Programmierung 1

im Wintersemester 2025/26

Praktikumsaufgabe 4

Methoden benutzen Instanzvariablen (Kap. 4)

Review: 25. November 2025 (DFIW) und 26. November 2025 (PI)

In dieser Praktikumsaufgabe vertiefen Sie Ihr Verständnis für das Zusammenspiel von *Instanzvariablen* und *Methoden*. Zur Erinnerung: Instanzvariablen repräsentieren den Zustand und Methoden das Verhalten eines Objekts. Der Schwerpunkt liegt nun auf der Beziehung zwischen Zustand und Verhalten eines Objekts.

In Aufgabe 1 deklarieren Sie Instanzvariablen, die den Zustand eines Objekts repräsentieren, und verwenden diese in einer Methode, um den Zustand des Objekts zu ändern.

In Aufgabe 2 verwenden Sie Parameter, um externe Werte an eine Methode zu übergeben, sowie Rückgabewerte von Methoden, um das Verhalten eines Objekts zu steuern.

In Aufgabe 3 machen Sie Instanzvariablen durch den Einsatz von Zugriffsmodifikatoren (`private`) nur innerhalb der Klasse zugänglich, um eine Datenkapselung zu erreichen.

In Aufgabe 4 zeigen Sie, dass == bei primitiven Typen die Werte vergleicht, während bei Objekten die Referenzen verglichen werden. Schließlich verwenden Sie die Methode `equals()` für den inhaltlichen Vergleich von Objekten.

Heads up!

Für diese und die folgenden Praktikumsaufgaben gilt:

Konventionen und Praktiken: Halten Sie sich an die Konventionen der professionellen Softwareentwicklung und folgen Sie bewährten Verfahren. Wählen Sie aussagekräftige Namen. Strukturieren Sie Ihren Code durch Einrückungen, damit er übersichtlich und leicht nachvollziehbar ist. Verwenden Sie Pseudo-UML, um Ihre Klassen zu entwerfen.

Trennung von Produktions- und Testcode: Schreiben Sie den Code, der den "richtigen" *Produktions-* oder *Anwendungscode* ausführt und testet, in einer eigenen Testklasse, deren Name auf `TestDrive` endet. Der dortige Code wird *Testcode* genannt. Auf diese Weise trennen Sie Ihren Testcode von Ihrem Produktions- bzw. Anwendungscode.

Kontinuierliches Lernen: Nutzen Sie diese Aufgaben, um das Ergebnis Ihrer Reflexion des Feedbacks aus den vorherigen Aufgaben umzusetzen. So lernen Sie kontinuierlich dazu und können Ihre Fähigkeiten als professionelle Softwareentwickler:in ausbauen.

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

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

1 Instanzvariablen in Methoden verwenden

Lernziele

- ☐ Sie deklarieren Instanzvariablen, die den Zustand eines Objekts repräsentieren.
- ☐ Sie implementieren eine Methode, die eine Instanzvariable abhängig von einem Parameter ändert.
- ☐ Sie beobachten die Auswirkungen von Methodenaufrufen auf den Zustand eines Objekts.

In dieser Aufgabe verwenden Sie eine Instanzvariable innerhalb einer Methode. Hierzu erstellen Sie zunächst eine Klasse, die den Zustand einer Pflanze repräsentiert. Anschließend erstellen Sie eine Methode, mit der sich der Zustand der Pflanze ändern lässt.

1.1 Deklaration und Initialisierung von Instanzvariablen

Erstellen Sie eine Klasse namens `Plant`, die mithilfe der Instanzvariablen `height` (Typ **`double`**), `type` (Typ `String`) und `waterLevel` (Typ **`int`**) den Zustand einer Pflanze beschreibt. Diese Instanzvariablen sollen den aktuellen Zustand einer Pflanze repräsentieren.

1.1.1 Deklaration

Erstellen Sie die Klasse `Plant` und deklarieren Sie darin die oben genannten Instanzvariablen.

1.1.2 Initialisierung

Erstellen Sie eine `main`-Methode, die ein Objekt Ihrer Klasse `Plant` erzeugt und die drei Instanzvariablen `height`, `type` und `waterLevel` initialisiert.

Weisen Sie den Instanzvariablen direkt Werte zu, indem Sie den Punktoperator verwenden. Beispiel: `plant.height = 10.0;`.

Zur Erinnerung: Der passende Name für die Klasse, die diese Methode zum Ausführen und Testen Ihrer Klasse `Plant` enthält, lautet `PlantTestDrive`.

1.2 Methode zum Verändern einer Instanzvariablen

Erweitern Sie Ihre Klasse `Plant` um eine Methode, mit der sich die `height` der Pflanze abhängig von den Sonnenlichtstunden verändern lässt.

