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_945596&client_id=produktiv

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

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

SWT1: Übungsblatt 5

Architekturstile, Entwurfsmuster, Implementierung und Git

Ausgabe: Mittwoch, 19. Juni 2019
Abgabe: Mittwoch, 3. Juli 2019, 12:00 Uhr

Geben Sie Ihre Lösung mit Deckblatt (mit Name, Matrikelnummer und der Nummer Ihres Tutoriums deutlich lesbar) ab, damit Ihr Tutor Korrekturhinweise und Ihre Punkte notieren kann. Werfen Sie es in die Holzkiste vor Raum 369 im Informatik-Hauptgebäude 50.34. Verwenden Sie ausschließlich das Deckblatt zur SWT1 aus ILIAS.

Alle Theorie-Aufgaben inklusive aller Diagramme sind handschriftlich anzufertigen!
Ausgeschlossen sind auch kopierte oder ausgedruckte „handschriftliche“ Lösungen!

Aufgabe 1: Architekturstile (4 Punkte)

Nachdem iMage mit der Grafischen Benutzeroberfläche „so langsam ansehnlich wird“ (Zitat Dr. Salzhügel aus dem Vorstand), wird es für die Pear Corp. Zeit, das Produkt zu kommerzialisieren. Hierzu soll ein Webshop entstehen, in dem Kunden unterschiedliche Produkte der Pear Corp. erwerben können.
Ihre Projektleiterin gibt die Idee von Dr. Salzhügel wie folgt wieder:

> *Die Informationen über die Produkte liegen in Datenbanken. Die Pear Corp verwendet unterschiedliche Datenbanksysteme, unter anderem MagdalenaDB und PregresSQL. Die Darstellung des Webshops soll je nach Endgerät variieren. Neue Darstellungsformen sollen zur Laufzeit hinzugefügt und entfernt werden. Zu Beginn sollen Darstellungen für das mobile Betriebssystem der Pear Corp., iOz, sowie für die weit verbreiteten Browser Mangan und Watercoon angeboten werden. Eine Registrierungskomponente soll das Hinzufügen von neuen Darstellungen steuern und eine Aktivitätskomponente soll Aktivitäten, wie Maus-Klicks registrieren und verarbeiten.*

Da Sie immer noch darauf hoffen, (irgendwann) zum Software-Architekten befördert zu werden, melden Sie sich freiwillig für den Entwurf einer Architektur für den Webshop. Natürlich möchten Sie mit einer besonders ausgefeilten Architektur glänzen und beschließen daher, nicht nur den in solchen Fällen üblichen Architekturstil *Modell-Präsentation-Steuerung* anzuwenden, sondern die einzelnen Bestandteile zusätzlich in einer *intransparenten 3-Schichtenarchitektur* zu gliedern.

a) Entwerfen Sie das System als Modell-Präsentation-Steuerung-Architektur. Gliedern Sie die Bestandteile außerdem als intransparente 3-Schichtenarchitektur. Stellen Sie die Architektur mithilfe eines UML-Paketdiagramms dar. Modellieren Sie dabei die Schichten als Pakete und die Schnittstellen und Klassen als Bestandteile dieser Pakete. Nutzen Sie – wenn nötig – das Entwurfsmuster Fassade für die Kapselung der öffentlichen Schnittstelle einer Schicht. Modellieren Sie außerdem die Benutzt-Beziehungen der Schnittstellen und Klassen innerhalb und zwischen den Paketen.

b) Welche Entwurfsmuster außer der Fassade verwendet Ihre Architektur. Nennen Sie jeweils den Grund für den Einsatz des Musters und erläutern Sie kurz die Funktionalität des Musters im konkreten Beispiel.

Hinweis: Geben Sie – wenn nötig – in Ihrem Paketdiagramm auch Methoden und Attribute der Klassen bzw. Schnittstellen an, um die Semantik vollständig wiedergeben zu können.

