



# Übungsblatt 10

Programmierung und Softwareentwicklung (WiSe 2019/2020)

Abgabe: Fr. 10.01.2020, 23:59 Uhr — Besprechung: KW 4

- Bitte lösen Sie die Übungsaufgabe in **Gruppen von 2 Studenten**.
- Dieses Übungsblatt besteht aus zwei Teilen (A+B). Teil A ist in der Präsenzübung zu lösen. Teil B ist in Heimarbeit (Gruppe von 2 Studenten) zu lösen und rechtzeitig abzugeben.
- 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. **SortingHamsterGame.java**
2. **ReplacementHamsterGame.java**
3. **Übungsblatt10-[Nachnamen der Teammitglieder]-Aufgabe2-Aufgabe3.pdf**

**Lernziel:** Dieses Übungsblatt dient dazu, dass Sie die in der Vorlesung vorgestellten Konzepte zu Vererbung und Clean Code vertiefen und anwenden. Zudem kennen Sie bereits ein umfangreiches Set von Stilrichtlinien, welche Sie anwenden, üben und vertiefen sollen.

**Vorbereitung:** Stellen Sie sicher, dass Sie sich mit den Vorlesungsunterlagen zum Thema Vererbung sowie Clean Code vertraut gemacht haben und die Übungsblätter 8 und 9 absolviert haben.

**Punkte:** Dieses Übungsblatt enthält zwei Teile. Teil A mit 3 Aufgaben und Teil B mit 2 Aufgaben. Im Teil B können Sie bis zu 71 Punkte erzielen.

**Style:** Bitte halten Sie die in der Vorlesung vorgestellten Style-Regeln ein. Dazu gehören unter anderem JavaDoc, Vor- und Nachbedingungen, Prüfung der Vorbedingungen mittels offensiver (mittels Asserts) und defensiver Checks (mittels Exceptions), Schleifeninvarianten und -varianten (mittels natürlicher Sprache oder JML im Kommentarblock über der Schleife), Sichtbarkeiten oder Read-Only Restriktionen. Der Style Ihrer Implementierung wird mit bis zu 50% bewertet.

**Unterlagen:** Alle relevanten Unterlagen, für die Bearbeitung des Übungsblatts, finden Sie wie gehabt in unserem git-Repository: <https://github.com/RSS-PSE-WS1920/>

**Viel Erfolg!**

# 1 Teil A - Präsenzaufgaben

## Aufgabe 1 Overriding vs. Overloading

Ziel dieser Aufgabe ist es, nochmal die Unterschiede zwischen Überschreiben und Überladen zu üben.

- (a) Betrachten Sie die Klassenhierarchie von `Animal` im Paket `...exercise10.override`. Zeichnen Sie die Vererbungshierarchie.
- (b) Implementieren Sie den geg. Pseudocode in der Klasse `OverrideUsageTest`. Bevor Sie das Programm ausführen, diskutieren Sie das zu erwartende Ergebnis. Danach testen Sie es. Nutzen Sie Begriffe wie Polymorphie, statischer/dynamischer Typ und dynamische Bindung, um das Verhalten zu erklären.
- (c) Implementieren Sie den geg. Pseudocode in der Klasse `OverloadedUsageText`. Bevor Sie das Programm ausführen, diskutieren Sie das zu erwartende Ergebnis. Danach testen Sie es. Nutzen Sie Begriffe wie Polymorphie, statischer/dynamischer Typ und dynamische Bindung, um das Verhalten zu erklären.

## Aufgabe 2 Polymorphie und Collections

In dieser Aufgabe testen wir einmal, was wir in der Vorlesung zu Polymorphie und den verschiedenen Implementierungen des List Interfaces gehört haben.

- (a) Machen Sie sich mit der API der Klasse `Timer` aus dem Paket `...exercise10.polymorphism` vertraut (nicht der `Timer` aus der Base Class Library).
- (b) Implementieren Sie, unter Benutzung von `Timer` Objekten, eine Testoperation, die das Einfügen von 100.000 Strings am Ende eines beliebigen Listenobjekts ausführt. Nutzen Sie `Timer`, um zu bestimmen, wie lange dies braucht. Bevor Sie den folgenden Test ausführen, stellen Sie eine Hypothese auf, welche der vorgegebenen Listen wohl die schnellste ist. Rufen Sie die Operation je einmal mit Instanzen von `LinkedList`, `ArrayList` und `Vector` auf.
- (c) Analog zur vorherigen Aufgabe, implementieren Sie eine Operation zum Einfügen von 100.000 Strings am Anfang einer beliebigen Liste. Experimentieren Sie wieder wie oben.
- (d) Analog zur vorherigen Aufgabe, implementieren Sie eine Operation zum Holen aller 100.000 Elemente über ihren Index. Experimentieren Sie wieder wie oben.

