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 3

Plug-Ins, Aktivitäts-, Sequenz-, Zustandsdiagramm und Substitutionsprinzip

Ausgabe: Mittwoch, 20. Mai 2020
Abgabe: Mittwoch, 3. Juni 2020, 12:00 Uhr

Aufgrund der aktuellen Covid-19-Situation finden die Abgaben der Theorieaufgaben vorerst nicht wie gewohnt über einen Kasten im Informatikgebäude statt. Stattdessen dient die LEZ zur Abgabe Ihrer Lösungen. Diese erreichen Sie unter https://lez.ipd.kit.edu.
Beachten Sie zudem die Hinweise am Ende des Übungsblatts.

Alle Theorie-Aufgaben inklusive aller Diagramme sind handschriftlich anzufertigen!
Ausgeschlossen sind auch digital (z.B. auf dem Tablet) angefertigte „handschriftliche“ Lösungen!

Aufgabe 1: Einschub-Architektur (3 Punkte + 1 Bonuspunkt)

Seit der letzten Projekt-Sitzung mit dem Software-Architekten ist Ihre Projektleiterin der Meinung, dass neue Funktionalitäten in iMage (siehe z.B. Blatt 2) unbedingt als Einschübe (Plug-Ins) entwickelt werden sollten. Sie wissen, dass eine Anwendung zunächst vorbereitet werden muss, bevor Einschübe integriert werden können. Sie machen sich sofort ans Werk und gehen dabei wie folgt vor:

a) Informieren Sie sich über die ServiceLoader-API von Java: https://docs.oracle.com/javase/tutorial/ext/basics/spi.html
b) Holen Sie sich die Vorlage aus ILIAS und fügen Sie sie in Ihr JMJRST-Modul ein.
c) Implementieren Sie in der Klasse `PluginForJMJRST` die nötigen Methoden, um Einschübe zu sortieren. Die Sortierung soll hierbei alphabetisch basierend auf ihren Namen erfolgen. Sollten zwei Einschübe den gleichen Namen tragen, soll als zweites Sortierkriterium die Anzahl ihrer Parameter verwendet werden. Hierbei soll derjenige Einschub vor dem anderen eingeordnet werden, welcher eine geringere Anzahl an Parametern hat.
d) Implementieren Sie die Methode des `PluginManagements`, welche die verfügbaren Einschübe ermittelt und gemäß ihrer alphabetischen Reihenfolge (bzw. basierend auf der Parameteranzahl) zurückgibt.
e) Fügen Sie JMJRST ein neues Menü hinzu, in dem später alle Plug-Ins aufgelistet werden. Das Menü soll am Ende der vorhandenen Menüzeile (aber noch vor dem „About“-Menü) eingefügt werden und den Namen „Load plug-ins“ tragen.
f) Fügen Sie die verfügbaren Plug-Ins gemäß der in Aufgabenteil c) definierten Ordnung in das „Load plug-ins“-Menü ein. Jedes Plug-In soll dort zum Starten und zum Konfigurieren aufgerufen werden können. Der Menüpunkt zur Konfiguration soll nur dann eingefügt werden, wenn das Plug-In auch konfiguriert werden kann. Fügen Sie nach jedem Plug-In einen Trennstrich in das Menü ein, außer nach dem letzten.

g) Bonusaufgabe: Implementieren Sie einen Hinweis auf fehlende Plug-Ins wie folgt: Ist kein Plug-In verfügbar, soll als einziger Menüpunkt unter dem Menü „Load plug-ins“ ein Feld mit dem ausgegrauten Hinweistext „(No plug-ins available!)“ angezeigt werden.
h) Erhöhen Sie die Versionsnummer Ihres `jmjrst.main`-Moduls auf 0.1.0-SNAPSHOT.

Führen Sie die üblichen Plausibilitätstests vor der Abgabe durch.

Aufgabe 2: Fortbildungs-Einschub (3 Punkte)

