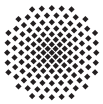

**Universität Stuttgart**

SQA
Software Quality and Architecture Group

# Übungsblatt 12

Programmierung und Softwareentwicklung (WS 2024/2025)

Abgabe: Fr. 31.01.2025, 23:59 Uhr — Besprechung: KW 05

- Bitte lösen Sie die Übungsaufgabe in **Gruppen von 2 Studierenden**.
- Dieses Übungsblatt besteht aus zwei Teilen (A, B). Teil A ist in der Präsenzübung zu lösen. Teil B ist in Heimarbeit (Gruppe von 2 Studierenden) zu lösen und rechtzeitig abzugeben. Die Abgabe erfolgt über ILIAS.
- Geben Sie `.java`-Dateien nur im UTF-8 Encoding ab. Ändern Sie das Textdateiencoding von Eclipse auf UTF-8 ab, bevor Sie die Unterlagen herunterladen. Abhängig von Ihrem Betriebssystem müssen Sie möglicherweise auch nichts tun.
- Geben Sie zu Beginn der Dateien Ihre Namen (Vor- und Nachname), die Matrikelnummern und die E-Mail-Adressen an. Nutzen Sie bei Java-Dateien die korrekte JavaDoc-Syntax.
- Benennen Sie die Dateien nach dem folgenden Schema:
  1. **PSE**[**ÜB-Nr**]-[**Nachnamen der Teammitglieder**]-[**Nachname des Tutors**]**.pdf**
     Beispiel: PSE12-StießSpethKrieger-Becker.pdf
  2. [**Klassenname**]**.java**: Alle von Ihnen bearbeiteten Java-Dateien, die Lösungen für die Aufgaben enthalten.
- Missachtung der formalen Kriterien kann dazu führen, dass einzelne Aufgaben oder die gesamte Abgabe mit 0 Punkten bewertet werden.

**Lernziel**: Auf diesem Übungsblatt werden Sie Ihr Wissen über Testen vertiefen und Ihr Wissen zu Fehlertypen auffrischen. Des Weiteren werden Sie den Umgang mit Tests, Lambdas und Streams und OOP weiter üben.

**Punkte**: Dieses Übungsblatt enthält zwei Teile. Im Teil B können Sie bis zu 45 Punkte erzielen. Zum Bestehen des Blatts benötigen Sie mindestens 22.5 Punkte.

**Style**: Bitte halten Sie die in der Vorlesung vorgestellten Style-Regeln ein. Dazu gehören auch JavaDoc sowie Vor- und Nachbedingungen. Der Style Ihrer Implementierung wird mit bis zu 50% bewertet.

**Vorbereitung**: Bitte erledigen Sie die folgenden Schritte **vor** der Präsenzübung.

- Importieren Sie das zu diesem Übungsblatt gehörende Maven Projekt in Ihre IDE. Sie finden das Maven Projekt in unserem Git-Repository:
  `https://github.com/SQA-PSE-WS-2024-2025/exercise-sheet-12`
- Stellen Sie sicher, dass Sie Übungsblatt 11 absolviert haben, sowie alle Software installiert und funktionsfähig ist (IDE (Eclipse, IntelliJ, VSCode,...) und Java 21).

**Unterlagen**:

- Git-Repositories: `https://github.com/SQA-PSE-WS-2024-2025/`
- Dokumentation des Hamstersimulators: `https://tinyurl.com/5yx654w8`
- Dokumentation von JUnit5: `https://junit.org/junit5/docs/current/api/index.html`

**Scheinkriterien**: Durch die Teilnahme am Übungsbetrieb können Sie sich für die Teilnahme an der Klausur qualifizieren:

- Bestehen von min. 80% aller Übungsblätter.
- Ein Übungsblatt gilt als bestanden, wenn 50% der Punkte des abgegebenen Heimarbeitsteils erreicht wurden.
- Aktive Teilnahme an min. 80% der Übungen.