Implementieren Sie dazu die Methode `grow(int sunlightHours)`, die den Wert der Instanzvariablen `height` direkt um `0.5 * sunlightHours` erhöht. Die Methode soll also keinen Wert zurückgeben, sondern die Instanzvariable selbst verändern.

1.3 Auswirkung einer Methode auf den Zustand eines Objekts

Testen Sie die Methode `grow()` und beobachten Sie, wie sich die `height` der Pflanze dadurch verändert.

Erweitern Sie dazu Ihre `main`-Methode in `PlantTestDrive` um Aufrufe der Methode `grow()` und verwenden Sie verschiedene Werte für `sunlightHours`, um die Änderungen an `height` nach jedem Aufruf zu beobachten.

Zur Erinnerung: Verwenden Sie den Punktoperator, um lesend auf die Instanzvariablen zuzugreifen, und die Methode `System.out.println()`, um die Werte auszugeben.

1.4 Reflexion im Coding-Team

Diskutieren Sie im Coding-Team, um Ihr Verständnis zu vertiefen:

1. Warum ist es sinnvoll, den Zustand einer Pflanze durch Instanzvariablen wie `height` und `waterLevel` zu repräsentieren?

2. Warum könnte es problematisch sein, die Methode `grow()` ohne Instanzvariablen zu implementieren?

3. In welchen Szenarien könnte die Methode `grow()` in einer realen Anwendung nützlich sein?

4. Was passiert, wenn `sunlightHours` als negative Zahl an `grow()` übergeben wird? Wie könnte man dieses Problem lösen?

2 Methoden mit Parametern und Rückgabewerten

Lernziele

- ☐ Sie verwenden Parameter, um externe Werte an eine Methode zu übergeben und somit ihr Verhalten zu steuern.
- ☐ Sie implementieren eine Methode, die einen Wert zurückgibt, und beobachten den Unterschied zu einer Methode ohne Rückgabewert.
- ☐ Sie nutzen den Rückgabewert einer Methode, um den Wert auszugeben und weitere Berechnungen damit durchzuführen.

In dieser Aufgabe verwenden Sie die Parameter und Rückgabewerte von Methoden. Dazu erweitern Sie Ihre Klasse `Plant` um Methoden, die externe Werte entgegennehmen und zurückgeben. Anschließend nutzen Sie den Rückgabewert, um das Verhalten der Pflanze zu steuern.

2.1 Verwendung von Parametern in Methoden

Erweitern Sie Ihre Klasse `Plant` um die Methode `water()`, mit der sich der `waterLevel` der Pflanze anhand eines externen Parameters dynamisch erhöhen lässt. So können Sie steuern, wie viel Wasser die Pflanze erhält.

2.1.1 Deklaration der Methode

Implementieren Sie dazu die Methode `water(int amount)`, die den `waterLevel` der Pflanze um den Wert des Parameters `amount` erhöht.

2.1.2 Testen der Methode

Testen Sie die Methode `water()` in der `main`-Methode von `PlantTestDrive`, indem Sie der Pflanze verschiedene Wassermengen zuführen und anschließend jeweils den aktuellen `waterLevel` ausgeben.

2.2 Methode mit Rückgabewert

Erweitern Sie Ihre Klasse `Plant` um die Methode `needsWater()`, die überprüft, ob die Pflanze Wasser benötigt, und das entsprechende Ergebnis zurückgibt.

2.2.1 Deklaration der Methode

Implementieren Sie dazu die Methode `needsWater()`, die **`true`** zurückgibt, wenn der durch `waterLevel` repräsentierte Wasserstand einen bestimmten Wert (z. B. `5`) unterschreitet. Andernfalls gibt sie **`false`** zurück.

2.2.2 Testen der Methode

Rufen Sie die Methoden `water()` und `needsWater()` in `PlantTestDrive` auf und geben Sie das Ergebnis auf der Konsole aus, um zu beobachten, ob die Pflanze je nach `waterLevel` Wasser benötigt oder nicht.

2.3 Rückgabewert einer Methode

Erweitern Sie `PlantTestDrive` so, dass der Rückgabewert der Methode `needsWater()` in einer Bedingung verwendet wird und die Pflanze bei Bedarf gewässert wird:

- Verwenden Sie dazu eine **`if`**-Anweisung, die den Rückgabewert von `needsWater()` prüft.
- Wenn `needsWater()` den Wert **`true`** zurückgibt, rufen Sie die Methode `water()` auf, um die Pflanze mit einer festgelegten Menge Wasser (z. B. drei Einheiten) zu versorgen.
- Geben Sie anschließend den `waterLevel` der Pflanze aus, um die Änderungen zu beobachten.

