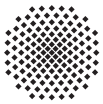

Universität Stuttgart

Software Quality and Architecture Group

Übungsblatt 11

Programmierung und Softwareentwicklung (WS 2024/2025)

Abgabe: Fr. 24.01.2025, 23:59 Uhr — Besprechung: KW 04

- 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 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: PSE11-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 Lambdas und Streams sowie über Gültigkeitsbereiche vertiefen. Des Weiteren werden Sie den Umgang mit UML Diagrammen, Clean Code und Fehlertypen weiter üben.

Punkte: Dieses Übungsblatt enthält zwei Teile. In Teil B können Sie bis zu 50 Punkte erzielen. Zum Bestehen des Blatts benötigen Sie mindestens 25.0 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-11`
- Stellen Sie sicher, dass Sie Übungsblatt 10 absolviert haben, sowie alle Software installiert und funktionsfähig ist (IDE (Eclipse, IntelliJ, VSCode,...) und Java 21).
- Machen Sie sich mit einem der unter *Unterlagen* genannten Tools zum Modellieren von UML Diagrammen vertraut.

Unterlagen:

- Git-Repositories: `https://github.com/SQA-PSE-WS-2024-2025/`
- Dokumentation des Hamstersimulators: `https://tinyurl.com/5yx654w8`
- Tools zum Modellieren von UML Diagrammen: `https://mermaid.js.org/`, `https://app.diagrams.net/`, `https://hylimo.github.io/` (nur Klassendiagramme)

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 Lambdas

In dieser Aufgabe geht es um Lambda Ausdrücke, Funktionale und funktionale Schnittstellen.

(a) Betrachten Sie die untenstehenden Lambda Ausdrücke. Bestimmen Sie für jeden Lambda Ausdruck das `FunctionalInterface`, das in Java als Typ des Funktionals verwendet wird. Schreiben Sie Ihre Ergebnisse auf. Notieren Sie die funktionalen Schnittstellen inklusive der Typparameter, also z.B `Consumer<Territory>`.

Hinweis: Einige der gegebenen Lambda Ausdrücke sind gekürzt notiert, sodass die Typen nicht eindeutig zu bestimmen sind. Notieren Sie in diesen Fällen einen der möglichen Typen.

i. `territory -> {Hamster ham = territory.getDefaultHamster(); ham.move();}`
ii. `(a, b) -> a+b`
iii. `(Hamster ham) -> ham.frontIsClear() && ham.grainAvailable()`
iv. `(terr) -> terr.getTotalGrainCount() + terr.getTotalHamsterCount()`
v. `() -> {return List.of(Location.from(0, 0), Location.from(1, 1));}`
vi. `(Hamster ham, List<Hamster> hamsters) -> hamsters.add(ham)`

Aufgabe 2 Lambdas und Streams

In dieser Aufgabe sollen Sie die Konzepte der Collection-Streams und Lambdas in Java vertiefen und anwenden. Alle relevanten Klassen für diese Aufgabe liegen im Paket `de.unistuttgart.iste.sqa.pse.sheet11.presence.collectionstreams`.

(a) In der Klasse `StreamsAndFilterExercise` finden Sie die `main`-Operation und die Operationen, die Sie im Folgenden füllen sollen. Machen Sie sich zunächst mit der Implementierung vertraut. Schauen Sie sich dabei die Klassen `StudentRecord`, `Student` und `Exam` an.

(b) `printAllStudentNames(...)`: Nutzen Sie Streams, um sich die Namen aller Studenten auf der Konsole ausgeben zu lassen.

(c) `printNumberOfStudentsOlderThan(...)`: Nutzen Sie Streams, um sich die Anzahl der Studenten auszugeben, welche älter als das (als Parameter) gegebene Alter sind.

(d) `printStudentNamesOlderThan(...)`: Nutzen Sie Streams, um sich die Namen der Studenten ausgeben zu lassen, welche älter als das (als Parameter) gegebene Alter sind.

(e) `printStudentNamesOlderThanAndFailedExam(...)`: Nutzen Sie Streams, um sich die Namen aller Studenten ausgeben zu lassen, welche älter als das (als Parameter) gegebene Alter sind und die (als Parameter) gegebene Klausur nicht bestanden haben.

