Karlsruher Institut für Technologie

Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Lehrstuhl Prof. Walter F. Tichy

E-Mail: swt1@ipd.kit.edu

Ilias: https://ilias.studium.kit.edu/goto.php?target=crs_1096553&client_id=produktiv

Web: https://ps.ipd.kit.edu

LEZ: https://lez.ipd.kit.edu

SWT1: Übungsblatt 1

Altsoftware vorbereiten, Programmierumgebung, Testen

Ausgabe: Mittwoch, 22. April 2020
Abgabe: Mittwoch, 6. Mai 2020, 12:00 Uhr

Bitte melden Sie sich an der LEZ mindestens einmal an: https://lez.ipd.kit.edu/

Aufgabe 1: Altsoftware vorbereiten (8 Punkte)

Ihre Firma Pear Corp. hat vor kurzem das erfolgversprechende Startup JIS Inc. gekauft, dessen leider völlig unbekanntes Flaggschiff „Java Mass JPEG Resizer Tool“ (JMJRST) zu einer umfassenden Plattform zum Vertrieb von Archivbildern (engl. *stock images*) unter dem Namen iMage ausgebaut werden soll. Da in Ihrer Firma Maven eingesetzt wird, müssen die Quelldateien von JMJRST anders organisiert werden, damit später die Erweiterungen gut integriert werden können. Ihre Aufgabe ist es, ein neues Projekt aufzusetzen und JMJRST zu migrieren. Dazu setzen Sie zuerst ein neues Maven-Projekt auf und fügen dann die Quelldateien aus dem Altprojekt der JIS Inc. in Ihr Git-Depot ein.
Nach Ihren Anpassungen kann

1. Eclipse (oder die IDE Ihrer Wahl) die Software übersetzen, ausführen und Fehlersuche durchführen,
2. Maven die Software übersetzen, Tests ausführen, verpacken etc.,
3. Git die Änderungen am Quelltext protokollieren und
4. CheckStyle Qualitätstests durchführen.

Buchen Sie (sobald das Git-Depot besteht) in sinnvollen Abschnitten ein – mindestens für jeden Aufgabenteil einmal oder besser für jeden Schritt einer Teilaufgabe. Geben Sie **immer** aussagekräftige Logbucheinträge an. Behalten Sie diese Praxis für alle zukünftigen Programmieraufgaben bei; die Logbucheinträge und die Granularität der Einbuchungen werden bewertet. Befolgen Sie dabei die folgenden Schritte:

a) Legen Sie einen neuen Eclipse-Workspace an. Diesen Workspace werden wir für alle Aufgaben dieses Semesters verwenden. Legen Sie über die Kommandozeile direkt innerhalb des Workspace ein neues Git-Depot an.
b) Laden Sie das Archiv `gitignore.zip` aus ILIAS herunter, speichern Sie die enthaltene Datei als `.gitignore` (mit führendem Punkt) in Ihrem Workspace und fügen Sie sie zur Versionskontrolle hinzu.
c) Legen Sie mit Eclipse ein neues Maven-Projekt namens `iMage` an. Verwenden Sie keinen fertigen „Archetype“ sondern „simple project“. Wählen Sie als Packaging `pom`, als GroupID `swt1` und als ArtifactID `iMage`. Das „Parent Project“ hat die GroupID `edu.kit.ipd.swt1.ss2020` und die ArtifactID `uebungsparent` und die Version `0.0.1-SNAPSHOT`.

d) Nach dem Anlegen des Projekts kann Maven die `pom.xml` des Elter-Projects (`parent pom.xml`) nicht ermitteln. Öffnen Sie die erzeugte `pom.xml` (ihres Projekts) und tragen Sie vor die letzte Zeile die folgenden Informationen ein:

```
<repositories>
  <repository>
    <id>ipdNexus</id>
    <url>http://bob.ipd.kit.edu/nexus/repository/maven-public-all/</url>
  </repository>
</repositories>
```

Kopieren Sie zudem die Datei `src/assembly/src.xml` aus dem Testprojekt vom Vorbereitungsblatt an die entsprechende Stelle in Ihrem neuen Projekt.