Aufgabe 2: Jäger des verlorenen Entwurfsmusters (3 Punkte)

Ihr Team ist inzwischen total angetan von Entwurfsmustern und sucht selbst überall nach Entwurfsmustern in Software-Projekten. Da die Kantine heute geschlossen hat, entscheiden Sie sich, Ihrem Hirn wenigstens geistige Nahrung zu geben und helfen ihnen bei der Suche in der Mittagspause. Schnell werden Sie fündig.

Geben Sie an, welches Entwurfsmuster die Klasse `org.hibernate.cache.CacheProvider` inklusive ihrer Unterklassen `OSCacheProvider` und `SwarmCacheProvider` bilden.
(https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/cache/CacheProvider.html)

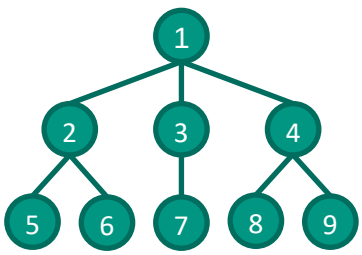
Zeichnen Sie zu Ihrem gefundenen Entwurfsmuster ein UML-Klassendiagramm, welches den Ausschnitt aus der Hibernate-API wiedergibt, der das gefundene Entwurfsmuster repräsentiert. Beschränken Sie sich in Ihrem UML-Klassendiagramm auf die für das Entwurfsmuster relevanten Methoden und Attribute. Geben Sie an, welche der Klassen, Methoden und Attribute in Ihrem UML-Klassendiagramm welchen Rollen im Entwurfsmuster entsprechen. Verwenden Sie dazu die Begriffe aus der Vorlesung. Begründen Sie in ein bis zwei Sätzen, warum Sie dieses Muster erkannt haben.

Aufgabe 3: Entwurfsmuster (Iterator, Besucher) (6 Punkte + 2 Bonuspunkte)

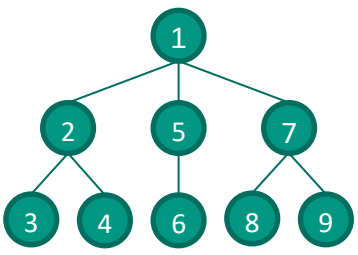
Der Vorstand der Pear Corp. hat beschlossen die iMage-Applikation so zu erweitern, dass diese selbstständig nach Bildern in Verzeichnisstrukturen sucht (und später in der Applikation auflistet). Ihre Projektleiterin hat derweil Wind davon bekommen, dass sich Ihr Team bestens mit Entwurfsmustern auskennt und möchte diese Fähigkeit gewinnbringend (für sich selbst) nutzen, indem sie das neue Projekt als Muster-Projekt unter dem Motto „Entwurfsmuster als Innovationstreiber“ von Ihrem Team umsetzen lässt.

Ihre Aufgabe besteht darin, einen Prototyp zu implementieren, welcher einen Unterverzeichnisbaum nach potenziellen Bilddateien absucht. Der Verzeichnisbaum soll in unterschiedlichen Varianten traversiert werden. Hierzu soll das Entwurfsmuster *Iterator* verwendet werden. Die eigentliche Baum-Traversierung soll mithilfe des Entwurfsmusters *Besucher* umgesetzt werden. Für den Prototypen genügen Ihnen zwei verschiedene Traversierungsarten (`Traversal`).:

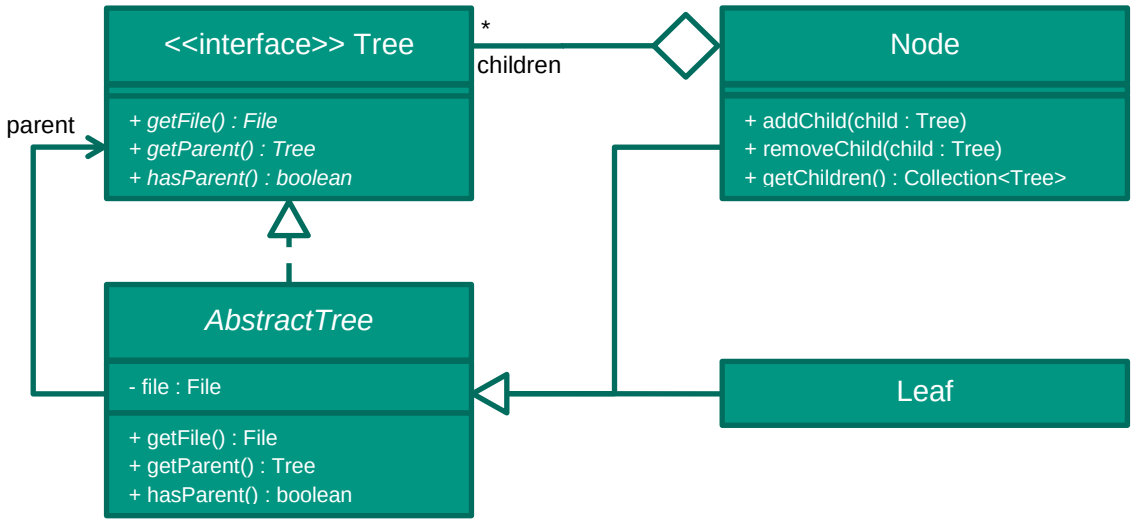
Ebenenweise (`BreadthTraversal`): Beginne bei der Wurzel und durchlaufe dann alle Ebenen nacheinander von links nach rechts (vgl. Breitensuche).



Hauptreihenfolge (`DepthTraversal`): Durchlaufe die Wurzel und anschließend zuerst den linken, dann den rechten Teilbaum (vgl. Tiefensuche).



Die verwendete Baum-Datenstruktur ist wie folgt aufgebaut:



Laden Sie aus dem ILIAS die Vorlage für diese Baumstruktur herunter und platzieren Sie diese in einem neuen Maven-Modul `iMage.treeTraversal`.

a) Erweitern Sie die Vorlage um Funktionalität, mit der die gegebene Baumstruktur mit unterschiedlichen Varianten traversiert werden kann. Damit Ihre Lösung später möglichst leicht erweitert und die Datenstruktur flexibel verändert werden kann, benutzen Sie zur Umsetzung die Entwurfsmuster *Iterator* und *Besucher*. Hierbei soll der *Iterator* (Klasse `Traversal`) intern einen *Besucher* benutzen, um flexibel die verschiedenen Bestandteile des Baumes zu verarbeiten. Ihr *Iterator* soll von einer beliebigen `Tree`-Instanz zur Verfügung gestellt werden. Achten Sie darauf, dass die benutzte Traversierungsmethode für den gelieferten Iterator gewählt werden kann (wenn er zur Verfügung gestellt wird).

b) Implementieren Sie in der Klasse `DepthTraversal` einen konkreten Iterator, welcher die oben dargestellte Traversierungsvariante (*Hauptreihenfolge*) umsetzt.

c) Bonusaufgabe: Implementieren Sie in der Klasse `BreadthTraversal` die ebenenweise Traversierung.

d) Schreiben Sie für Ihre Implementierung Testfälle, die sicherstellen, dass Ihre Implementierung den Baum korrekt traversiert.

e) In der Klasse `Runner` ist der Algorithmus für das Auslesen eines Verzeichnisbaums und die Auswahl bestimmter Dateitypen mithilfe der `Tree`-Datenstruktur vorgegeben. Implementieren Sie die fehlenden Methoden.
   i. Vom gegebenen Startordner abwärts sollen alle Ordner und Dateien in einem `Tree` gespeichert werden (`buildFolderStructure`-Methode).
   ii. Die `getFiles`-Methode traversiert den Baum mit der gegebenen Traversierungsvariante und gibt alle `.jpg`- und `.png`-Dateien zurück.
   iii. Benutzen Sie den vorgegebenen Einschub (`selectFiles`) der Schablonenmethode, um zwei unterschiedliche Ausprägungen des Algorithmus zu erstellen. Eine soll nur `.jpg`-Dateien, die andere nur `.png`-Dateien zurückgeben.
   iv. Geben Sie in der `printResults`-Methode die Pfade der Dateien in geeigneter Form auf der Kommandozeile aus.

f) Passen Sie die Kommandozeilenschnittstellen-Implementierung (Klasse `App`) so an, dass sie Ihre entsprechende Runner-Implementierung passend zur gewählten Option aufruft.

