Programmierung 1

im Wintersemester 2024/25

Praktikumsaufgabe 4

Methoden benutzen Instanzvariablen

Review: 27. November 2024

In dieser Praktikumsaufgabe vertiefen Sie Ihr Verständnis des Zusammenspiels von *Instanzvariablen* und *Methoden*. Zur Erinnerung: Diese repräsentieren den Zustand und das Verhalten eines Objekts. Der Schwerpunkt liegt nun auf der Beziehung zwischen dem Zustand und dem 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, und 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 den 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" *Produktionscode* ausführt und testet, in einer separaten Klasse "`TestDrive`". Dieser Code wird *Testcode* genannt.

Kontinuierliches Lernen: Nutzen Sie diese Aufgaben, um das Ergebnis Ihrer Reflexion des Feedbacks aus den vorherigen Aufgaben umzusetzen. So lernen Sie kontinuierlich 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 in der Gruppe 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 in Abhängigkeit von einem Parameter ändert.
- ☐ Sie beobachten die Auswirkungen von Methodenaufrufen auf den Zustand eines Objekts.

In dieser Aufgabe verwenden Sie eine Instanzvariable in einer Methode. Dazu erstellen Sie eine Klasse, die den Zustand einer Pflanze repräsentiert, und eine Methode, die den Zustand der Pflanze ändert.

1.1 Deklaration und Initialisierung von Instanzvariablen

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

1.1.1 Deklaration

Erstellen Sie eine Klasse `Plant`, die diese Instanzvariablen deklariert.

1.1.2 Initialisierung

Erstellen Sie eine `main`-Methode, die ein Objekt der Klasse `Plant` erstellt und die 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, ist `PlantTestDrive`.

1.2 Methode zum Verändern einer Instanzvariablen

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

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 und 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 auch, um auf die Instanzvariablen lesend zuzugreifen, und die Methode `System.out.println()` zum Ausgeben der Werte.

1.4 Reflexion im Coding-Team

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

1. Wieso 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 wir `sunlightHours` als negative Zahl an `grow()` übergeben? Wie könnte man das Problem lösen?

2 Methoden mit Parametern und Rückgabewerten

Lernziele

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

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

2.1 Verwendung von Parametern in Methoden

Erweitern Sie die Klasse `Plant` um eine Methode `water()`, die den `waterLevel` der Pflanze dynamisch anhand eines externen Parameters erhöht. Dadurch können Sie steuern, wie viel Wasser die Pflanze erhält.

2.1.1 Deklaration der Methode

Implementieren Sie 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 in `PlantTestDrive`, indem Sie der Pflanze verschiedene Wassermengen zuführen und nach jedem Aufruf den aktuellen `waterLevel` ausgeben.

2.2 Methode mit Rückgabewert

Erweitern Sie die Klasse `Plant` um eine Methode `needsWater()`, die überprüft, ob die Pflanze Wasser benötigt, und ein entsprechendes Ergebnis zurückgibt.

2.2.1 Deklaration der Methode

Implementieren Sie die Methode `needsWater()`, die **`true`** zurückgibt, wenn der durch `waterLevel` repräsentierte Wasserstand unter einem bestimmten Wert liegt (z. B. `5`), und sonst **`false`**.

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`, um den Rückgabewert der Methode `needsWater()` in einer Bedingung zu verwenden und darauf basierend die Pflanze zu wässern:

- Verwenden Sie eine **`if`**-Anweisung, die den Rückgabewert von `needsWater()` prüft.
- Falls `needsWater()` **`true`** zurückgibt, rufen Sie die Methode `water()` auf, um die Pflanze mit einer festgelegten Menge Wasser zu versorgen (z. B. `3` Einheiten).
- Geben Sie den `waterLevel` der Pflanze nach der Wässerung 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, z. B. für 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 durch die Verwendung von Getter- und Setter-Methoden an. Sie validieren die Eingabe von Werten, um sicherzustellen, dass Objekte in einem sinnvollen bzw. gültigen Zustand bleiben.

Heads up!

Sobald Sie die Instanzvariablen in der Klasse `Plant` auf **`private`** setzen, können Sie von außerhalb der Klasse `Plant` – also auch in `PlantTestDrive` – nicht mehr direkt auf diese Variablen zugreifen. Ab diesem Zeitpunkt müssen Sie die Getter- und Setter-Methoden in `PlantTestDrive` verwenden, um die Werte der Instanzvariablen zu lesen oder zu ändern.

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

3.1 Kapseln von Instanzvariablen durch Zugriffsmodifikatoren

Ändern Sie die Sichtbarkeit aller Instanzvariablen (`height`, `type` und `waterLevel`) in der Klasse `Plant` auf **`private`**, um die Kapselung zu gewährleisten. Dies verhindert direkten Zugriff von außerhalb der Klasse und schützt den Zustand der Pflanze.

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

Fügen Sie für jede private Instanzvariable (`height`, `type`, `waterLevel`) eine Getter-Methode hinzu, die den aktuellen Wert der jeweiligen Variablen zurückgibt.

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

Testen Sie 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 für die Instanzvariablen `height`, `type` und `waterLevel` Setter-Methoden hinzu, die über einen Parameter neue Werte zuweisen.

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

Testen Sie die Setter-Methoden in `PlantTestDrive`, indem Sie die Werte für `height`, `type` und `waterLevel` ändern und anschließend die neuen Werte über die Getter-Methoden ausgeben.

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

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

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

- Falls ein ungültiger Wert übergeben wird, soll der Wert unverändert bleiben und eine Fehlermeldung 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.

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 von primitiven Typen und Objektreferenzen.
☐ Sie verwenden `equals()` für den Vergleich von Objekten.

In dieser Aufgabe zeigen Sie, dass `==` bei primitiven Typen die Werte vergleicht, während bei Objekten die Referenzen verglichen werden. Schließlich verwenden Sie die Methode `equals()`, um den Inhalt zweier Objekte zu vergleichen, wie z. B. den Typ zweier Pflanzen.

4.1 Vergleich von primitiven Typen

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

Überprüfen Sie mit `==`, 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 innerhalb eines erklärenden Textes ausgeben.

4.2 Vergleich von Objektreferenzen

Erstellen Sie zwei `Plant`-Objekte mit identischen Eigenschaften, also mit den gleichen Werten für `height`, `type` und `waterLevel`. Z. B. `plant1` und `plant2`, die beide `1.0` hoch sind, den Typ `"cactus"` haben und `3` Einheiten Wasser enthalten.

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

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

4.3 Vergleich von Objekten mit `equals()`

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

Vergleichen Sie für beide Fälle 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 `equals()` 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 könnte es zu unerwarteten Ergebnissen kommen, wenn == anstelle von `equals()` für den Vergleich von Objekten verwendet wird?

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

4. Wie könnten Sie in einem Programm sicherstellen, dass zwei Objekte denselben Inhalt haben, auch wenn sie unterschiedliche Referenzen besitzen?