## Viel Erfolg!

# 1 Teil A - Präsenzaufgaben

**Aufgabe 1** Äquivalenzklassen

In dieser Aufgabe geht es um Blackbox Tests, das heißt Sie leiten Testfälle aus der Spezifikation des zu testenden Codes ab. Betrachten Sie verschiedene Operationen der Klasse `Math` der JBCL.

(a) Betrachten Sie die Operation `addExact(int x, int y)`[1]. Was sind mögliche Eingabedaten, um die Operation `addExact` zu testen? Was sind die erwarteten Ergebnisse?

(b) Betrachten Sie weiterhin die Operation `addExact(int x, int y)`. Um alle möglichen Eingabedaten abzudecken, brauchen Sie sehr viele Testfälle. Dies ist nicht sinnvoll. Stattdessen zerlegt man den Eingabebereich normalerweise in Äquivalenzklassen. Betrachten Sie die in Tabelle 1 beschriebenen Äquivalenzklassen.

Diskutieren Sie mit Ihrem Partner, welche der Äquivalenzklassen sinnvoll gewählt wurden und welche nicht. Begründen Sie Ihre Antwort und halten Sie die Ergebnisse schriftlich fest.

| Äquivalenzklasse | Repräsentant | Ergebnis |
|---|---|---|
| Summanden a und b, sodass a+b außerhalb des Integer Wertebereichs liegt | (-1, `Integer.MIN_VALUE`) | `ArithmeticException` |
| Summanden a und b sind gerade und a+b liegt im Integer Wertebereich | (42,64) | 106 |
| Summanden a und b sind ungerade und a+b liegt im Integer Wertebereich | (21,3) | 24 |
| Summanden a und b, sodass a+b ungerade ist und im Integer Wertebereich liegt | (21,18) | 39 |
| Summanden a und b sind beide 0 | (0,0) | 0 |

Tabelle 1: Schlecht gewählte Äquivalenzklassen.

(c) Definieren Sie eigene Äquivalenzklassen für die Operation `floorDiv`[2]. Notieren Sie jeweils eine Beschreibung der Äquivalenzklassen, einen Repräsentanten und das für diesen zu erwartende Ergebnis analog zu Tabelle 1.

(d) Öffnen Sie die Klasse `MathTest`. Implementieren Sie Testfälle für die in Teilaufgabe (c) identifizierten Äquivalenzklassen. Führen Sie die Testfälle aus. Sie finden die Klasse im Packet `de.unistuttgart.iste.sqa.pse.sheet12.presence.blackboxtests`. Beachten Sie, dass Sie das Packet in *src/test/java* finden, nicht wie bisher in *src/main/java*.

*Hinweis:* Um diese Teilaufgaben etwas spannender zu gestalten, testen Sie anstelle der `floorDiv` Operation aus der JBCL – die hoffentlich korrekt implementiert wurde – eine fehlerhafte Reimplementierung der Operation. In der fehlerhaften Reimplementierung gibt es zwei Fehler zu finden!

**Aufgabe 2** Fehlertypen

In diese Aufgabe geht es um Fehlertypen[3].

(a) Finden Sie im folgenden Code der `PandaZoo`-Klasse aus Listing 1 zehn *unterschiedliche* Fehler. Dies können lexikalische Fehler, Syntaxfehler, statische oder dynamische Semantikfehler, oder auch Stilfehler sein. Wichtig ist hier, dass sich die Fehler unterscheiden (z.B. nicht für zwei unterschiedliche Zeilen jeweils „Parameterklammern vergessen“ notieren).

Geben Sie für **jeden** Fehler Folgendes an:

- Die Zeilennummer(n)
- Eine Klassifizierung des Fehlers (lexikalisch, syntaktisch, dynamisch/statisch semantisch, stilistisch)
- Eine kurze Begründung (1-2 Sätze), warum es sich um den von Ihnen vorgeschlagenen Fehlertyp handelt

[1] `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html#addExact(int,int)`

[2] `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html#floorDiv(int,int)`

[3] Diese Aufgabe basiert auf einer PSE Klausuraufgabe aus dem WS 2021/22. Die Original Aufgabe ist etwas länger und enthält auch Streams und Lambdas.