Führen Sie vor der Abgabe die üblichen Plausibilitätstests durch.

Aufgabe 4: Implementierung umgekehrt (5 Punkte)

Trotz mehrfacher Intervention Ihrerseits wurde der Praktikant Herr Knöpfle von Ihrer Projektleiterin dazu bestimmt, das neue Warenkorb-Verwaltungssystem für den Webshop der Pear Corp. zu implementieren (siehe A1). Ihre Projektleiterin meint dazu: „Der arbeitet hier quasi kostenlos. Wenn's gar nichts wird, können Sie es ja nochmal richtig bauen. Andernfalls war die Entwicklung sehr billig." Widerwillig geben Sie Ihr in diesem Punkt recht, misstrauen aber weiterhin den Implementierungsfähigkeiten von Herrn Knöpfle. Nach seiner vermutlich letzten Einbuchung des Tages mit der Nachricht „Feierabend-Commit (wip)" klonieren Sie sein Depot und versuchen seine bisherige Implementierung zu verstehen. Schnell stellen Sie fest, dass es sinnvoll wäre ein UML-Klassendiagramm anzufertigen, um die Zusammenhänge besser zu verstehen.

Klasse Warenkorb:

```
01:  public class Warenkorb {
02:
03:    Map<String, Artikel> warenkorbArtikel;
04:    Bestandteil[] warenkorbelemente;
05:    Set<Bestellvorgang> vorgaenge;
06:
07:    public Warenkorb() {
08:      warenkorbArtikel = new HashMap<>();
09:      warenkorbelemente = new Bestandteil[42];
10:      for (Bestandteil b : warenkorbelemente) { b = new Bestandteil(this); }
11:      vorgaenge = new HashSet<>();
12:    }
13:
14:    public void bestellvorgangHinzufuegen (Bestellvorgang b) { vorgaenge.add(b); }
15:  }
```

Klasse Artikel:

```
01:  public class Artikel {
02:  }
```

Klasse Bestandteil:

```
01:  public class Bestandteil {
02:
03:    Warenkorb w;
04:
05:    public Bestandteil(Warenkorb w) { this.w = w; }
06:  }
```

Klasse Bestellvorgang:

```
01:  public class Bestellvorgang {
02:
03:    Warenkorb w;
04:    Bestellung b;
05:
06:    public Bestellvorgang(Warenkorb w, Bestellung b) {
07:      this.w = w;
08:      this.b = b;
09:      w.bestellvorgangHinzufuegen(this);
10:      b.bestellvorgangHinzufuegen(this);
11:    }
12:  }
```

