Programmierung 1

im Wintersemester 2024/25

Praktikumsaufgabe 6

Die Java-API kennenlernen

Review: 11. Dezember 2024

Diese Praktikumsaufgabe führt Sie in die Verwendung der mit Java gelieferten Klassen und Methoden ein. Anschließend organisieren Sie Ihren Code mit Hilfe von Paketen und wechseln zur Konsole, um Ihre Programme zu kompilieren und auszuführen.

In Aufgabe 1 implementieren Sie einen Wrapper um ein Array, der verschiedene Funktionalitäten objektorientiert bereitstellt.

In Aufgabe 2 wechseln Sie von einem Array-basierten Ansatz zu einem `ArrayList`-basierten Ansatz, um eine dynamische Datenstruktur zu verwenden.

In Aufgabe 3 erweitern Sie den Wrapper, um mehr über die API der Klasse `ArrayList` zu erfahren.

In Aufgabe 4 verwenden Sie Import-Anweisungen, um Klassen aus anderen Paketen zu verwenden. Abschließend organisieren Sie Ihren Code in Paketen und führen ihn auf der Konsole aus.

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. Zum Beispiel: 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.

Testgetriebene Entwicklung: Gehen Sie iterativ und inkrementell vor! Erstellen Sie zunächst eine Liste von Testszenarien für die geplanten Funktionalitäten. Implementieren Sie ein Testszenario aus der Liste als fehlschlagenden Test (*Red*). Entwickeln Sie dann nur so viel Code, wie nötig ist, um den Test zu bestehen (*Green*). Optimieren Sie schließlich Ihren Code (*Refactor*). Führen Sie dabei nach Änderungen alle Tests erneut aus. Wiederholen Sie diesen Zyklus für alle Testszenarien. Diese Vorgehensweise ermöglicht eine kontrollierte und schrittweise Entwicklung.

Trennung von Produktions- und Testcode: Schreiben Sie den *Testcode*[1], der den *Produktionscode* mit der eigentlichen Programmlogik ausführt und testet, in einer separaten Klasse "`TestDrive`".

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] Wenn Sie möchten, können Sie JUnit 5 anstelle einer `TestDrive`-Klasse mit einer `main`-Methode verwenden, um Ihre Tests zu automatisieren. Das Video Java Unit Testing with JUnit - Tutorial - How to Create And Use Unit Tests bietet eine gute und leicht verständliche Einführung.

1 Objektorientierter `String[]`-Wrapper

Lernziele

- ☐ Sie implementieren einen Wrapper um ein Array, um objektorientiert auf die Funktionalität des umhüllten Arrays zuzugreifen.
- ☐ Sie implementieren Methoden, um auf die Elemente des umhüllten Arrays zuzugreifen, Elemente hinzuzufügen und zu entfernen.
- ☐ Sie entwickeln eine Strategie, um das umhüllte Array zu vergrößern, wenn es voll ist.

Stellen Sie sich vor, Sie möchten die Namen Ihrer Lieblingslieder speichern und verwalten. Dazu können Sie ein Array von Zeichenketten (`String`-Objekte) verwenden, deren Elemente die Namen der Lieder sind.

In dieser Aufgabe untersuchen Sie die Einschränkungen von statischen Arrays, indem Sie einen objektorientierten Wrapper um ein solches Array von `String`-Objekten erstellen.

In Java sind die meisten Typen von `Object` abgeleitet. Dazu gehört auch `String`. Warum gehören Arrays und primitive Typen wie **`int`**, **`char`** und **`boolean`** nicht dazu? Diese Typen werden in Java aus folgenden Gründen nicht von `Object` abgeleitet:[2]

Primitive Typen: Primitive Typen sind keine Objekte, sondern einfache Werte, die direkt im Speicher abgelegt werden. Sie sind nicht von `Object` abgeleitet, um eine effizientere Speichernutzung und schnellere Verarbeitung zu ermöglichen. Stattdessen gibt es für jeden primitiven Typ eine entsprechende Wrapper-Klasse (z. B. `Integer` für **`int`**), die von `Object` abgeleitet ist.[3]

Arrays: Obwohl Arrays in Java als Objekte behandelt werden, sind sie technisch gesehen keine regulären Objekte einer bestimmten Klasse. Arrays werden vom Java-Compiler speziell behandelt und zur Laufzeit als Objekte verwaltet, ohne dass eine explizite Array-Klassendefinition vorliegt. Dies ermöglicht eine effizientere Implementierung und bessere Performance bei der Arbeit mit Arrays.

Sie implementieren nun einen Wrapper um ein Array von `String`-Objekten, also Referenzen

[2] Es gibt Programmiersprachen, die von Grund auf objektorientiert sind. Beispiele hierfür sind Smalltalk, Ruby und Scala.

[3] Mehr dazu im Kapitel *Zahlen und Statisches*.

auf Zeichenketten, so dass Sie die Vorteile von Objekten nutzen können.

Ein Wrapper (engl. für "Verpackung" oder "Umhüllung") oder Adapter (lat. für "Anpassung") ist ein Stück Software, das ein anderes Stück Software umhüllt. Ein solcher Wrapper dient dazu, eine Schnittstelle in eine andere zu übersetzen.[4]

In unserem Fall handelt es sich um die Übersetzung einer vollständig objektorientierten Schnittstelle in die Funktionalität eines Arrays. Der Wrapper muss also die Funktionalität eines Arrays zur Verfügung stellen, aber wie ein Objekt behandelt werden können.