Bei der Durchsicht des Quelltextes Ihrer Kollegen stellen Sie fest, dass diese anscheinend noch wenig von den neuesten Funktionalitäten von Java wissen. Als Sie diese Erkenntnis Ihrer Projektleiterin nahelegen, bekommen Sie prompt die Aufgabe, Ihren Kollegen einen „Crashkurs“ zu geben. Da für solche Weiterbildungsmaßnahmen in der Pear Corp. nur ein begrenztes Budget zur Verfügung steht, empfiehlt Ihre Projektleiterin Ihnen dies doch direkt mit der Fortbildung zu Einschüben zu kombinieren. Sie bekommen den Auftrag, einen Prototyp zu Demonstrationszwecken zu erstellen. Sie implementieren in einem Einschub-Prototyp einige neue Java-Funktionalitäten, um diese Ihren Kollegen präsentieren zu können und nutzen dabei gleichzeitig die in Aufgabe 1 erstellte Einschub-Infrastruktur:

a) Erstellen Sie ein neues Modul namens `iMage.JavaCrashCourse-plugin` für einen Demonstrations-Prototypen eines Einschubs, welcher einen Überblick über neue Funktionalitäten in Java bietet. Der Einschub (Name: JavaCrashCourse) soll die abstrakte Klasse `PluginForJmjrst` beerben und die nötigen Methoden implementieren.
b) Fügen Sie der `pom.xml` eine Abhängigkeit zu Ihrem JMJRST-Modul hinzu (dann können Sie auf die benötigten Schnittstellen zugreifen). Denken Sie an die üblichen benötigten Konfigurationsdateien (`src.xml`, usw.).
   Damit Maven die Abhängigkeit zu Ihrem Projekt erfassen kann, müssen Sie die Lösung von Aufgabe 1 in Ihr lokales Repository installieren (mit `mvn install`). Falls dies hartnäckig scheitert, können Sie stattdessen eine Abhängigkeit zum Projekt mit der ArtifactID `jmjrst-plugin-helper`, der GroupID `edu.kit.ipd.swt1` und der Version `0.0.2-SNAPSHOT` in Ihre `pom.xml` hinzufügen.
c) Wenn der Einschub erzeugt wird, soll eine unveränderliche Liste (https://docs.oracle.com/javase/9/core/creating-immutable-lists-sets-and-maps.htm) von Strings erstellt und als Attribut gespeichert werden. Diese Liste soll die bisherigen Versionen von Java seit Java 8 (nur Major-Releases) enthalten. Die Anzahl der Parameter des Plug-Ins soll der Größe dieser Liste entsprechen.
d) Der Einschub soll beim Initialisieren die Anzahl der Parameter wie folgt auf der Standard-Ausgabe ausgeben: „Found <Anzahl der Parameter> Java versions since Java 8“
e) Der Einschub soll beim Konfigurieren ein Fenster in JMJRST öffnen und darin alle Einträge der Liste auflisten. Sorgen Sie dafür, dass die Einträge zeilenweise angezeigt werden und dem Muster „Java <Versionsbez.>,“ folgen. Benutzen Sie für das Erstellen der Ausgabezeichenkette Lambda-Ausdrücke auf Streams. (https://www.baeldung.com/java-stream-operations-on-strings)
f) Der Einschub soll beim Ausführen zufällig eines der Einträge aus der Java-Versions-Liste ziehen und basierend auf dem gezogenen Eintrag eines der folgenden drei Dinge auf der Standardausgabe ausgeben: Handelt es sich um Java 14 soll „Keeping updated“ ausgegeben werden. Wird eine der anderen Java-Versionen gezogen wird „Running late“ ausgegeben. Sollte keine der erwarteten Java-Versionen gezogen oder die Liste leer sein, soll der Name des Einschubs mit angehängter Parameteranzahl ausgegeben werden (z.B. JavaCrashCourse(7)). Setzen Sie die Entscheidung über die Ausgabe mit Java-14-Switch-Expressions um. (https://openjdk.java.net/jeps/361)

g) Verpacken Sie Ihr Plug-In mit Maven entsprechend der ServiceLoader-API in ein JAR. Erzeugen Sie die benötigte Service-Datei, indem Sie in die `pom.xml` das folgende Projekt eintragen und entsprechend der Dokumentation verwenden: http://metainf-services.kohsuke.org/

Führen Sie die üblichen Plausibilitätstests vor der Abgabe durch.

Aufgabe 3: iMage-Bundle (2 Punkte)

Ihre Projektleiterin ist begeistert von der Aufteilung des iMage-Projektes in einzelne Teilmodule. „Teile und herrsche, war schon immer mein Credo!“, stellt sie fest. „Man muss aber stets darauf bedacht sein, die delegierten Teile auch zusammenzufügen!“, belehrt sie Sie. Sie erahnen, wohin dieser Monolog führt, und beginnen, noch während sie redet, proaktiv damit, ein neues Modul anzulegen, das die Module zusammenfasst, die ausgeliefert werden können.