Nach diesem Schritt sollten Sie `mvn package` erfolgreich in `iMage` ausführen können.

e) Legen Sie mit Eclipse ein neues Maven-Modul namens `jmjrst.main` an; öffnen Sie dazu die pom.xml im pom-Editor und wählen Sie unter „Overview→Modules“ „Create“ aus. Wählen Sie auch hier „simple project“ aus. Belassen Sie die Standardeinstellungen unverändert. Zu ArtifactID und GroupID siehe Anmerkungen am Ende des Übungsblatts.

f) Laden Sie das Archiv `jmjrst.zip` aus dem ILIAS herunter und entpacken Sie es.

Verschieben Sie die Dateien `LICENSE` und `options.properties` sowie den Inhalt von `src` und das Verzeichnis `templates` an ihren Maven-konformen Ort. Trennen Sie Java-Dateien von Nicht-Java-Dateien: Suchen Sie alle Nicht-Java-Dateien (die im ursprünglichen Verzeichnisbaum von JMJRST mit den Quelltextdateien gemischt abgelegt sind) und verschieben Sie diese in das entsprechende Unterverzeichnis von `src/main/resources`. Beachten Sie dabei, dass Sie ggf. analoge Paketverzeichnisstrukturen anlegen müssen. (Das Verzeichnis `templates` dürfen bzw. müssen Sie ins Stammverzeichnis von `jmjrst.main` verschieben. Sie brauchen dessen Benutzung seitens JMJRST nicht zu korrigieren; es ist in Ordnung, wenn `templates` nicht Maven-artig funktioniert.)

Bearbeiten Sie diesen Aufgabenteil nicht in Eclipse, sondern mit einem Datei-Manager Ihrer Wahl. In diesem Schritt müssen Sie keinen Quelltext ändern. Wenn Sie Quelltext ändern müssen, um „Fehler“ (z.B. der Paketstruktur) zu beseitigen, haben Sie etwas falsch gemacht. Siehe hierzu auch:
http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

Notiz: Der Quelltext kann unter Umständen nicht kompiliert werden, wenn Sie jetzt `mvn package` ausführen. Siehe Aufgabenteil g).

g) Aktualisieren Sie die Ansicht in Eclipse. Sie sollten nun die Quelldateien von JMJRST sehen können. Darunter sind zwei Dateien, die Warnungen oder Fehler hervorrufen: Sie verwenden die Klasse `ImageFormatException`, welche bei Oracle zugriffsgeschützt und im OpenJDK nicht vorhanden ist. Diese Warnungen oder Fehler beheben Sie nun, indem Sie den Patch einspielen, der in ILIAS zum Herunterladen bereitsteht (`patch.zip`). (In Eclipse können Sie einen Patch einspielen, indem Sie ihn mit der rechten Maustaste anklicken und „Team → Apply Patch“ auswählen.) Nach diesem Schritt können Sie das Programm in Eclipse starten und ausführen. (Wenn bei Ihnen nur eine Warnung angezeigt wird, müssen Sie diesen Schritt trotzdem durchführen, damit JMJRST auf Rechnern ohne Oracle-JDK bzw. -JRE übersetzt und ausgeführt werden kann, z.B. auf der LEZ.)

h) Kopieren Sie die folgenden Dateien aus dem Testprojekt vom Vorbereitungsblatt in das Projekt `jmjrst.main`:
   1. `src/assembly/src.xml`
   2. `src/test/resources/checkstyle_swt1.xml`
   3. `*.launch`

   `src.xml` steuert die Erzeugung der ZIP-Datei zur Abgabe. Editieren sie `src.xml` so, dass `.gitignore` und die oben genannten Dateien (die nicht unterhalb von `src/` liegen) ebenfalls in die ZIP-Datei gepackt werden. Ebenso soll die Lizenzdatei im ZIP enthalten sein.

i) Stellen Sie sicher, dass beim Aufruf von `mvn package` die richtige Hauptklasse im resultierenden Java-Manifest hinterlegt wird. (Dadurch wird das JAR, das Maven erzeugt, ausführbar.) Setzen Sie außerdem für das `maven-jar-plugin` die Einstellung `addClasspath` auf `true`.

j) Sorgen Sie dafür, dass das Verzeichnis `target` nicht in der Versionskontrolle von Git geführt wird. (In der Ansicht „Navigation“ Rechtsklick auf `target` → „Team“ → „Ignore“. Dadurch entsteht ein neuer Eintrag in der Datei `.gitignore`, sofern es diesen Eintrag nicht ohnehin schon gab.)

