Programmierung 1

im Wintersemester 2025/26

Praktikumsaufgabe 3

Elementare Datentypen und Referenzen (Kap. 3)

Review: 19. November 2025 (DFIW) und 19. November 2025 (PI)

In dieser Praktikumsaufgabe vertiefen Sie Ihr Verständnis für den Unterschied zwischen *elementaren Datentypen* und *Referenztypen*. Die einzelnen Aufgaben führen Sie durch verschiedene Aspekte dieser Konzepte, so dass Sie die Unterschiede in der Speicherverwaltung und im Verhalten besser verstehen.

In Aufgabe 1 beginnen Sie mit der Deklaration und Zuweisung von Variablen für elementare Typen und Referenztypen und befassen sich dabei auch mit möglichen Typkompatibilitätsfehlern. In Aufgabe 2 lernen Sie, wie sich primitive Typen und Referenztypen in Bezug auf den Speicher unterscheiden und wie sich Änderungen an diesen Typen auswirken. In Aufgabe 3 untersuchen Sie den `null`-Wert, um zu sehen, wie nicht initialisierte Referenzen in Java behandelt werden. In Aufgabe 4 schließlich arbeiten Sie mit Arrays als Referenztypen und erfahren, wie das Zuweisen einer Arrayreferenz und die Veränderung von Elementen über mehrere Referenzen hinweg funktioniert.

Verwenden Sie die Metaphern aus dem Buch, wie z. B. “Becher” und “Fernbedienung”, und lassen Sie sich von den Beispielen zu eigenen Visualisierungen inspirieren, um die Konzepte anschaulich und greifbar zu machen. Diskutieren Sie Ihre Ergebnisse im Coding-Team, um Ihr Wissen zu vertiefen und von den Erkenntnissen anderer zu profitieren.

Hinweis: Die meisten Lösungen bestehen aus kurzen *Snippets*. Im Zusammenhang mit den Praktikumsaufgaben verstehen wir darunter kleine “Codeschnipsel”, mit denen Sie eine Aufgabe lösen. Schreiben Sie jeweils eine `main`-Methode in einer entsprechend benannten Klasse, um Ihre Snippets zu testen und zu überprüfen, ob sie wie erwartet funktionieren. Im Review

können Sie diese Snippets schließlich präsentieren und diskutieren. Um die Übersichtlichkeit und Wartbarkeit Ihres Codes zu verbessern, empfiehlt es sich, zumindest für jede Hauptaufgabe eine eigene Testklasse zu erstellen. Ein besonders gut geeigneter Klassenname für Aufgabe 1 ist beispielsweise `DeclarationAssignmentTestDrive`.

Heads up!

Für diese und die folgenden Praktikumsaufgaben gilt:

Konventionen und Praktiken: Halten Sie sich an die Konventionen der professionellen Softwareentwicklung und folgen Sie den bewährten Praktiken. Wählen Sie aussagekräftige Namen. Verwenden Sie Pseudo-UML, um Ihre Klassen zu entwerfen.

Und nach wie vor gilt: Lösen Sie die Aufgaben nicht allein, sondern mindestens zu zweit! Zum Beispiel mit Ihrer Coding-Partner:in. Denn in der Gruppe lernt es sich besser.

Wir als Ihre Lernbegleiter:innen unterstützen Sie gerne bei der Lösung der Aufgaben.

1 Variablendeklaration und -zuweisung

Lernziele

- ☐ Sie deklarieren Variablen für elementare Typen und Referenztypen und weisen ihnen geeignete Werte zu.
- ☐ Sie verwenden elementare Datentypen entsprechend ihrer Wertebereiche.
- ☐ Sie erkennen Fehler, die bei der Zuweisung unterschiedlicher Datentypen auftreten können.
- ☐ Sie verwenden Literale bei der Wertzuweisung und berücksichtigen dabei die Typanforderungen.

In dieser Aufgabe lernen Sie die Grundlagen der Variablendeklaration und -zuweisung in Java kennen. Sie deklarieren elementare Datentypen und Referenztypen, weisen ihnen Werte zu und unterscheiden den Wertebereich verschiedener Datentypen. Darüber hinaus setzen Sie sich mit der Kompatibilität verschiedener Datentypen auseinander. Diese Kenntnisse bilden die Grundlage für das Verständnis von Variablen in Java und deren Verwendung in Programmen.