(f) **Herausforderung:** Implementieren Sie die Teilaufgaben (b)-(e) ohne die Nutzung von Streams und Lambdas. Verwenden Sie hierzu die `...Challenge()`-Operationen.

Aufgabe 3 Wir sind hier nämlich alle Paule.

In dieser Aufgabe geht es um Gültigkeitsbereiche von Variablen.

Dazu ist das Spiel `WeAreAllPauleHereHamsterGame` gegeben, in dem es viele Hamster gibt, die alle Paule sind. Listing 1 zeigt eine gekürzte Version der Klasse. Die vollständige Klasse finden Sie im Paket `de.unistuttgart.iste.sqa.pse.sheet11.presence.scopes`.

Das Spielfeld hat nach der Initialisierung den in Abbildung 1 abgebildeten Zustand. Der blaue Hamster links oben an der Position (1,1) ist der default Hamster (also `paule`).

Hinweis: Die Zeilennummern in den folgenden Teilaufgaben beziehen sich immer auf Listing 1.

```
1  public class WeAreAllPauleHereHamsterGame extends SimpleHamsterGame {
2      private final Hamster paule;
3
4      public WeAreAllPauleHereHamsterGame() {
5          // initialize territory, including default hamster paule
6          this.paule = new Hamster(game.getTerritory(), Location.from(3,1),
               Direction.EAST, 0);
7      }
8
9      @Override
10     protected void run() {
11         Consumer<Hamster> consumer = (Hamster paule) -> {paule.move();
               this.paule.move();};
```

```
12          List<Hamster> paules = List.of(new Hamster(game.getTerritory(),
                Location.from(5, 1), Direction.EAST, 0));
13
14          paule.move();
15          for (Hamster paule : paules) {
16              paule.move();
17          }
18          paule.move();
19
20          final Hamster paule = paules.get(0);
21          paule.move();
22          this.paule.move();
23          super.paule.move();
24
25          doMovement();
26          doMovement(paule);
27
28          consumer.accept(paule);
29      }
30
31      private void doMovement() {
32          paule.move();
33          this.paule.move();
34      }
35
36      private void doMovement(final Hamster paule) {
37          paule.move();
38          this.paule.move();
39      }
40  }
```

Listing 1: Auszug aus einer Klasse, in der für fast alle Variablen der Bezeichner *paule* gewählt wurde.

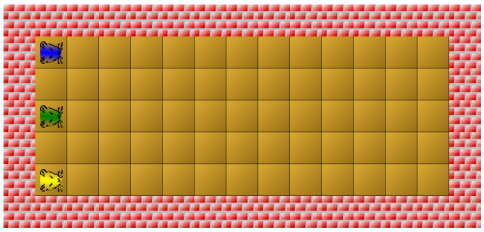

Abbildung 1: Alle Hamster Paule auf ihren Startpositionen.

(a) Welche Hamster werden durch die Aufrufe `paule.move()`, `this.paule.move()`, `super.paule.move()` in den Zeilen 21, 22 und 23 bewegt? Nennen Sie jeweils die Farbe der Hamster, und begründen Sie, warum die Aufrufe genau diese Hamster bewegen. Halten Sie Ihre Ergebnisse schriftlich fest.

(b) Welche Hamster werden durch die Aufrufe von `doMovement` in Zeile 25 und 26 bewegt? Nennen Sie jeweils die Farbe der Hamster, und begründen Sie, warum die Aufruf genau diese Hamster bewegen. Halten Sie Ihre Ergebnisse schriftlich fest.

(c) Wie Sie in den vorhergehenden Teilaufgaben beobachten konnten, sind die Bezeichner in der Klasse `WeAreAllPauleHereHamsterGame` sehr schlecht gewählt. Ersetzen Sie alle in dieser Klasse definierten Bezeichner `paule` durch besser gewählte Bezeichner.

Hinweis: Der default Hamster, also der in der Elternklasse gewählte Bezeichner, soll weiterhin `paule` bleiben.