a) Legen Sie ein neues Maven-Modul namens `iMage.bundle` mit den üblichen Einstellungen und Dateien für ein Java-Projekt an.
b) Tragen Sie als Abhängigkeit die JMJRST-Version aus Aufgabe 1 dieses Blatts ein sowie den JavaCrashCourse-Einschub aus Aufgabe 2.
c) Implementieren Sie nun die `main()`-Methode der `App`-Klasse so, dass JMJRST gestartet wird.
Tragen Sie die `App`-Klasse als Hauptklasse im Manifest ein und lassen Sie Maven die Einstellungen zum Klassenpfad ins Manifest schreiben (siehe Blatt 2). Damit es aufgrund der von uns vorgegebenen ArtifactID nicht zu Konflikten kommt (denn diese ist ja für alle Module gleich), setzen Sie zudem die Option `classpathLayoutType` auf `repository`.
Konfigurieren Sie das copy-dependencies-Plugin so, dass die Abhängigkeiten passend in das Verzeichnis `target` kopiert werden, wenn Sie `mvn package` ausführen. Anschließend sollten Sie die JAR-Datei ohne weitere Angaben mit `java -jar 1234567.jar` ausführen können (natürlich mit Ihrer Matrikelnummer oder einer nicht angepassten ArtifactID wie z.B. `iMage.bundle` und ohne die Bibliotheken aufzuführen) und in der GBO von JMJRST den JavaCrashCourse-Einschub benutzen können.

Führen Sie die üblichen Plausibilitätstests vor der Abgabe durch.

Aufgabe 4: UML-Aktivitätsdiagramm (6 Punkte)

Nachdem das neuartige Bildformat erfolgreich in iMage integriert wurde, gibt der Vorstand der Pear Corp. die Entwicklung eines Importmechanismus für klassische RGB-basierte Bilder in Auftrag. Die Abteilung für *Forschung, Entwicklung und Fortschritt* hat das Importieren bereits durchdacht und versucht nun, Ihnen den Ablauf eines Imports in einer Projektsitzung näher zu bringen. Leider können Sie den komplexen Gedankengängen der Forscherinnen und Forscher nur bedingt folgen. Um die Kommunikation zwischen den Abteilungen zu verbessern, beschließen Sie daher gemeinsam, den Ablauf des Imports in Form eines UML-Aktivitätsdiagramms zu modellieren. Die Forscherinnen und Forscher geben Ihnen folgende Beschreibung an die Hand:

„Solange noch nicht alle Bildpunkte des RGB-Bildes importiert wurden, wählen Sie einen der noch nicht importierten Punkte aus. Anschließend rechnen Sie die Farbkomponenten des Bildpunktes (R, G und B) in einen Helligkeitswert und zwei Buntheitswerte um. Als Laie können Sie die beiden Buntheitswerte als „Bläue“ und „Röte“ auffassen. Beide Werte müssen nun so reduziert werden, dass sie durch die Farbtiefe des Bildpunktes darstellbar sind. Bestimmen Sie daher zunächst die Farbtiefe des Punktes, bevor Sie die „Bläue“ und „Röte“ parallel

zueinander reduzieren. Im nächsten Schritt prüfen Sie, ob der Bildpunkt zusammen mit seinen Nachbarpunkten einen bereits vollständig importierten Block von 8x8 Punkten ergibt. Falls nicht, importieren Sie den nächsten Bildpunkt. Falls doch, wenden Sie anschließend eine Transformation auf die Farbwerte des Blocks an. Im Anschluss müssen Sie die transformierten Farbwerte zunächst in ganzzahlige Werte umformen. Um die Farbwerte zu komprimieren, unterziehen Sie den Block danach basierend auf der durchschnittlichen Größe seiner Farbwerte entweder einer Huffman-Kodierung (>= 128) oder einer arithmetischen Kodierung (< 128). Anschließend fahren Sie mit dem Import des nächsten Bildpunktes fort. Sobald alle Punkte importiert wurden, ist der Import beendet. Mit den Details der obigen Aktionen müssen Sie sich nicht auseinandersetzen, denn diese erhalten Sie aus der Forschungsabteilung als Softwarebibliotheken."

Aufgabe: Setzen Sie die obenstehende Beschreibung als Aktivitätsdiagramm um.

Hinweis: Verwenden Sie bei Ihrer Modellierung korrekte UML-Notation. Orientieren Sie sich dabei am Abstraktionsniveau der obigen Beschreibung. Verwenden Sie – falls nötig – sinnvolle Objektknoten um die Eingabe- und Ausgabedaten einer Aktion zu modellieren.