### Aufgabe 3 Clean Code

In dieser Übungsaufgabe lernen wir, Dirty Code in Clean Code zu verwandeln.

```
public void printSortedByAge() {
    Iterator iter = customers.iterator();
    ArrayList list = new ArrayList();
    while (iter.hasNext()) {
        Customer customer = iter.next();
        list.add(customer.getAge() + "_" + customer.getLoginName());
    }
    Collections.sort(list);
    Iterator iter2 = list.iterator();
    while (iter2.hasNext()) {
        try {
            String str = iter2.next();
            System.out.println(findCustomerByName(str.substring(str.lastIndexOf("_") + 1)));
        } catch (CustomerNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Listing 1: Dirty Code

- (a) Versuchen Sie den Code aus Listing 1 zu verstehen. Diskutieren Sie in Ihrer Gruppe, was der Code wohl tun könnte.
- (b) Identifizieren Sie Verbesserungen, die am Code durchzuführen wären.
- (c) **Herausforderung:** Implementieren Sie den Algorithmus in Clean Code. Implementieren Sie dazu auch die fehlenden Klassen und Operationen.

## 2 Teil B - Heimarbeit

### Aufgabe 1 Paule ordnet

In dieser Aufgabe geht es erneut darum, Ihre Fähigkeiten zum Thema Algorithmen und Datenstrukturen unter Beweis zu stellen. Dazu geht es im Speziellen darum, Objekte in einer Datenstruktur neu zu ordnen (z.B. alphabetisch oder numerisch auf- oder absteigend).

Dazu ist das folgende Szenario aus Abbildung 1 gegeben. Es zeigt das initiale Territorium. Paulas Aufgabe ist es nun, das Territorium abzulaufen und dabei die Körner auf dem Boden aufzusammeln und sich zu merken, wie viele Körner auf jedem der Felder lagen. Auf der anderen Seite des Territoriums soll er sich umdrehen und zurücklaufen. Dabei soll er die Anzahl der Körner, die er pro Feld aufgesammelt hat, sortiert von der höchsten Zahl zur niedrigsten Zahl, wieder ablegen (siehe Abbildung 2).

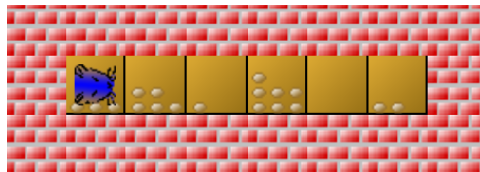


Abbildung 1: initiales Territorium vor dem Sortieren

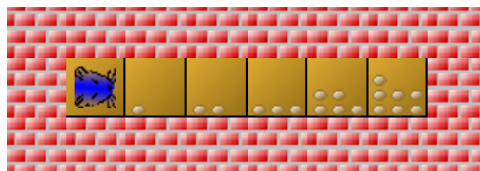


Abbildung 2: Territorium mit aufsteigend sortierten Körnern

- (a) (3 Punkte) Beschreiben Sie zunächst in natürlicher Sprache (oder Pseudocode) einen Algorithmus, der beliebige natürliche Zahlen in eine Reihenfolge bringt.  
*Hinweis:* Dies wurde nicht in der Vorlesung behandelt. Sie müssen sich dies also selbst erarbeiten.
- (b) (2 Punkte) Schauen Sie sich die Dokumentation zu den in Java vorhandenen Collections an. Geben Sie zwei Datenstrukturen an, die geeignet sind, sortierte Inhalte zu speichern und bereits eine Implementierung zum Sortieren enthalten. Begründen Sie Ihre Antwort kurz.
- (c) (27 Punkte) Setzen Sie das oben gegebene Szenario um. Nutzen Sie dafür die Klasse `SortingHamsterGame`. Nutzen Sie eine geeignete Datenstruktur und benutzen Sie die entsprechende Operation, die bereits von der Datenstruktur zur Verfügung gestellt wird.
- (d) (4 Punkte) *Bonus:* Implementieren Sie im Folgenden den beschriebenen Sortieralgorithmus aus der Teilaufgabe a) selbst.

**Aufgabe 2** Paule dreht am Rad

In der folgenden Aufgabe steht Paule in einem Territorium mit  $n \times n$  Feldern ohne Innenwände, aber umgrenzt von Außenwänden. Auf den Feldern liegen beliebig viele Körner. Im Folgenden sollen Sie einen Algorithmus erarbeiten, der den Inhalt des Territoriums um 90 oder 180 Grad dreht (siehe Abbildung 3).

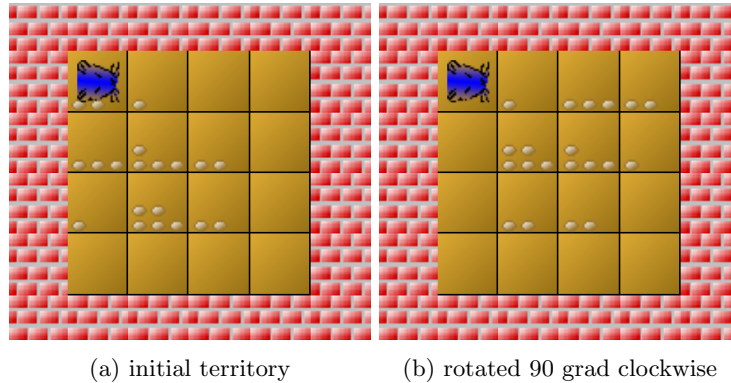


Abbildung 3: Beispielterritorium

- (a) (3 Punkte) Beschreiben Sie einen Algorithmus in natürlicher Sprache (oder Pseudocode), um die Aufgabe zu lösen.