- Eine kurze Beschreibung einer Verbesserung des Fehlers

*Hinweis*: Gehen Sie davon aus, dass die Klasse `Panda` die Operationen `double getWeight()` und `void feed(final Bamboo bamboo)` bereitstellt.

```
// package and imports
/**
 * A zoo that takes care of pandas.
 */
public class PandaZoo {
    private String ? name;
    private final Set<Panda> pandas;

    /*
     * Create a new zoo with out any pandas.
     */
    public constructor PandaZoo(String s) {
        if (s == null || s.equals("") == true) {
            throw new IllegalArgumentException("name must not be null or
                empty");
        }
        this.name = s;
        this.pandas = new HashSet<>();
    }

    /**
     * Add a panda to the zoo, if the panda is not yet part of the zoo.
    */
    public boolean addPanda(Panda panda) {
        if (panda == null || !this.pandas.contains(panda)) {
            throw new IllegalArgumentException("Panda cannot be null");
        }
        this.pandas.add(panda);
        return true;
    }

    /**
     * Find all pandas in the zoo that are heavier than 100 kilogram.
     */
    default List<Panda> 100KiloPandasAndHeavierAmongAllPandasOfTheZoo() {
        Set<Panda> pandasHeavierThan100Kilos = new HashSet<>();
        for (Panda element : this.pandas)
            if (element.getWeight() > 100.00) {
                pandasHeavierThan100Kilos. add(element);
            }
        return pandasHeavierThan100Kilos;
    }

    /**
     * Feed all pandas in the zoo.
     */
    public void %fedPandas() {
        for(int i = 0, i < pandas.size(), i++)
        {
            pandas.get(i).feed();
        }
    }

    /**
    * Calculate the sum of the weight of all pandas in the zoo.
    *
    * @return total weight of all pandas in the zoo.
    */
   public int sumOfPandaWeight() {
        int sum;
        for(int i = 0;; i++)
           sum += pandas.get(i).getWeight();
        return sum;
   }
}
```

Listing 1: Quellcode der Klasse `PandaZoo`.

**Aufgabe 3** Äquivalenzklassen cont'd

Diese Aufgabe ist die Fortsetzung der Aufgabe 1.

(a) Definieren Sie Äquivalenzklassen für die Operation `abs(int a)`[4]. Notieren Sie jeweils eine Beschreibung der Äquivalenzklassen, einen Repräsentanten und das für diesen zu erwartende Ergebnis analog zu Tabelle 1.

(b) Öffnen Sie die Klasse `MathTest`. Implementieren Sie Testfälle für die in Teilaufgabe (a) identifizierten Äquivalenzklassen. Führen Sie die Testfälle aus.

*Hinweis:* Auch in dieser Teilaufgaben testen Sie eine fehlerhafte Reimplementierung der Operation. Es gibt einen Fehler zu finden!

(c) Definieren Sie Äquivalenzklassen für die Operation `subtractExact`[5]. Notieren Sie jeweils eine Beschreibung der Äquivalenzklassen, einen Repräsentanten und das für diesen zu erwartende Ergebnis analog zu Tabelle 1.

(d) Öffnen Sie die Klasse `MathTest`. Implementieren Sie Testfälle für die in Teilaufgabe (c) identifizierten Äquivalenzklassen. Führen Sie die Testfälle aus.

*Hinweis:* Auch in dieser Teilaufgabe testen Sie eine fehlerhafte Reimplementierung der Operation. Es gibt einen Fehler zu finden! Allerdings ist der Fehler so gravierend, dass vermutlich mehr als nur einer Ihrer Testfälle fehlschlägt.

[4] `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html#abs(int)`

[5] `https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html#subtractExact(int, int)`

# 2 Teil B - Heimarbeit

**Aufgabe 1** Unit-Tests (PDF, `.java`)

In dieser Aufgabe geht es um die Definition von Testfällen, den Begriff der Testabdeckung, sowie die Erstellung und Ausführung von Tests mittels JUnit 5.