(d) Welche Hamster werden durch den Aufruf `consumer.accept(paule)` in Zeile 28 bewegt? Nennen Sie jeweils die Position oder Farbe der Hamster, und begründen Sie, warum der Aufruf genau diese Hamster bewegt. Halten Sie Ihre Ergebnisse schriftlich fest.

(e) Wenn Sie die Zeile 11 hinter die Zeile 20 verschieben, sodass der Lambda Ausdruck, der ursprünglich in Zeile 11 stand, erst nach der Deklaration der lokalen Variable `paule` kommt, ist das Programm nicht mehr korrekt. Klassifizieren Sie den Fehler und begründen Sie Ihre Entscheidung.

2 Teil B - Heimarbeit

Aufgabe 1 UML (PDF, `.java`)

Im Aufbaustrategiespiel „Die Siedler“ von 1993 besiedeln Siedler ein unbekanntes Land. In der folgenden Aufgabe wollen wir die Elemente des Spiels mittels UML Klassendiagrammen modellieren und in Code gießen.

Dazu ist folgende Beschreibung von Teilen des Spiels gegeben:

„*In ‚Die Siedler‘ gibt es verschiedene Arten von Siedlern, die sich durch ihren Beruf unterscheiden. So gibt es beispielsweise Soldaten und Müller. Siedler verarbeiten Ressourcen zu neuen Ressourcen oder führen Kämpfe. Ressourcen sind beispielsweise Weizen oder Holz. Sie können in Lagerhäusern gespeichert werden. Siedler leben in Gebäuden, von denen es auch entsprechende Arten gibt. Gebäude sind durch Straßen verbunden, auf denen die Ressourcen transportiert werden. Am Ende jeder Straße können maximal fünf Ressourcen zwischengelagert werden.*“

Ein erstes Diagramm, das Teile des Modells beinhaltet, sieht wie folgt aus:

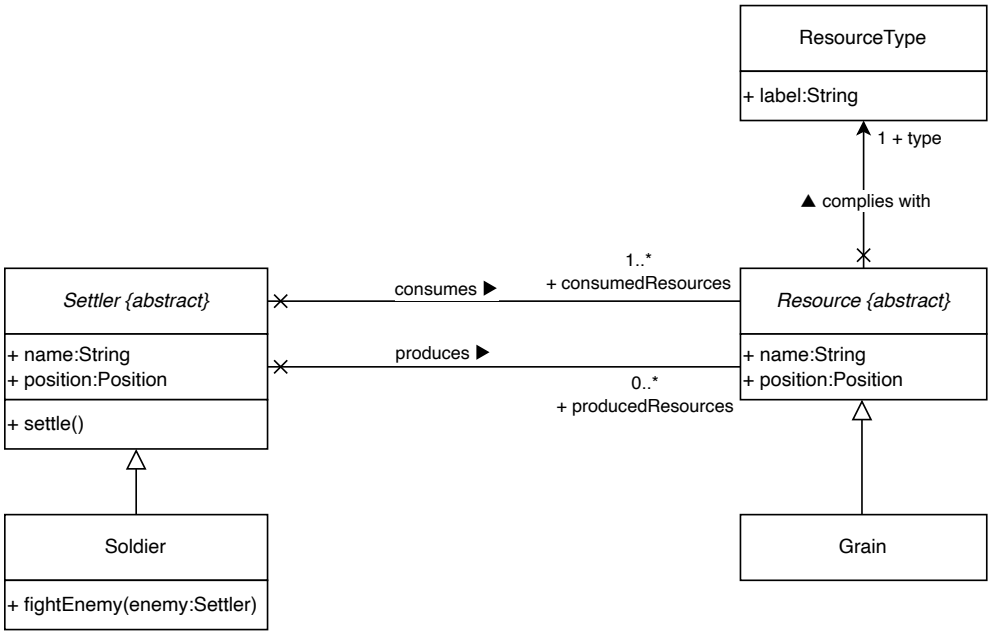

Abbildung 2: Initiales Modell.