1.1 Deklaration und Zuweisung

Deklarieren Sie Variablen der Typen `int`, `double`, `boolean` und `String` und weisen Sie ihnen Startwerte zu! Achten Sie auf aussagekräftige Namen und eine dazu passende initiale Zuweisung. Sie können sich an der folgenden Liste orientieren:

- Eine Variable vom Typ `int` mit dem Wert `42`.
- Eine Variable vom Typ `double` mit dem Wert `3.1415`.
- Eine Variable vom Typ `boolean` mit dem Wert `true`.
- Eine Variable vom Typ `String` mit dem Wert `"Hello, World!"`.

Diskutieren Sie anschließend im Coding-Team, in welchen Fällen Sie diese Typen einsetzen würden.

1.2 Wertebereich und Typwahl

Wählen Sie für die folgenden Werte die passenden elementaren Datentypen und deklarieren Sie Variablen mit diesen Werten! Überlegen Sie, welcher Typ den Wertebereich des Wertes am besten abdeckt und warum.

- `100`
- `3.14`
- `1_000_000`
- `false`

Hinweis: Verwenden Sie `byte`, `short`, `int`, `double` oder `boolean`.

Diskutieren Sie im Coding-Team Ihre Wahl und wie die Wertebereiche der verschiedenen Typen Ihre Entscheidung beeinflusst haben.

1.3 Typkompatibilität und Fehlererkennung

Deklarieren Sie eine `int`-Variable, z. B. `intValue`, mit einem Wert Ihrer Wahl und versuchen Sie dann, diese Variable einer anderen Variablen vom Typ `byte`, z. B. `byteValue`, zuzuweisen![1]

Notieren Sie den Compilerfehler und diskutieren Sie im Coding-Team, warum dieser Fehler auftritt. Überlegen Sie gemeinsam, wie sich solche Fehler in größeren Programmen auswirken könnten, wenn sie nicht vom Compiler gemeldet würden.

1.4 Arbeiten mit Literalen und Typanforderungen

Deklarieren Sie eine `long`-Variable und weisen Sie ihr eine große Ganzzahl mit dem Suffix `L` zu! Deklarieren Sie dann eine `float`-Variable und weisen Sie ihr eine Fließkommazahl mit dem Suffix `f` zu!

[1] Ohne explizites Casting, falls Sie diese Möglichkeit bereits kennen. Dazu kommen wir später.

Diskutieren Sie im Coding-Team, warum diese Suffixe nötig sind.

2 Elementartypen vs. Referenztypen

Lernziele

- ☐ Sie deklarieren und initialisieren Variablen sowohl für primitive Typen als auch für Referenztypen.
- ☐ Sie zeigen, wie sich Änderungen an einer Referenzvariablen auf alle Referenzen auf dasselbe Objekt auswirken können, während primitive Typen davon unberührt bleiben.

In dieser Aufgabe beschäftigen Sie sich mit den grundlegenden Unterschieden zwischen primitiven Datentypen und Referenztypen in Java. Sie zeigen, wie primitive Datentypen Werte direkt speichern, während Referenztypen auf Objekte im Speicher verweisen. Das Verständnis dieses Unterschieds ist wichtig, um korrekt mit Variablen in Java zu arbeiten und unerwartetes Programmverhalten zu vermeiden. Sie diskutieren diese Konzepte und wenden sie praktisch an, um ein tieferes Verständnis der Speicherverwaltung für verschiedene Datentypen zu entwickeln.

2.1 Deklaration und Initialisierung von primitiven und Referenztypen

Deklarieren und initialisieren Sie eine Variable vom Typ `int` und eine vom Typ `Dog` und weisen Sie beiden Variablen Werte Ihrer Wahl zu!

```
1  class Dog {
2
3    int size;
4    String breed;
5    String name;
6
7    void bark() {
8      System.out.println(name + " barks: Ruff! Ruff!");
9    }
10
11 }
```

Diskutieren Sie im Coding-Team, warum `Dog` ein Referenztyp ist und wie er sich von `int`

unterscheidet. Überlegen Sie gemeinsam, welche Rolle dieser Unterschied für die Speicherverwaltung spielt.

2.2 Änderungen an Referenztypen und primitiven Typen

Referenztypen und Elementartypen unterscheiden sich in ihrer Speicherstruktur und in der Art und Weise, wie sie Werte speichern. Im Folgenden machen Sie sich selbst ein Bild von diesem Unterschied.

2.2.1 Referenztypen

Deklarieren Sie zwei `Dog`-Variablen, die auf denselben Hund verweisen, ändern Sie eine Eigenschaft des Hundes über eine der beiden Variablen und überprüfen Sie, ob sich die Änderung auf die andere Variable auswirkt!

Deklarieren Sie dazu eine `Dog`-Variable, z. B. `dog1`, mit einem Hund, der die von Ihnen gewünschten Eigenschaften hat, d. h. Größe, Rasse und Name.