(a) *(9 Punkte)* Es sollen Black-Box-Testfälle für den folgenden Konstruktor der Klasse `Hamster` definiert werden. Ziel ist die Abdeckung des Eingabebereichs basierend auf Äquivalenzklassen.

```
/**
 * Create and initialize a new hamster object with the given
 * parameters.
 * @param territory The territory this hamster lives in.
 * @param location The location in the territory, where the
 *                 hamster starts.
 * @param direction The direction in which the hamster looks
 *                  initially.
 * @param newGrainCount The number of grain objects initially
 *                      placed into the hamster's mouth.
 */
public Hamster(final Territory territory, final Location location,
               final Direction direction, final int newGrainCount)
```

Geben Sie sechs Testfälle an, indem Sie die Äquivalenzklasse kurz beschreiben, sowie einen Repräsentanten (Beispielwerte für alle Parameter des Konstruktors) und das erwartete Ergebnis definieren.

(b) *(4 Punkte)* Der Quellcode der Operation `putGrain` der Klasse `Hamster` sieht wie folgt aus:

```
/**
 * Drop a random grain object from the hamster's mouth.
 * @throws MouthEmptyException when the hamster does not carry
 * any grain.
 */
public void putGrain() {
    if (this.internalHamster.getGrainInMouth().isEmpty()) {
        throw new MouthEmptyException();
    }
    this.game.processCommandSpecification(
            new PutGrainCommandSpecification(this.internalHamster,
                this.internalHamster.getGrainInMouth().get(0)));
    }
```

Geben Sie eine Menge verschiedener Testfälle an, mit der eine vollständige Anweisungsüberdeckung erreicht wird. Begründen Sie kurz.

(c) *(1 Punkt)* Im Paket `de.unistuttgart.iste.sqa.pse.sheet12.homework` finden Sie die Klasse `TestHamster`. Das Paket befindet sich im Ordner *src/test/java*. Die Klasse enthält bereits ein JUnit-Test für den Hamster (`testConfiguredHamsterOnTerritory`). Machen Sie sich mit dem Aufbau des Tests vertraut. Führen Sie den Test mittels JUnit aus. Sie werden feststellen, dass der Test fehlschlägt, was an einem Fehler im Testcode liegt. Beheben Sie den Fehler. Überprüfen Sie Ihre Korrektur durch erneute Ausführung, welche nun erfolgreich sein sollte.

(d) Es sollen Testfälle für die Operationen `move()` und `pickGrain()` der Klasse `Hamster` definiert und implementiert werden.

i. *(6 Punkte)* Geben Sie Testfälle an, indem Sie den benötigten Zustand des Hamsters vor der Ausführung der getesteten Operation und das erwartete Ergebnis beschreiben. Geben Sie zwei Testfälle für jede Operation an.

ii. *(5 Punkte)* Implementieren Sie für die Operationen `move()` und `pickGrain()` jeweils einen Ihrer Testfälle mittels JUnit. Implementieren Sie in der Klasse `TestHamster`. Erzeugen Sie wo nötig, nach dem Schema des Beispieltests ein geeignetes Territorium.

**Aufgabe 2** Lambdas und Streams

In dieser Aufgabe geht es um Streams und Lambda-Ausdrücke.

Im Paket `de.unistuttgart.iste.sqa.pse.sheet12.homework.hamsterclub` finden Sie die Klasse `HamsterClub` und das Interface `Club`. Die Klasse `HamsterClub` modelliert einen Verein und besitzt eine Liste an Mitgliedern der Klasse `ClubMember`. Der Verein besitzt verschiedene Operationen, um seine Vereinsmitglieder zu analysieren. Ihre Aufgabe ist es, die noch leeren Operationen der Klasse `HamsterClub` mit Hilfe von Streams und Lambda-Ausdrücken wie unten beschrieben zu implementieren.

*Hinweis:* Verwenden Sie dabei ausschließlich Streams und Lambda-Ausdrücke.

*Hinweis:* Die genauen Spezifikationen zu den einzelnen Operationen finden Sie im `Club` Interface.

*Hinweis:* Die `ClubMember` Klasse dient hier lediglich als 'Container' und sollte von Ihnen nicht modifiziert werden. Arbeiten Sie ausschließlich in `HamsterClub`.