Hinweis: Um diese Aufgabe zu Lösen, stellen wir Ihnen das initiale Modell in allen von uns vorgeschlagenen Tools zum modellieren von UML Diagrammen zur Verfügung. Das initiale Modell für *Mermaid* finden Sie hier: `https://tinyurl.com/4nt3raxk`.
Die initialen Modelle für *diagrams.net* und *HyLimo* finden Sie im Ilias im Ordner des elften Übungsblattes, im Archiv *Initiale_Modelle_Teil_B_Aufgabe_1.zip*.

Falls Sie Modellelemente nutzen, die vom Tool Ihrer Wahl nicht unterstützt werden, so tragen Sie diese bitte nachträglich in Ihre Lösung ein, z.B. von Hand.

(a) *(3 Punkte)* Ergänzen Sie im Modell aus Abbildung 2 Klassen für Müller (Miller) und Mehl (Flour). Müller haben eine Operation, um Korn (Grain) in Mehl (Flour) zu mahlen (grind).

(b) *(6 Punkte)* Ergänzen Sie eine Oberklasse für Gebäude (Building) und Unterklassen für Mühlen (Mill) und Kasernen (Barracks). Lassen Sie Soldaten in der Kaserne und die Müller in der Mühle wohnen (lives in). Jeder Siedler lebt in maximal einem der Gebäude. Jede Mühle hat exakt einen Bewohner (inhabitant), Kasernen beherbergen mehrere Bewohner (inhabitants). Gebäude und ihre Bewohner sollen gegenseitig navigierbar sein.

In den Gebäuden können jeweils bis zu fünf Ressourcen gelagert (stores) werden. Ressourcen sollen es nicht ermöglichen, zu den Gebäuden, in denen sie lagern, zu navigieren.

(c) *(4 Punkte)* Ergänzen Sie eine Klasse für Siedlungen (Settlement). Siedlungen bestehen aus mindestens zwei Gebäuden. Einzelne Gebäude können noch keine Siedlung bilden. Wählen Sie eine passende Assoziation zwischen Gebäuden und Siedlung. Begründen Sie Ihre Entscheidung in 1-2 Sätzen. Notieren Sie die Begründung als Kommentar im Ihrem Klassendiagramm.

(d) *(10 Punkte)* Übersetzen Sie die im initialen Modell (vgl. Abbildung 2) auftretenden Klassen `Settler`, `Soldier`, `Ressource` und `Grain` in Javacode. Erstellen Sie die neuen Klassen im Paket `de.unistuttgart.iste.sqa.pse.sheet11.homework.settler`. Verwenden Sie, wo sinnvoll die in diesem Paket bereits vorgegebenen Klassen `Position` und `RessourceType`. Ignorieren Sie alle Diagrammelemente, die Sie für die Teilaufgaben (a) bis (c) ergänzt haben.

Beachten Sie die folgenden Hinweise:

- Auf die konkreten Implementierung der Operationen `settle` und `fightEnemy` dürfen Sie verzichten. Schreiben Sie statt dessen ein TODO Kommentar (`// TODO`) in die Rümpfe der Operationen.
- Alle Klassen sollen unveränderlich sein. Nutzen Sie, wo sinnvoll und möglich, Recordtypen.
- Alle Attribute sollen mittels des in Java üblichen Prinzip des gleichförmigen Zugriffs (UAP) umgesetzt werden.
- Nutzen Sie die Vorteile Ihrer IDEs zum Generieren von Konstruktoren, Getter- und Setter-Operationen und ähnlichem (siehe Übungsblatt 06).

Aufgabe 2 Clean Code (PDF, `.java`)

In dieser Aufgabe geht es um Clean Code. Öffnen Sie die Klasse `SomethingWithCalendars` aus dem Paket `de.unistuttgart.iste.sqa.pse.sheet11.homework.cleancode`. Bearbeiten Sie dann die folgenden Teilaufgaben.

Hinweis: Die Klasse `Calender` ist Teil der JBCL, das heißt Sie finden die Dokumentation der Klasse im Internet.

Hinweis: Das Programm erwartet unter anderem eine Eingabe über die Konsole. Achten Sie darauf, dass Sie in Ihrer IDE die entsprechende Ansicht geöffnet haben, währen Sie das Programm ausführen.