Aufgabe 5: UML-Zustandsdiagramm (5 Punkte + 2 Bonuspunkte)

Um die Attraktivität der Pear Corp. als Arbeitgeber zu erhöhen, beschließt der Vorstand, eine Betriebstankstelle auf dem Firmengelände zu errichten. Leider kommt es bei den Softwaretechnikern mehrfach zu Verständnisproblemen bei der Benutzung der Zapfsäulen. Um die Verständlichkeit für ausgebildete Softwaretechniker zu erhöhen, beschließen Sie uneigennützig, die Funktionsweise der aufgestellten Zapfsäulen durch ein UML-Zustandsdiagramm zu modellieren.

a) Modellieren Sie die folgende Funktionsweise einer Zapfsäule als UML-Zustandsdiagramm mit minimaler Anzahl an Transitionen:

   *Die Zapfsäule ist zunächst inaktiv. Das Ereignis **aushängen** tritt ein, wenn die Zapfpistole aus der Halterung ausgehängt wird. Im Anschluss daran kann eine Vorauswahl des zu investierenden Geldbetrages getroffen werden. Im Auslieferungszustand ist kein Geldbetrag vorausgewählt. Der einzige bei der Vorauswahl unterstützte Betrag ist 50 € und kann durch Empfang des Ereignisses **auswählen** ausgewählt werden. Das Zurücksetzen der Vorauswahl ist per **abwählen** möglich. Wird die Zapfpistole während des Auswahlvorgangs wieder in die Halterung eingehängt, tritt das Ereignis **einhängen** ein und die Zapfsäule wird erneut inaktiv. Beginnt der Tankvorgang, so tritt das Ereignis **tanken** ein. Das Tanken wird manuell beim Eintreten des Ereignisses **loslassen** pausiert, oder automatisch beim Eintreten des Ereignisses **erreichen** (d. h. der vorausgewählte Geldbetrag wurde erreicht). In beiden Fällen kann das Tanken jedoch fortgesetzt werden, was durch den erneuten Eintritt des Ereignisses **tanken** repräsentiert wird. Die Zapfpistole lässt sich nur bei pausiertem Tankvorgang wieder in die Halterung einhängen, was durch das Eintreten des Ereignisses **einhängen** dargestellt wird. Sobald danach das Ereignis **bezahlen** eintritt, geht die Zapfsäule in ihren Ursprungszustand über. Bei jedem Aushängen der Zapfpistole wird die zuletzt getroffene Vorauswahl erneut gewählt.*

b) Bonusaufgabe: Leider kann einer der Softwaretechniker keine UML-Zustandsdiagramme mit Hierarchie oder Gedächtnis verstehen. Um Abhilfe zu schaffen, wandeln Sie Ihre Lösung aus Aufgabenteil a) in ein äquivalentes Zustandsdiagramm ohne Hierarchie und Gedächtnis um.

Aufgabe 6: Geheimnisprinzip (6 Punkte)

Bei den Entwicklungsteams verbreitet sich immer mehr Unmut über die Verwendung von JMJRST und auch die Geschäftsführung realisiert so langsam, dass Sie vielleicht „auf das falsche Pferd gesetzt haben“. Daher wird Ihre Projektleiterin damit beauftragt, konzeptionelle Probleme von JMJRST zusammenzutragen. Ihre Projektleiterin delegiert die Aufgabe direkt an Sie (wie es sich für Führungskräfte gehört) und möchte von Ihnen eine erste Abschätzung dazu, „wie schlimm es denn wirklich ist“. Sie erinnern sich, dass Verletzungen des Geheimnisprinzips eine wesentliche Fehlerquelle (gerade bei der Wartung und Erweiterung von Alt-Software) darstellen. Sofort beginnen Sie JMJRST nach Verletzungen des Geheimnisprinzips zu durchforsten.

Aufgabe: Finden Sie vier (4) Verletzungen (bzw. Verstöße) gegen das Geheimnisprinzip im Projekt `jmjrst.main`. Dokumentieren Sie die Verletzungen jeweils indem Sie die zugehörige Klasse, die Zeilennummer(n) und relevante Quelltextbestandteile angeben. Beschreiben Sie außerdem jeweils stichpunktartig, warum es sich an dieser Stelle um eine Verletzung des Geheimnisprinzips handelt und erläutern Sie (ebenfalls stichpunktartig) wie das Problem behoben werden könnte.

Hinweis: Für die Beschreibung der Fehlerbehebung genügt eine konzeptionelle Beschreibung, Sie müssen keine verbesserte Implementierung angeben (dürfen das aber natürlich gerne tun).