*Hinweis:* Die Korrektheit Ihrer Implementierung können Sie mit den JUnit Tests in `HamsterClubTest` im Paket `de.unistuttgart.iste.sqa.pse.sheet12.homework.hamsterclub` unter *src/test/java* verifizieren.

(a) *(2 Punkte)* Implementieren Sie die Operation `getMembersWithUnpaidFees()`. Die Operation git eine Liste alle Mitglieder, die ihren Beitrag noch nicht gezahlt haben, zurück.

(b) *(3 Punkte)* Implementieren Sie die Operation `getNameOfMembers()`. Die Operation die Namen aller Mitglieder in einer alphabetisch sortierten Liste zurückgeben.

(c) *(3 Punkte)* Implementieren Sie die Operation `getTotalContributions()`. Die Operation soll die Summe aller Beiträge aller Mitglieder zurück geben.

(d) *(3 Punkte)* Implementieren Sie die Operation `applyDiscount()`. Die Operation soll die Beiträge aller Mitglieder über 60 Jahre um 10% reduzieren. Bei mehrfachem Aufrufen der Operation soll die Reduktion multiplikativ wirken.

(e) *(4 Punkte)* Ein weiterer Anwendungsbereich für Java Lambda-Ausdrücke ist die Implementierung von Functional-Interfaces.

Implementieren die Operation `getOldestMember()`. Die Operation soll das älteste Mitglied zurückgeben. Falls der Club keine Mitglieder hat, soll eine `NoSuchElementException` geworfen werden. Falls es mehrere älteste Mitglieder gibt, soll das alphabetisch letzte gewählt werden. Bei zwei gleichaltrigen Mitgliedern Tadeus und Siegbert soll es also zur Entscheidung (Siegbert > Tadeus) kommen.

*Hinweis:* Die Klasse `ClubMember` implementiert die Schnittstelle `Comparable`.

**Aufgabe 3** Polymorphe Zuweisungen und dynamisches Binden

In diese Aufgabe vertiefen Sie Ihre Kenntnisse zu polymorphen Zuweisungen und dynamischem Binden. Dazu sind die Klassen aus Listing 2 gegeben. Machen Sie sich mit den Klassen vertraut. Bearbeiten Sie dann die folgende Teilaufgabe.

*Hinweis:* Die dieser Aufgabe verwendeten Bezeichner sind stilistisch grauenvoll und eignen sich lediglich zu Demonstrationszwecken. Verwenden Sie solche Bezeichner niemals in einem echten Programm.

```
1  interface Zipper {
2    public void close();
3  }
4
5  interface Clothing {
6    public void wash();
7  }
8
9  abstract class Pants implements Clothing {
10   @Override
11   public void wash() { /*Implementation*/ }
12 }
13
14 class Shorts extends Pants {
15   @Override
16   public void wash() { /*Implementation*/ }
17
18   public void mend(final Zipper zipper) { /*Implementation*/ }
19
20 }
21
22 class Jumper extends Pants implements Zipper {
23   @Override
24   public void close() {/*Implementation*/ }
25
26 }
```

Listing 2: Ausschnitte verschiedener Klassen und Interfaces. Einige Stilregeln wurden zugunsten einer kompakteren Notation missachtet.

```java
final Zipper zipper1 = new Jumper();
final Clothing clothing = new Pants();
final Pants pants = new Shorts();
final Shorts shorts1 = new Shorts();
final Jumper jumper = new Jumper();


final Zipper zipper2 = shorts1;
final Zipper zipper3 = jumper;
final Shorts shorts2 = pants;

zipper1.close();
clothing.wash();
pants.mend(zipper1);

shorts1.mend(jumper);
shorts1.mend(shorts1);
jumper.equals(zipper3);
```

Listing 3: Code Schnipsel mit Fehlern.

(a) *(5 Punkte)* Welche Zeilen aus Listing 3 enthalten statisch semantische Fehler? Begründen Sie für jede ausgewählte Zeile, warum sie fehlerhaft ist.