2.4 Reflexion im Coding-Team

Diskutieren Sie im Coding-Team, um Ihr Verständnis zu vertiefen:

1. Warum sind Parameter wie `amount` in der Methode `water()` notwendig, um das Verhalten flexibel zu gestalten?

2. Was ist der Unterschied zwischen einer Methode, die einen Wert zurückgibt, und einer Methode ohne Rückgabewert?

3. Welche Rückgabewerte erwarten wir für die Methode `needsWater()` in verschiedenen Szenarien? Was sagt uns das Ergebnis über den Zustand der Pflanze?

4. Wie könnte eine Methode wie `needsWater()` in einem komplexeren Programm genutzt werden? Denken Sie beispielsweise an automatische Benachrichtigungen in einer Pflanzenpflege-App.

3 Kapselung durch Getter und Setter

Lernziele

- ☐ Sie machen Instanzvariablen durch den Einsatz von Zugriffsmodifikatoren (**`private`**) nur innerhalb der Klasse zugänglich, um eine Datenkapselung zu erreichen.
- ☐ Sie erstellen Getter-Methoden, um von außerhalb der Klasse kontrolliert auf private Instanzvariablen zuzugreifen.
- ☐ Sie schreiben Setter-Methoden, die Parameter verwenden, um den Zustand (Wert) von Instanzvariablen sicher zu ändern und zu kontrollieren.
- ☐ Sie implementieren Validierungen innerhalb von Setter-Methoden, um sicherzustellen, dass nur gültige Werte zugewiesen werden.

In dieser Aufgabe wenden Sie das Konzept der Kapselung mithilfe von Getter- und Setter-Methoden an. Dabei validieren Sie die Eingabe von Werten, um sicherzustellen, dass sich die Objekte stets in einem sinnvollen bzw. gültigen Zustand befinden.

Heads up!

Wenn Sie die Instanzvariablen der Klasse `Plant` auf **`private`** setzen, ist ein direkter Zugriff auf diese Variablen von außerhalb der Klasse – also auch in `PlantTestDrive` – nicht mehr möglich. Ab diesem Zeitpunkt müssen Sie die Getter- und Setter-Methoden verwenden, um die Werte der Instanzvariablen von außen zu lesen oder zu ändern.

Ersetzen Sie im Laufe dieser Aufgabe die direkten Zugriffe mittels des Punktoperators durch Aufrufe der Getter- und Setter-Methoden. Anschließend sollte Ihr Testcode erneut kompilierbar sein und die Tests sollten wieder funktionieren. Stellen Sie sicher, dass dies der Fall ist!

3.1 Kapseln von Instanzvariablen durch Zugriffsmodifikatoren

Um die Kapselung zu gewährleisten, ändern Sie die Sichtbarkeit aller Instanzvariablen der Klasse `Plant` auf **`private`**. Dadurch wird ein direkter Zugriff von außerhalb der Klasse verhindert und der Zustand der Pflanze geschützt.

3.2 Getter-Methoden, um den Zustand eines Objekts zu lesen

Fügen Sie für jede private Instanzvariable eine Getter-Methode hinzu, die den aktuellen Wert der jeweiligen Variablen zurückgibt.

Ein Beispiel: Die Methode `getHeight()` gibt die Höhe der Pflanze zurück.

Testen Sie anschließend die Getter-Methoden in `PlantTestDrive`, indem Sie die aktuellen Werte der Instanzvariablen über die Getter ausgeben.

3.3 Setter-Methoden, um den Zustand eines Objekts zu verändern

Fügen Sie als Nächstes für jede Instanzvariable eine Setter-Methode hinzu, die über einen Parameter neue Werte zuweist.

Ein Beispiel: Die Methode `setHeight(double height)` setzt die Höhe der Pflanze auf den übergebenen Wert.

Testen Sie anschließend die Setter-Methoden in `PlantTestDrive`, indem Sie die Werte der Instanzvariablen ändern und die neuen Werte über die Getter-Methoden ausgeben.

3.4 Validierungen in Setter-Methoden, um fehlerhafte Zustandsänderungen zu verhindern

Ergänzen Sie abschließend die Setter-Methoden für `height` und `waterLevel` um eine Validierung:

- Da eine negative Größe keinen Sinn ergibt, darf `height` nicht negativ sein.
- Der Wert von `waterLevel` darf einen bestimmten Maximalwert (z. B. `10`) nicht überschreiten, da die Pflanze sonst ertrinkt.