- (b) (3 Punkte) Nennen Sie eine geeignete Datenstruktur für diese Aufgabe und beschreiben Sie, warum diese geeignet ist.
- (c) (27 Punkte) Füllen Sie die Operation `rotate90` in der Klasse `ReplacementHamsterGame`, sodass die Körner am Ende um 90 Grad gedreht sind. Erstellen Sie Hilfsoperationen, wenn benötigt.
- (d) (2 Punkte) Füllen Sie die Operation `rotate180` in der Klasse `ReplacementHamsterGame`, sodass die Körner am Ende um 180 Grad gedreht sind.

*Hinweis:* Vermeiden Sie Codeduplikate und ignorieren Sie Performanceaspekte.

### 3 Teil L - Lehramtsteil

Dieser Teil enthält eine Präsenzaufgabe die speziell für Lehramtsstudierende konzipiert ist. Das bedeutet allerdings nicht, dass “Nicht-Lehramtsstudierende” diese nicht auch bearbeiten dürfen. Sie bezieht sich auf den Vorlesungsstoff und die Übungsaufgaben, betrachtet diese aber aus einer didaktischen Perspektive.

#### Aufgabe 1 Debugging im Unterricht

Während des Entwicklungsprozesses wird man als Entwickler immer wieder auf die Debugging-Werkzeuge der genutzten IDE zurückgreifen. Die Werkzeuge helfen dabei, Fehler im Code einfacher ausfindig zu machen, um diese beheben zu können. Diese Aufgabe soll Sie darüber nachdenken lassen, in wie weit man genau diese Werkzeuge auch didaktisch einsetzen könnte.

- (a) Die Klasse `BuggyHamsterGame` enthält einen Fehler. Machen Sie sich mit dem Debugging-Tool und der Debugging-Ansicht von eclipse vertraut und suchen Sie damit den Fehler in der Klasse, indem Sie folgende Schritte ausführen:
1. Setzen Sie einen Haltepunkt (“Breakpoint”) in der Zeile, die mit “// TODO: Hier Haltepunkt setzen” markiert ist. Einen Haltepunkt setzt man in eclipse in dem man einen Doppelklick links neben die Zeilennummer macht.
  2. Führen Sie die Klasse `BuggyHamsterGame` im Debug-Modus aus, indem Sie oben in der Leiste auf den Käfer klicken. Ggf. fragt eclipse Sie, ob es in die Debug-Ansicht wechseln soll – bejahen Sie dies.
  3. Das Programm sollte in der Zeile mit dem Haltepunkt die Ausführung anhalten.
  4. Sie haben eine Reihe von Kontrollfunktionen in der Werkzeugleiste oben (beschrieben von links nach rechts):

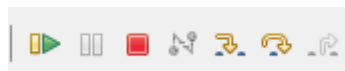



Abbildung 4: Palette mit Debugging-Werkzeugen in eclipse

- “Resume”: Setzt die Ausführung bis zum nächsten Haltepunkt fort.
  - “Suspend”: Pausiert die Ausführung.
  - “Terminate”: Beendet die Ausführung
  - “Disconnect”: Trennt die Verbindung zu einer “Remote JVM”.
  - “Step into”: Springt in die nächste Operation, die aufgerufen wird.
  - “Step over”: Führt die nächste Operation aus und hält danach wieder an.
  - “Step return”: Führt die aktuelle Operation fertig aus und hält in der aufrufenden Operation wieder an.
5. Führen Sie die aktuelle Zeile aus, indem Sie auf “Step over”  klicken.
  6. Testen Sie andere der oben gezeigten Werkzeuge aus, um den Fehler im Code zu finden.
- (b) Diskutieren Sie mit Ihrem Team, wie man die Debugging-Werkzeuge den Schülerinnen und Schülern sinnvoll erklären könnte. Die Schülerinnen und Schüler sollen danach auch den Sinn dieser Werkzeuge erkennen können.
- (c) Überlegen Sie sich mit Ihrem Team, wie man die Debugging-Werkzeuge im Unterricht noch benutzen könnte. Denken Sie auch an Anwendungen, die über die ursprünglich angedachten Funktionen hinausgehen.

### **Hinweise zu den Übungen:**

- Die Abgabe erfolgt im ILIAS.
- 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 erreicht wurden.
  - Teilnahme an min. 80% der Übungen.
  - Bestehen der Scheinklausur.

**Viel Erfolg!**