k) Konfigurieren Sie CheckStyle in Eclipse so, dass es die im Projekt hinterlegte Konfigurationsdatei `src/test/resources/checkstyle_swt1.xml` verwendet.
   Führen Sie CheckStyle in Eclipse aus. Berichtigen Sie drei beliebige von CheckStyle gemeldete Fehler (einzelne Fehler, nicht ganze Kategorien!) im Projekt und dokumentieren Sie diesen Vorgang durch eine Einbuchung in Ihrem Git-Depot. Beschreiben Sie im Logbucheintrag, welche Fehler Sie wo korrigiert haben.

l) Erzeugen Sie mit dem Maven-Aufruf `mvn package` auf `iMage` die nötigen ZIP-Dateien für die Abgabe. Es entstehen ZIP-Dateien in `iMage/target` und in `iMage/jmjrst.main/target` mit den Quelltexten Ihres angepassten Projekts. Übertragen Sie die ZIP-Datei aus `iMage/jmjrst.main/target` in die LEZ. Übertragen Sie *nicht* die JAR-Datei, sondern die ZIP-Datei.

Prüfen Sie vor der Abgabe, ob die folgenden Dinge mit Ihrem iMage-Projekt funktionieren:

- `mvn package` läuft ohne Fehler durch.
- Sie können das Programm JMJRST mit dem Befehl `java -jar <Dateiname>.jar` starten. Alternativ: Sie können JMJRST mit einem Doppelklick auf die erzeugte JAR-Datei starten.
- Die ZIP-Datei mit den Quelltextdateien enthält die Datei `docs/changelog.html`.

Aufgabe 2: Modultests (7 Punkte)

Ihre Projektleiterin hat dem Vorstand vollmundig zugesichert, dass JMJRST höchsten Qualitätsstandards genügt. Sie hat jedoch keine Ahnung, wie die Qualität des Quelltextes festgestellt werden und ggf. verbessert werden kann. Sie schlagen in einer Projektsitzung vor, automatisch laufende Modultests zu verwenden. Ihre Projektleiterin ist über diesen Vorschlag so erfreut, dass sie Sie sofort mit der Umsetzung dieses Vorhabens betraut. Sie beginnen damit, Testfälle für die Methode `copyFile(...)` der Klasse `org.jis.generator.LayoutGalerie` zu erstellen. Sie gehen dabei wie folgt vor:

a) Erstellen Sie in Ihrem JMJRST-Projekt einen neuen Entwicklungszweig (*branch*) namens **test** für die Testerstellung und ggf. anfallende Korrekturen; in diesem Entwicklungszweig dokumentieren Sie Ihre Schritte durch Einbuchungen in das Versionskontrollsystem.

b) Fügen Sie Ihrem Maven-Projekt die Abhängigkeit für JUnit hinzu (die Versionsangabe wird aus dem parent-pom.xml übernommen, d.h. Sie sollen keine Version angeben) und erstellen Sie die benötigten Verzeichnisse, z.B. `src/test/java/...`

c) Kopieren Sie die Testklasse `LayoutGalerieTest` aus ILIAS an den Maven-konformen Platz innerhalb Ihres Projektes. Diese Klasse enthält bereits einen Testfall, der das erfolgreiche Kopieren einer Textdatei mit zufälligem Inhalt testet. Bringen Sie den Testfall mittels JUnit zur Ausführung. Prüfen Sie, inwiefern der Testfall sich an die aus der Vorlesung bekannten Konventionen der Rahmenarchitektur von JUnit hält. Beheben Sie etwaige Probleme, indem Sie die Implementierung des Testfalls ausbessern. Ändern Sie an dieser Stelle nicht die Implementierung der getesteten Methode `copyFile(...)` aus der Klasse `LayoutGalerie`!

d) Schreiben Sie zwei neue Testfälle für `copyFile(...)`, in denen Sie das Verhalten beim Kopieren von Verzeichnissen sowie von nichtexistierenden Quelldateien testen. In beiden Fällen soll eine `FileNotFoundException` ausgelöst werden.