- Wenn ein ungültiger Wert übergeben wird, soll der Wert unverändert bleiben und eine Fehlermeldung soll ausgegeben werden.[1]

Testen Sie die Validierungen in `PlantTestDrive`, indem Sie ungültige Werte für `height` und `waterLevel` über die Setter zuweisen und die Ausgaben überprüfen.

Tipp: Erinnern Sie sich an die Reflexionsfrage zu den negativen `sunlightHours` aus Aufgabe 1? Nutzen Sie die Gelegenheit, um mithilfe einer passenden Validierung die Methode `grow()` robuster zu gestalten.

3.5 Reflexion im Coding-Team

Diskutieren Sie im Coding-Team, um Ihr Verständnis zu vertiefen:

1. Warum ist es sinnvoll, die Instanzvariablen `height`, `type`, und `waterLevel` in der Klasse `Plant` als **`private`** zu deklarieren?

2. In welchen Fällen könnten Getter-Methoden besonders nützlich sein, um Informationen über den Zustand eines Objekts zu erhalten?

3. Welche Vorteile bieten Setter-Methoden, die Validierungen enthalten? Wie würden Sie den Nutzen in einer realen Anwendung beschreiben?

4. Stellen Sie sich vor, `height` könnte versehentlich auf einen negativen Wert gesetzt werden. Welche negativen Folgen könnte das haben und wie schützt die Kapselung vor solchen Fehlern?

[1] Dies ist eine Übergangslösung. Im späteren Verlauf werden wir uns mit Exceptions beschäftigen.

4 Vergleich von primitiven Werten und Referenzen

Lernziele

☐ Sie unterscheiden zwischen dem Vergleich primitiver Typen und Objektreferenzen.
☐ Sie wenden die Methode `equals()` zum Vergleich von Objekten an.

In dieser Aufgabe zeigen Sie, dass `==` bei primitiven Typen die Werte vergleicht, während bei Objekten die Referenzen verglichen werden. Zum Vergleich des Inhalts zweier Objekte, beispielsweise zweier Pflanzen, verwenden Sie schließlich die Methode `equals()`.

4.1 Vergleich von primitiven Typen

Erstellen Sie in `PlantTestDrive` zwei Variablen vom Typ **`double`** namens `height1` und `height2` und weisen Sie ihnen den gleichen Wert zu.

Überprüfen Sie mithilfe von `==`, ob die beiden Variablen gleich sind, und geben Sie das Ergebnis auf der Konsole aus.

Zum Beispiel so: `System.out.println(height1 == height2);`

Noch anschaulicher ist es, wenn Sie das Ergebnis in einer Variablen speichern und diese dann in einem erklärenden Text ausgeben.

4.2 Vergleich von Objektreferenzen

Erstellen Sie zwei `Plant`-Objekte mit identischen Eigenschaften, d. h. mit den gleichen Werten für `height`, `type` und `waterLevel`. Sie könnten beispielsweise die Objekte `plant1` und `plant2` erstellen, die beide eine Höhe von `1.0` haben, vom Typ `"cactus"` sind und `3` Einheiten Wasser enthalten.

Vergleichen Sie die beiden Objekte mit `==` und geben Sie das Ergebnis auf der Konsole aus.

Diskutieren Sie anschließend im Coding-Team, warum dies im Vergleich zu primitiven Typen nicht das gewünschte Ergebnis liefert.

4.3 Vergleich von Objekten mit `equals()`

Setzen Sie zunächst den `type`-Wert der beiden `Plant`-Objekte `plant1` und `plant2` auf unterschiedliche Werte (z. B. `"cactus"` und `"rose"`), dann auf den gleichen Wert (z. B. `"rose"`).

Vergleichen Sie in beiden Fällen den `type`-Wert von `plant1` und `plant2` mit der Methode `equals()`.

Zum Beispiel so: `plant1.getType().equals(plant2.getType());`

Geben Sie das Ergebnis auf der Konsole aus und diskutieren Sie im Coding-Team, warum diese Methode hier nützlich ist.

4.4 Reflexion im Coding-Team

Diskutieren Sie im Coding-Team, um Ihr Verständnis zu vertiefen:

1. Warum vergleicht == bei primitiven Typen wie **`double`** die Werte, bei Objekten jedoch nur die Referenzen?

2. In welchen Fällen kann es zu unerwarteten Ergebnissen kommen, wenn == statt der Methode`equals()` für den Vergleich von Objekten verwendet wird?

3. Warum ist die Methode `equals()` wichtig, wenn wir die Inhalte von Objekten vergleichen wollen? Wie funktioniert das bei `String`-Objekten?

4. Wie kann in einem Programm sichergestellt werden, dass zwei Objekte denselben Inhalt haben, auch wenn sie unterschiedliche Referenzen besitzen?