Deklarieren Sie dann eine zweite `Dog`-Variable und weisen Sie ihr den gleichen "Wert" zu wie der ersten `Dog`-Variablen.[2] Zum Beispiel so:

```
Dog dog2 = dog1;
```

Verändern Sie nun die Eigenschaften des Hundes über eine der beiden Variablen und untersuchen Sie, ob sich die Veränderung auf die andere Variable auswirkt!

Prüfen Sie dazu, wie sich Änderungen über eine Variable, also bswp. `dog1` *oder* `dog2`, auf "beide Hunde" auswirken. Ändern Sie z. B. den Namen des Hundes über eine der beiden Variablen und rufen Sie die Methode zum Bellen auf beiden Variablen auf. Alternativ können Sie nach einer Änderung über eine Variable die Eigenschaften "beider Hunde" ausgeben und vergleichen.

[2] Dadurch wird lediglich der "Wert" der Referenz, also die Adresse des Objekts im Speicher, kopiert. Das Objekt selbst wird hingegen *nicht* kopiert.

Hinweis: Die Referenzvariablen `dog1` und `dog2` funktionieren wie die im Buch beschriebene "Fernbedienung". Mit dem Punktoperator können Sie auf die Methode und die Instanzvariablen Ihres Hundes zugreifen.

2.2.2 Primitive Typen

Wiederholen Sie diese Schritte mit zwei **`int`**-Variablen!

Erstellen Sie also eine erste **`int`**-Variable mit einem Wert Ihrer Wahl und weisen Sie diesen Wert einer zweiten **`int`**-Variable zu:

```
int num2 = num1;
```

Ändern Sie nun den Wert der ersten Variable, indem Sie ihr eine neue Zahl zuweisen, und überprüfen Sie, ob sich der Wert der zweiten Variable ändert.

2.2.3 Diskussion

Diskutieren Sie im Coding-Team, wie sich Änderungen an einer Referenzvariablen auf alle Verweise auf dasselbe Objekt auswirken können, während primitive Typen davon unberührt bleiben.

3 Referenzen und `null`

Lernziele

- ☐ Sie verwenden den **`null`**-Wert, um eine nicht zugewiesene Referenzvariable darzustellen.
- ☐ Sie zeigen den Unterschied zwischen einer nicht zugewiesenen Referenzvariablen und einer Referenzvariablen, die auf ein Objekt verweist.

In dieser Aufgabe nutzen Sie den **`null`**-Wert, den Java verwendet, um eine nicht zugewiesene Referenz darzustellen. Dieses Verständnis ist wichtig, um den Umgang mit Referenztypen in Java sicher zu beherrschen und unerwartetes Verhalten bei nicht initialisierten Variablen zu vermeiden.[3]

Hinweis: In Java wird **`null`** als spezieller "Wert" verwendet, um anzuzeigen, dass eine Referenzvariable auf kein Objekt verweist. Im Gegensatz zu anderen Werten repräsentiert **`null`** nicht tatsächlich einen Datenwert, sondern zeigt das Fehlen einer Zuweisung an. Der Wert **`null`** bedeutet also, dass die Referenz "leer" ist und auf kein Objekt im Speicher zeigt. Dies ist besonders wichtig, da der Zugriff auf eine **`null`**-Referenz zu Laufzeitfehlern[4] führen kann, wenn versucht wird, auf Methoden oder Attribute eines nicht existierenden Objekts zuzugreifen.

3.1 `null` zur Darstellung nicht zugewiesener Referenzen

Deklarieren Sie eine Variable vom Typ `Dog` (siehe Aufgabe 2) ohne Zuweisung eines neuen `Dog`-Objekts und geben Sie die Referenz aus!

Zum Beispiel so: `System.out.println("Dog: " + nullDog);.`

Diskutieren Sie im Coding-Team, was ausgegeben wird und warum.

Tipp: Weisen Sie der Variablen bei der Deklaration explizit den Wert **`null`** zu. Damit signali-

[3] Die berühmt-berüchtigte `NullPointerException` lässt ganz herzlich grüßen. Wir werden uns ihr später noch ausführlich widmen.

[4] Da ist sie wieder, diese `NullPointerException`.

sieren Sie, dass sie auf kein Objekt verweist.[5]

3.2 Unterschied zu einem initialisierten Referenztyp

Deklarieren Sie eine Variable vom Typ `Dog`, weisen Sie ein neues `Dog`-Objekt zu und geben Sie die Referenz aus!