(a) *(4 Punkte)* Lesen Sie das Programm und beschreiben Sie, was es wohl machen soll. Geben Sie für 2 Beispieleingaben an, welche Ausgaben Sie erwarten würden.

(b) *(3 Punkte)* Geben Sie eine Beispieleingabe an, die das Programm mit einem unerwarteten Ergebnis quittiert. Begründen Sie Ihre Antwort.

(c) *(6 Punkte)* Nennen Sie genau drei Verletzungen von CleanCode- oder Style-Regeln und begründen Sie jeweils mit einem Beispiel aus dem Code.

(d) *(6 Punkte)* Implementieren Sie ein Programm, welches die in Teilaufgaben a vermutete Funktionalität realisiert. Dabei müssen Sie nicht unbedingt das existierende Programm retten, sondern können auch den Inhalt der Klasse löschen und völlig neu beginnen. Ebenso müssen nicht exakt die gleichen Ausgaben bei gleichen Eingaben rauskommen. Es reichen inhaltlich sinnvolle Ausgaben. Achten Sie dabei selbst auf sauberen Code und unsere Style-Regeln. Verwenden Sie wo sinnvoll weitere Klassen aus der JBCL.

Aufgabe 3 Vererbung (PDF)

In diese Aufgabe vertiefen Sie Ihre Kenntnisse zu Vererbung. Dazu sollen Sie aus einer Menge an natürlichsprachlichen Bedingungen eine Vererbungshierarchie ableiten und skizzieren.

Zum Beispiel lässt sich aus den folgenden Bedingungen die Klassenhierarchie aus Abbildung 3 ableiten.

- Es gibt die Klassen Bix, Foo, Quaz und das Interface Bar.
- Foo ist ein Subtyp von Bar.
- Quaz ist ein Subtyp von Foo.
- Obwohl Foo kein Interface ist, können keine Instanzen von Foo erstellt werden.

Hinweis: Die in dieser Aufgabe verwendeten Bezeichner sind stilistisch grauenvoll. Verwenden Sie solche Bezeichner niemals in einem echten Programm.

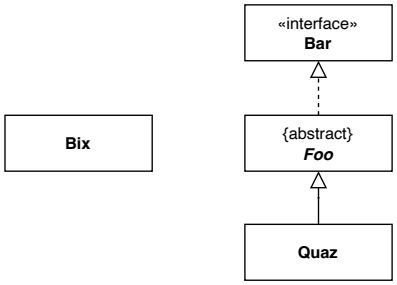

Abbildung 3: Klassenhierarchie, die die im Aufgabetext gegebenen Bedingungen erfüllt. Abstrakte Klassen sind mit {*abstract*} markiert und Interfaces mit «*interface*». Gestrichelten Pfeilen stellen `implements`-Beziehungen dar, durchgezogenen Pfeilen stellen `extends`-Beziehungen dar.

(a) *(8 Punkte)* Skizzieren Sie eine Klassenhierarchie, die die folgenden Bedingungen 1. bis 9. erfüllt. Die Lösung soll minimal sein, das heißt, sie soll keine überflüssigen Klassen, Schnittstellen oder Vererbungsbeziehungen enthalten. Verwenden Sie nur die in den Bedingungen genannten Klassen und Interfaces. Verwenden Sie die in Abbildung 3 verwendeten Notation für (abstrakte) Klassen, Interfaces sowie `extends`- und `implements`-Beziehungen.

1. A und B sind abstrakte Klassen.
2. Es kann sowohl Instanzen vom Typ D als auch vom Typ G geben.
3. H und F implementieren kein Interface, weder direkt noch indirekt.
4. A ist Subtyp von C und F.
5. D ist Subtyp von A, E und H.
6. E ist kein Subtyp von A.
7. G ist Subtyp von C.
8. A ist kein Subtyp von einer der angegebenen Klassen.
9. Alle Objekte können Variablen vom Typ C zugewiesen werden.
10. D implementiert genau ein Interface direkt.
11. Es kann keine Instanzen geben, die einer Variable vom Typ B zugewiesen werden können.