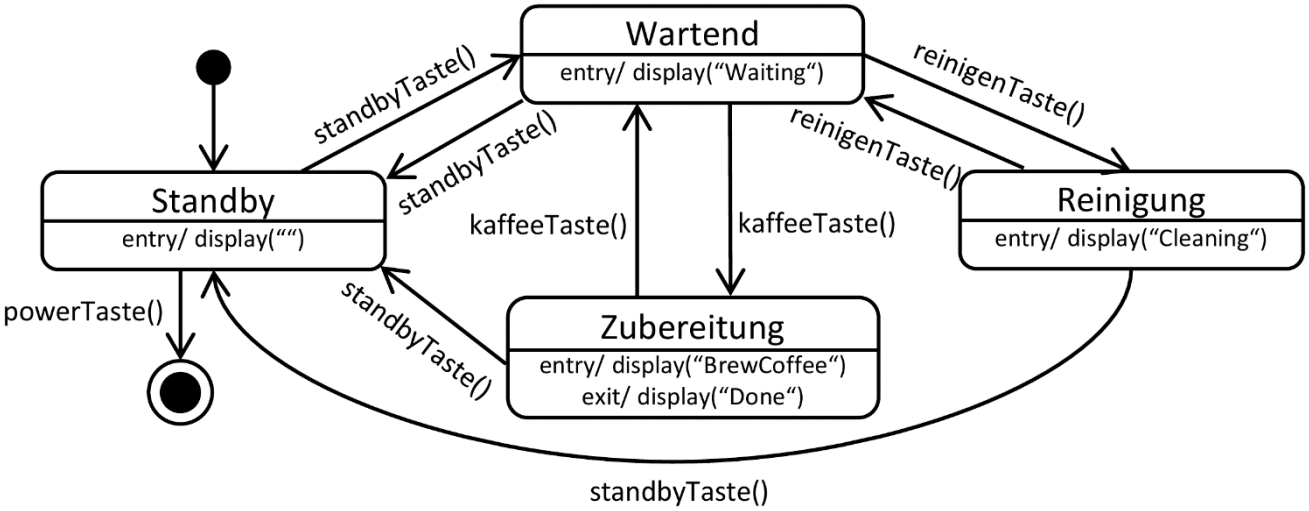
Klasse Bestellung:

```
01:  public class Bestellung {
02:
03:    Set<Bestellvorgang> vorgaenge;
04:
05:    public Bestellung() { vorgaenge = new HashSet<>(); }
06:
07:    public void bestellvorgangHinzufuegen (Bestellvorgang b) { vorgaenge.add(b); }
08:  }
```

Aufgabe: Erstellen Sie anhand der oben angegebenen Implementierung ein UML-Klassendiagramm. Halten Sie sich an die in der Vorlesung vorgestellten Regeln zur Umsetzung von UML-Klassendiagrammen in Quelltext (natürlich umgekehrt in diesem Fall). Achten Sie insbesondere darauf, die jeweiligen Assoziationen möglichst exakt wiederzugeben.

Aufgabe 5: Implementierung eines Zustandsautomaten (5 Punkte)

Nachdem Sie vergangene Woche gemeinsam mit Ihrer besten Entwicklerin festgestellt haben, dass die Software der Kaffeemaschine nicht in Ordnung ist (siehe ÜB 4, A2), beobachten Sie eine deutliche Verschlechterung des Arbeitsklimas in Ihrem Team aufgrund der unzureichenden Kaffeeversorgung. Voller Begeisterung für Ihr Zustandsdiagramm hat Ihre beste Entwicklerin ein Zustandsdiagramm für eine sinnvolle Steuerungssoftware der Kaffeemaschine entworfen. Da Sie so viel Enthusiasmus natürlich unterstützen möchten (und außerdem befürchten, dass ohne Kaffee in Ihrem Team bald gar nichts mehr läuft), beschließen Sie den Zustandsautomaten (noch nach Feierabend) zu implementieren.



Aufgabe: Implementieren Sie den obenstehenden Zustandsautomat in Java. Geben Sie wie gewohnt iMage als Elter-Projekt an und fügen Sie die benötigten Dateien (`src.xml` usw.) für die Abgabe hinzu.