Aufgabe 7: Liskov’sches Substitutionsprinzip (2 Bonuspunkte)

Das Entwicklungsteam, das für die Entwicklung der internen Buchhaltungs-Software der Pear Corp. zuständig ist, ist auf ein (für das Team) nicht lösbares Problem gestoßen. Natürlich verschweigt die Projektleitung das Problem. Da sich innerhalb der Pear Corp. aber bereits herumgesprochen hat, dass Sie sich darauf verstehen auch die schwierigsten Probleme zu lösen, wendet sich eine Entwicklerin des Teams in einer Kaffeepause vertrauensvoll an Sie. Sie hat bereits die relevanten Programmteile ausgedruckt und reicht sie nun an Sie weiter:

Klasse `AppPrototype`:

```
01: public static void main(String[] args) {
02:
03:    List<String> el = new EmployeeManagementPrototype().generateInitialEmployeeList();
04:      el.add("Martina Mustermann");
05:   }
06: }
```

Klasse `IEmployeeManagement`:

```
07: public interface IEmployeeManagement {
08:
09:   /**
10:    * Generates the initial list of employees. Each employee is represented as a String
       * (Given Name(s), Last Name). New employees (Strings) may be added or removed from
       * this List.
11:    * @return the initial list of employees
12:    */
13:   List<String> generateInitialEmployeeList();
14: }
```

Klasse `EmployeeManagementPrototype`:

```
15: public class EmployeeManagementPrototype implements IEmployeeManagement {
16:
17:   @Override
18:   public List<String> generateInitialEmployeeList() {
19:     return List.of("Martin S. Pinnér", "Marvin Bertsch", "Max Power");
20:   }
21: }
```

Aufgabe: Die obenstehende Implementierung verletzt das Liskov'sche Substitutionsprinzip. Geben Sie an, an welcher Stelle (bzw. an welchen Stellen) das Substitutionsprinzip verletzt wird und erläutern Sie stichpunktartig, warum es sich jeweils um eine Verletzung des Prinzips handelt.

Hinweise zur digitalen Abgabe der Theorieaufgaben

Digitalisierung und Komprimierung:

Damit Ihre handschriftlichen Lösungen entgegengenommen werden können, müssen Sie diese digitalisieren. Hierfür scannen Sie Ihre Lösungen bitte ein oder erstellen alternativ Fotos von diesen. Bitte nummerieren Sie die Blätter dazu und lösen maximal eine Aufgabe pro Seite. Also z.B. Aufgabe 1 a) bis c) auf Seite 1, Aufgabe 2 auf Seite 2 usw. Schreiben Sie nicht die Lösungen für Aufgabe 2 und 3 auf die gleiche Seite, da die LEZ eine Abgabe pro Aufgabe erwartet. Erstellen Sie die Bilder im JPEG-Format, um die Dateigröße später einfach reduzieren zu können.
Die LEZ nimmt Abgaben nur bis zu einer Größe von **15Mb pro Aufgabe** an. Nachdem Sie Ihre Lösungen digitalisiert haben, müssen Sie diese also ggf. komprimieren. Wir empfehlen hierzu die Anwendung JPEGOptimizer (https://github.com/collicalex/JPEGOptimizer/releases).

JPEG Optimizer:

Wenn Sie die .jar Datei öffnen, finden Sie die Möglichkeit Quelle und Ziel Ihrer Dateien festzulegen. Wenn Sie Ihre Originale behalten wollen, wählen Sie 2 unterschiedliche Dateipfade. Ansonsten ist es auch möglich, Ihre Bilder zu überschreiben. In dem Feld „Max Diff:“ empfehlen wir Werte zwischen 0.75% und 1.5%, diese haben sich als guter Kompromiss zwischen Größe und Qualität ergeben. Anschließend starten Sie den Komprimierungsvorgang mit einem Klick auf „Optimize“ in der rechten oberen Ecke der GUI.

Abgabe bei der LEZ:

Kontrollieren Sie Ihre Abgaben auf Lesbarkeit. Sollte Ihre Abgabe unlesbar sein, führt dies zu Punktabzug.
Erstellen Sie **ein .zip Archiv pro Aufgabe** und nennen dieses **Aufgabe_<Nummer>.zip.**
Das fertige .zip-Archiv laden Sie dann nach dem gleichen Schema wie bei den Praxisaufgaben zur LEZ hoch. **Sie haben zwei Wochen zur Bearbeitung jedes Übungsblattes Zeit, wir können nicht garantieren, dass die LEZ 5 Minuten vor Fristende standhält, wenn hunderte Studenten gleichzeitig hochladen.**

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.