e) Schreiben Sie einen zusätzlichen Testfall, in dem die Zieldatei vor dem Kopiervorgang bereits existiert. In diesem Fall soll die Zieldatei durch die zu kopierende Quelldatei ersetzt werden. Ihr Testfall sollte zunächst fehlschlagen, da stattdessen der Inhalt der Quelldatei an den Inhalt der Zieldatei angehängt wird. Beheben Sie dieses Problem, indem sie den Quelltext der Methode `copyFile(...)` sinnvoll anpassen. Daraufhin sollte Ihr Testfall grün werden.

f) Schreiben Sie zwei weitere Testfälle, bei denen der Lesezugriff auf eine zu kopierende Quelldatei bzw. der Schreibzugriff auf eine bereits existierende Zieldatei gesperrt ist. In beiden Fällen soll eine `IOException` ausgelöst werden. Nutzen Sie Instanzen der Klasse `java.nio.channel.FileLock`, um den Zugriff auf die entsprechenden Dateien zu sperren. Nutzen Sie alternativ die Methoden `setReadable(...)` bzw. `setWriteable(...)` der Klasse `File`, falls sich Dateien auf Ihrem Betriebssystem nicht mittels `FileLock` sperren lassen. Beachten Sie, dass die Methode `copyFile(...)` nur dann Schreibzugriff auf die Zieldatei beantragt, wenn Text in der Quelldatei enthalten ist. Je nach Implementierung können Ihre Testfälle zunächst fehlschlagen. Beheben Sie ggf. dieses Problem, indem Sie den Quelltext der Methode `copyFile(...)` sinnvoll erweitern. In jedem Fall sollten Ihre Testfälle abschließend grün sein.

g) Erstellen Sie eine Datei mit einem Patch für die Anpassungen aus e), der ausschließlich die Änderungen an der Klasse `LayoutGalerie` enthält. Speichern Sie den Patch unter `src/test/resources/LayoutGalerie.patch` ab.

h) Wechseln Sie zurück zum **master**-Zweig und verschmelzen Sie den neu angelegten Entwicklungszweig **test** mit dem **master**-Zweig.

Prüfen Sie vor der Abgabe, ob die folgenden Dinge mit Ihrem JMJRST-Projekt funktionieren:

- `mvn package` läuft ohne Fehler durch und die Tests werden ausgeführt.
- Die ZIP-Datei mit den Quelltextdateien enthält die Datei `docs/changelog.html`.
- Die ZIP-Datei enthält alle Quelltexte und Testfälle, die Sie zur Aufgabe erstellt oder verändert haben.

Aufgabe 3: Testüberdeckung (4 Punkte + 3 Bonuspunkte)

Kopieren Sie die Testklasse `GeneratorTest` aus ILIAS an ihren Maven-konformen Ort in Ihrem Projekt. Diese Klasse enthält Modultests für die Klasse `org.jis.generator.Generator`. Speichern Sie außerdem die für die Testklasse benötigte `image.jpg` an Ihrem Maven-konformen Ort. Nun sollten die Testfälle in `GeneratorTest` ohne Anpassungen „grün“ werden. Fügen Sie der Testklasse (funktionierende, sinnvolle, „grüne“) Modultests hinzu, so dass Sie mindestens 35% Testüberdeckung für die Klasse (nicht für die Datei) `Generator` erreichen. Schreiben Sie mindestens vier sinnvolle Testfälle und belassen Sie den Quelltext von

JMJRST unverändert. Sollte ein „korrekter Test“ fehlschlagen, weil die Implementierung von JMJRST ein falsches, nicht zu erwartendes Ergebnis produziert, so annotieren Sie den Test vor der Abgabe mit `@Ignore`. (Fehlschlagende Testfälle verhindern das Verpacken der Quelldateien mit `mvn package`, daher müssen sie ignoriert werden.)

Für Testfälle ohne Erwartung (d.h. ohne den Aufruf von Methoden der Klasse `Assert` bzw. ohne Annotieren des Testfalls) erhalten Sie keine Punkte. Schreiben Sie ausschließlich Tests, die vollständig automatisch ablaufen und geprüft werden.

Bonusaufgabe: Verwenden Sie die Bibliothek Mockito (mockito.org) um die Testüberdeckung auf mindestens 50% zu erhöhen (verwenden Sie statt `null` eine Nachahmung der `Main`-Klasse).