Beachten Sie folgendes:

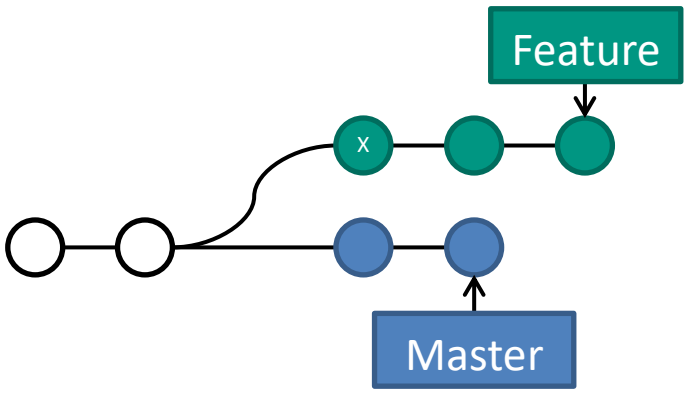
1. Legen Sie ein neues Modul `iMage.iLonghDe` an, in dem Sie die Implementierung durchführen.
2. Verwenden Sie für Ihr neues Modul die Abhängigkeit mit der GroupID `swt1`, der ArtifactID `iMage.iLonghDe.base` und Version `0.0.1-SNAPSHOT`. Ihr Automat muss die darin enthaltene Schnittstelle `IStateMachine` implementieren.

3. Ihr Zustandsautomat muss den Klassennamen `CoffeeMachine` tragen und im Paket `org.iMage.iLonghDe` liegen.
4. Schreiben Sie mindestens 4 sinnvolle, grüne Tests für Ihren Zustandsautomaten.
5. Ihre Implementierung soll sicherstellen, dass es zu jedem Zeitpunkt einfach ist einen neuen Zustand hinzuzufügen ohne alle anderen Klassen anpassen zu müssen. Wählen Sie hierfür die geeignete Umsetzung aus den in der Vorlesung vorgestellten.

Führen Sie vor der Abgabe die üblichen Plausibilitätstests durch.

Aufgabe 6: Git (4 Punkte)

Nicht erst seit dem Klonieren des Depots von Herrn Knöpfle ist Ihnen klar, dass die beiden Praktikanten Knöpfle und Spätzle wohl keine allzu gute Ausbildung an der Schwäbischen Hochschule für Informationstechnologie genossen haben. Insbesondere der Umgang mit Versionsverwaltungssystemen lässt zu wünschen übrig. Den beiden ist zum Beispiel völlig unklar, warum man neben merge auch rebase in Git benötigt. Da eine Kette nur so stark ist, wie ihr schwächstes Glied, beschließen Sie diese beiden besonders schwachen Glieder weiterzubilden. Neben einem kleinen Seminar mit dem Namen „git – get it truly“ bereiten Sie auch kleine Übungsaufgaben zum Verständnis einzelner Git-Funktionen für die beiden vor. Um sich über die Erwartungshaltung klar zu werden, lösen Sie die Aufgaben zunächst selbst.



Gegeben sei der obenstehende Einbuchungsgraph (für Git).

a) Verdeutlichen Sie anhand des obigen Beispiels die unterschiedlichen Möglichkeiten das Arbeitsverzeichnis auf den mit „X“ markierten Zustand zurückzusetzen. Beschreiben Sie hierzu sowohl den nötigen Befehl als auch die Auswirkungen auf die Historie.
b) Zeichnen Sie jeweils den resultierenden Einbuchungsgraph für das Einbinden der Änderungen des Master-Zweigs in den Feature-Zweig nach der Ausführung der Befehle *rebase* bzw. *merge*.
c) Diskutieren Sie stichpunktartig die Vor- und Nachteile der beiden Strategien bzw. Befehle *rebase* und *merge* bei Verwendung in einem lokalen Depot

Hinweis: Sie können für die Aufgabenteile a), b) und c) davon ausgehen, dass Sie sich aktuell auf dem Feature-Zweig befinden und von dort die Befehle ausführen, z.B. *git merge master*

Betrachten Sie das folgende Szenario: Sie arbeiten lokal an Ihrem Feature-Zweig und Ihre Kollegen arbeiten unterdessen auf dem im entfernten Depot liegenden Master-Zweig.

d) Begründen Sie, warum es keine gute Idee ist mittels *rebase* die Änderungen im Feature-Zweig in den Master-Zweig zu überführen.

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 11.

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: Herr Knöpfle lässt Frau Spätzle 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.