Zum Beispiel so: `System.out.println("Dog: " + bullDog);`.

Diskutieren Sie im Coding-Team, was der Unterschied ist und warum.

[5] Was passiert, wenn Sie den Wert `null` nicht explizit zuweisen? Warum? Probieren Sie es aus und diskutieren Sie anschließend im Coding-Team, welche Ausgabe erfolgt und warum.

4 Arrays als Referenztypen

Lernziele

- ☐ Sie zeigen, dass Arrays Referenztypen sind und als Objekte im Speicher verwaltet werden.
- ☐ Sie zeigen, dass die Zuweisung einer Array-Variablen an eine andere Variable die Referenz kopiert, anstatt eine Kopie des Arrays zu erstellen.
- ☐ Sie ändern die Elemente eines Arrays und beobachten, wie die Änderungen in allen Referenzen sichtbar werden, die auf dasselbe Array zeigen.

Strecklernziele

- ☐ Sie verwenden Arrays, um mehrere verschiedene Instanzen desselben Typs zu speichern und darauf zuzugreifen.

In dieser Aufgabe zeigen Sie, dass Arrays in Java Referenztypen sind und als Objekte im Speicher verwaltet werden. Das Verständnis dieser Eigenschaften ist wichtig, um Arrays korrekt zu handhaben und unerwartetes Verhalten bei der Zuweisung und Änderung von Array-Inhalten zu vermeiden. Sie untersuchen, wie das Kopieren von Array-Referenzen funktioniert und wie Änderungen an Array-Elementen über mehrere Referenzen hinweg sichtbar bleiben.

Schließlich haben Sie die Möglichkeit, Arrays zu verwenden, um eine Sammlung von Objekten zu speichern und auf die darin enthaltenen Objekte zuzugreifen.

4.1 Arrays als Referenztyp

Arrays können sowohl Elemente von primitiven Datentypen als auch von Referenztypen speichern. Arrays sind selbst Referenztypen.[6]

4.1.1 Arrays mit Elementartypen

Deklarieren und initialisieren Sie ein Array von Ganzzahlen (`int`)!

[6] Arrays können also auch andere Arrays enthalten. Dies führt zu mehrdimensionalen Arrays.

Erstellen Sie dazu ein neues Zahlenarray wie zum Beispiel `int[] numbers` mit 5 Elementen und initialisieren Sie es mit Werten wie 1, 2, 3, 4 und 5.

4.1.2 Arrays mit Referenzen

Deklarieren Sie ein Array von `Dog`-Objekten (siehe Aufgabe 2)!

Zum Beispiel `Dog[] dogs` mit 3 Hunden mit jeweils unterschiedlichen Eigenschaften.

4.1.3 Diskussion

Diskutieren Sie im Coding-Team anhand Ihrer konkreten Beispiele, warum Arrays in Java Objekte sind und daher als Referenztypen behandelt werden und wie sie im Speicher verwaltet werden.

4.2 Array-Referenzen vs. neue Array-Instanzen

Erstellen Sie eine zweite Referenz auf das Zahlenarray von oben, ändern Sie ein Element über die zweite Referenz und geben Sie das Element über beide Referenzen aus!

Erstellen Sie dazu eine neue Referenzvariable, z. B. `int[] numbers2`, und weisen Sie ihr die Referenz des ersten Zahlenarrays zu. Ändern Sie dann ein Element des Arrays über die neue Referenzvariable, z. B. `numbers2[0] = 10`, und geben Sie das geänderte Element über *beide* Referenzvariablen aus, z. B. `System.out.println(numbers[0])` für die erste Referenz.

Diskutieren Sie im Team, warum die Änderung in beiden Referenzen bzw. in “beiden” Arrays erscheint.

4.3 Arrays als Schrank voller Hunde

Erstellen Sie ein `Dog`-Array mit verschiedenen Hunden und lassen Sie alle Hunde bellen!

Erstellen Sie dazu ein neues Array von `Dog`-Objekten, z. B. `Dog[] dogs`, und initialisieren Sie es mit Hunden, die alle Eigenschaften Ihrer Wahl haben. Die Hunde sollten zumindest unterschiedliche Namen haben. Lassen Sie dann alle Hunde im Array bellen, indem Sie eine Schleife über das Array laufen lassen und für jeden Hund die Methode `bark()` aufrufen.

Tipp: Im Buch finden Sie auf Seite 62 ein Codebeispiel, das zeigt, wie man ein Array durchläuft[7] und auf die Elemente zugreift.

[7] Man sagt auch *iteriert*: Man iteriert über ein Array. Dabei kann man auf die Elemente zugreifen.