Prüfen Sie vor der Abgabe, ob die folgenden Dinge mit Ihrem JMJRST-Projekt funktionieren:

- `mvn package` läuft ohne Fehler durch und die Tests werden ausgeführt (falls Sie auch Aufgabe 2 bearbeitet haben, dann werden die dazu gehörenden Tests ebenfalls ausgeführt).
- Die ZIP-Datei mit den Quelltextdateien enthält die Datei `docs/changelog.html`.
- Die ZIP-Datei enthält alle Quelltexte und Testfälle, die Sie zur Aufgabe erstellt oder verändert haben.

Hinweis zu den Werkzeugen

Git ist das in den Übungsaufgaben und in den Tutorien verwendete Werkzeug zur Versionskontrolle. Eclipse ist die in der Vorlesung und in den Tutorien verwendete Programmierumgebung. Sie können auch eine andere Programmierumgebung einsetzen, wenn Sie damit ausreichend vertraut sind und die gestellten Aufgaben genauso bearbeiten können. Es werden keine Aufgaben ausgegeben, deren Lösungen Eclipse-spezifische spezielle Fähigkeiten benötigen. Sie müssen „nur" in einer modernen Entwicklungsumgebung programmieren und Versionskontrolle einsetzen können.
Verwenden Sie bitte für alle Aufgaben Java 14.

Hinweis zu den Programmieraufgaben

Sofern nicht anders angegeben, sind die Aufgaben mit Java zu lösen. Die Einbuchungen und ihre Kommentare in Git werden bewertet (Existenz, sinnvolle Einbuchungshäufigkeit, -zeitpunkte und -dateien, sprechende Kommentare).
Konfigurieren Sie Maven mittels der Datei `pom.xml` **immer** wie folgt, um konsistente Projekte zu erhalten:

1. `ArtifactID` ist Ihre Matrikelnummer.
2. `GroupID` ist „swt1.ub<Übungsblatt-Nr.>.a<Aufgaben-Nr.>", also z. B. „swt1.ub0.a2" für Aufgabe 2 des Vorbereitungsblatts oder „swt1.ub3.a1" für Aufgabe 1 des dritten Übungsblattes.
3. Ändern Sie keine Einstellungen, die im XML-Dokument mit Kommentaren von uns versehen wurden (bspw. den Bereich DependencyManagement). Änderungen an diesen Bereichen können die automatische Analyse Ihrer Programme verhindern – in so einem Fall müssen Sie mit Punktabzug rechnen.
4. Wenn Sie Quelltexte abgeben, erzeugen Sie die Abgabedatei immer mit dem Befehl `mvn package`; laden Sie nicht die Binärfassung des Projekts hoch (*.jar), sondern die ZIP-Datei mit den Programmquellen. **Lösungen ohne Quelltext können nicht bewertet werden!**

Hinweis zur Lösungseinzugszentrale (LEZ)

Die Abgabe erfolgt per Lösungseinzugszentrale (LEZ). Sie können sich bei der LEZ mit Ihrem SCC-Benutzerkonto anmelden (https://lez.ipd.kit.edu/). Sie können pro Aufgabe eine einzelne Datei, ggf. auch ein gepacktes Archiv im ZIP-Format, abgeben. Soweit nicht anders angegeben, ist für die Programmieraufgaben hier immer `mvn package` auszuführen und die resultierende ZIP-Datei mit den Quelltextdateien Ihrer Lösung zu übertragen.
Achten Sie darauf, dass Sie eine signierte E-Mail der LEZ erhalten, in welcher eine Prüfsumme Ihrer abgegebenen Lösung enthalten ist. Diese dient bei technischen Problemen als Nachweis der Abgabe.

Hinweis zur Einzelbewertung

Die Lösungen aller Übungsblätter sind Einzelleistungen. Plagiieren von Lösungen führt zum Abzug von 25 Punkten bei allen am Plagiat Beteiligten.
Beispiel: Piggeldy lässt Frederick Aufgabe 2 abschreiben, ansonsten bearbeiten sie Übungsblatt 1 aber getrennt. Beide erhalten nun 25 Minuspunkte sowie keine Punkte für die übrigen Aufgaben von Blatt 1.

Hinweis zu Aktualisierungen des Übungsblatts

Verfolgen Sie Nachrichten in ILIAS und prüfen Sie es regelmäßig auf Aktualisierungen und Korrekturen des aktuellen Übungsblatts.