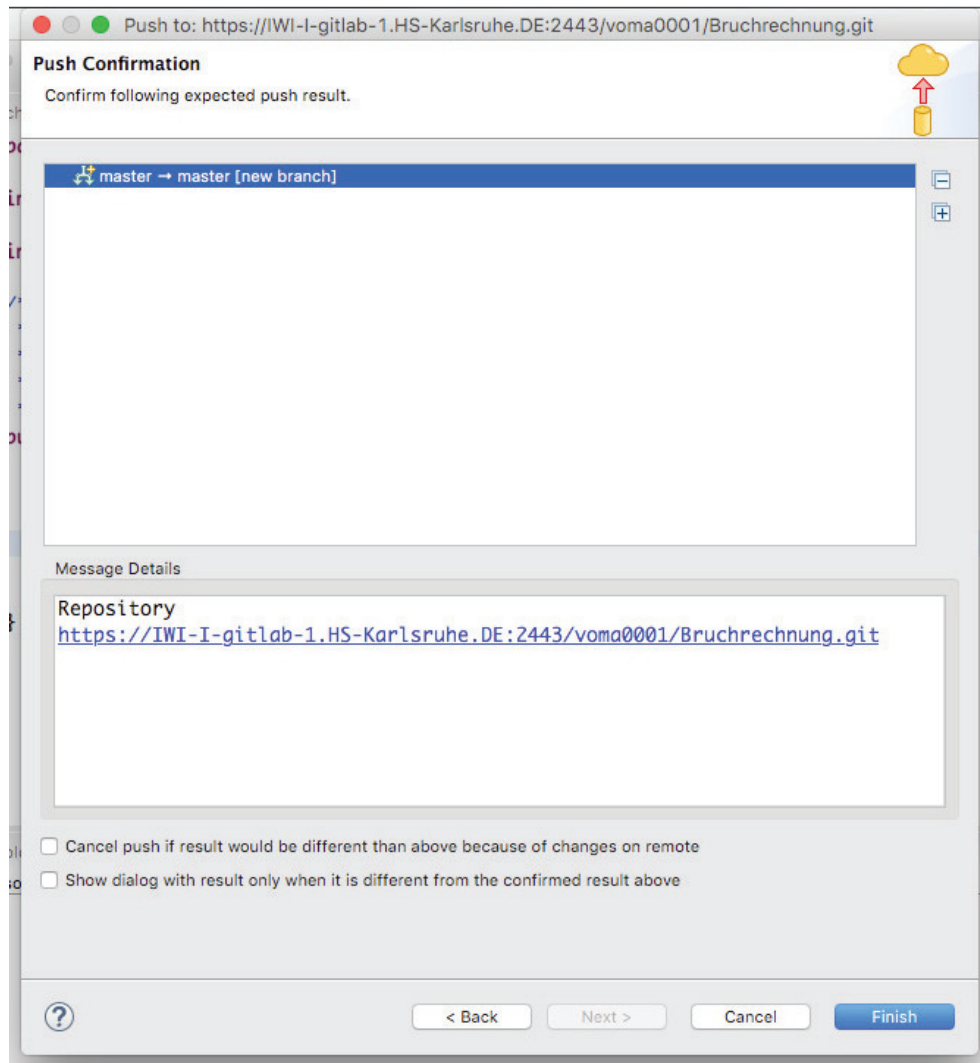
Dann haben Sie das folgende Ergebnis:



**Abbildung 3.16:** Ergebnis nach dem „Pushen“ in Eclipse (1)

### 3.5 Einfügen ins Git-Repository

---

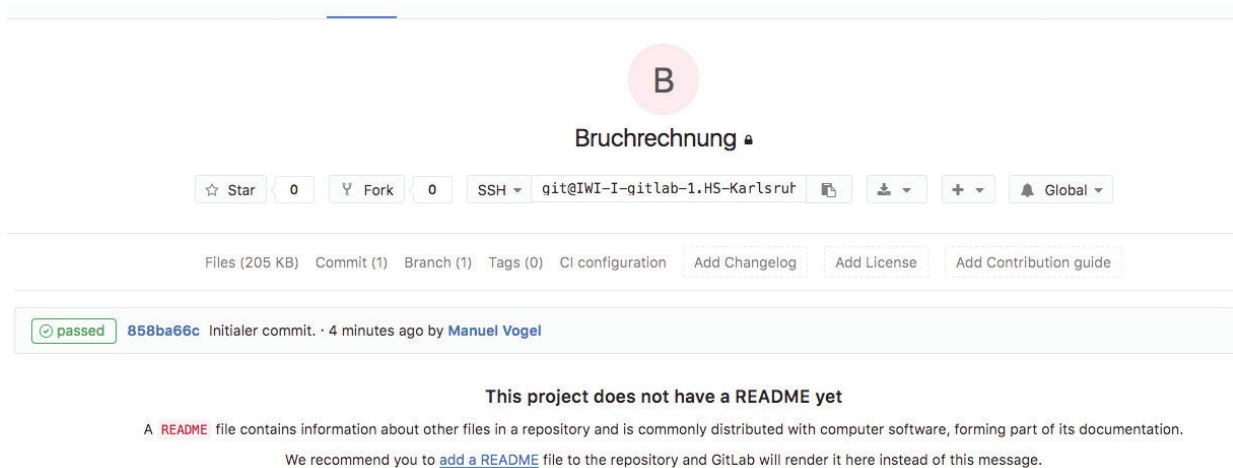


**Abbildung 3.17:** Ergebnis nach dem „Pushen“ in Eclipse (2)

Und im Gitlab sehen Sie folgendes:



### 3.5 Einfügen ins Git-Repository



**Abbildung 3.18:** Ergebnis nach dem „Pushen“ in der Web-Anzeige des Gitlab

Sollte die Prüfung durch Checkstyle fehlgeschlagen sein, dann sehen Sie ein rotes 'failed' anstatt eines grünen 'passed' in der grau hinterlegten Zeile.

## Literaturverzeichnis

---

- [Ull02] Ullmann C.: Java ist auch eine Insel, Galileo Computing (kostenlos als HTML-Dokumente unter <http://www.tutego.de/javabuch/>)
- [Ull03] Ullmann C.: Java ist nicht nur eine Insel, Galileo Computing
- [RaScWi11] Ratz D., Scheffler J., Seese D., Wiesenberger J.: Grundkurs Programmieren in Java. Hanser Fachbuchverlag, 2011
- [ORA01] Oracle: Java-Tutorial,  
<http://download.oracle.com/javase/tutorial/>
- [ORA02] Oracle: JDK-Referenz  
<http://www.oracle.com/technetwork/java/javase/documentation/index.html>
- [ORA03] Oracle: Code Conventions for the Java™ Programming Language  
<http://www.oracle.com/technetwork/java/codeconv-138413.html>

# 5

## Stichwortverzeichnis

---

### Symbole

PATH ..... 5

### C

Checkstyle ..... 7  
    Installation ..... 7  
    Konfiguration ..... 7  
Code-Konventionen ..... 7

### E

Eclipse ..... 5, 6  
    JUnit ..... 10

### F

FindBugs ..... 8  
    Installation ..... 8  
    Konfiguration ..... 8  
    Pool-Installation ..... 9  
    Verwendung ..... 8

### I

IDE ..... 5  
    Eclipse ..... 5  
    IntelliJ IDEA ..... 6  
    JDeveloper ..... 6  
    Netbeans ..... 5  
IntelliJ IDEA ..... 6

### J

Java Development Kit ..... 5  
Java Runtime Environment ..... 5  
Javadoc ..... 6  
JDeveloper ..... 6  
JDK ..... 5  
Jetbrains ..... 6  
JRE ..... 5  
JUnit ..... 9  
JUnit ..... 9, 10  
    @AfterClass ..... 13  
    @After ..... 12  
    @BeforeClass ..... 13  
    @Before ..... 12  
    @Test ..... 11  
    Ausführung ..... 12  
    Automatische Initialisierung je Testfall ..... 12  
    Automatische Initialisierung je Testklasse ..... 13  
    Eclipse ..... 10  
    Testüberdeckung ..... 13  
    Testfall ..... 10  
    Testklasse ..... 10

Testmethoden ..... 11

### N

Netbeans ..... 5

### O

Oracle ..... 5, 6

### T

Testfall ..... 10

### U

Unit-Tests ..... 9