Heads up! Vergessen Sie nicht, testgetriebene Entwicklung (TDD) zu verwenden! Zusätzlich kann ein abstrakter Entwurf in Form eines Flussdiagramms sowie Vorcode hilfreich sein, insbesondere für die Resize-Logik.

1.1 Einfache Zugriffsmethoden

Implementieren Sie den Wrapper `StringArray` und die zugehörigen Tests in der Testklasse `StringArrayTestDrive`!

Der Wrapper soll intern ein `String`-Array (`String[]`) verwenden und extern die folgenden Methoden zur Verfügung stellen:

`String get(int index)` Gibt eine Zeichenkette zurück. Gibt **`null`** zurück, wenn der Index ungültig ist oder das Element nicht gesetzt wurde.

`void set(int index, String value)` Fügt eine Zeichenkette an der angegebenen Position ein. Wenn der Index ungültig ist, geschieht nichts.

`int size()` Ermittelt, wie viele Zeichenketten (Referenzen) und Nullwerte enthalten sind. Kurz: Die Länge des internen Arrays.

`void remove(int index)` Eine Zeichenkette (Referenz) entfernen. Wenn der Index ungültig ist, passiert nichts.

[4] In der objektorientierten Programmierung definiert eine Schnittstelle (engl. interface oder protocol), welche Methoden in den verschiedenen Klassen usw. vorhanden sind oder vorhanden sein müssen.

Heads up! Die obige Liste von Methoden ist ein guter Anhaltspunkt für eine anfängliche Liste von Testszenarien, die für die Durchführung von TDD erforderlich sind.

Hinweis: Eine bessere Lösung für ungültige Indizes und andere Fehlerfälle (sog. Ausnahmen oder Exceptions) werden wir im Kapitel *Exception-Handling* behandeln. Mit diesen kann eine spezifische Rückmeldung an die Nutzer:innen des Wrappers gegeben werden, wenn ein Fehler auftritt. Für den Moment ist es jedoch ausreichend, nichts zu tun oder `null` zurückzugeben.

1.2 Hinzufügen mit Resize-Logik

Implementieren Sie die Methode `int add(String value)`!

Die Methode `add` fügt eine Zeichenkette (Referenz) an das Ende des internen Arrays an. Sie gibt den Index zurück, an dem die Zeichenkette eingefügt wurde.

Heads up! Wenn das Array voll ist, soll es automatisch vergrößert werden. Die Größe des Arrays soll sich um 50% erhöhen, wenn es voll ist.[5] Erstellen Sie ein neues, größeres Array und kopieren Sie die Elemente. Verwenden Sie dazu *keine* Methoden wie `System.arraycopy()` oder `Arrays.copyOf()`, sondern *kopieren Sie die Elemente manuell*. Verwenden Sie dazu eine geeignete Schleife.

Hinweis: Entscheiden Sie, woran Sie erkennen können, dass das Array voll ist. Eine Möglichkeit ist, dass die Anzahl der enthaltenen Zeichenketten (Referenz) gleich der Länge des internen Arrays ist. In diesem Fall ist kein Platz mehr für eine weitere Zeichenkette.

Eine geeignete Resize-Logik zur Umgehung der statischen Größe von Arrays besteht im Prinzip aus zwei Schritten:

Erstellung eines neuen Arrays: Erstellen Sie ein neues Array, das 50% größer ist als das aktuelle Array. Berechnen Sie dazu zuerst die neue Größe und erstellen Sie dann das neue Array mit der neuen Größe.

Kopieren der Daten: Kopieren Sie die Elemente aus dem aktuellen Array in das neue Array mit der angepassten Größe. Verwenden Sie dazu eine geeignete Schleife, um über die

[5] Diese Strategie bietet einen guten Kompromiss zwischen Speicherverbrauch und Leistung.

Elemente zu iterieren und sie zu kopieren.

Somit würden Sie aus dem internen Array

```
["Periódico De Ayer", "Fuego En El 23", "Pedro Navaja", "Vivir
 ↪  Mi Vida"]
```

das *neue* interne Array

```
["Periódico De Ayer", "Fuego En El 23", "Pedro Navaja", "Vivir
 ↪  Mi Vida", "Sin Salsa No Hay Paraíso", null]
```

machen, wenn Sie `add("Sin Salsa No Hay Paraíso")` aufrufen.

1.3 Reflexion im Coding-Team

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

1.3.1 Grenzen von Arrays

Welche Einschränkungen gibt es bei der Verwendung von Arrays?

- Was ist der Hauptunterschied zwischen der normalen und der rein objektorientierten Nutzung der Funktionalität von Arrays durch Methoden?
- Wie haben Sie die Funktionalität von Arrays in Ihrem Wrapper implementiert, damit diese objektorientiert genutzt werden kann?

- Welche Funktionalität des von Ihnen “gewrappten” Arrays würden Sie gerne nativ von Java implementiert sehen?

1.3.2 Resize-Logik

Wie sind Sie bei der Umsetzung der Resize-Logik vorgegangen?

- Welche Strategie haben Sie gewählt? Warum?
- Wie hat Ihnen ein abstrakter Entwurf in Form eines Flussdiagramms bei der Entscheidung geholfen?
- Wie hat Ihnen der Vorcode bei der konkreten Implementierung geholfen?

1.3.3 Testgetriebene Entwicklung

Wie haben Sie TDD eingesetzt, um die Aufgabe zu lösen?

- Welche Testszenarien haben Sie anfänglich definiert? Wozu?
- Welche Testszenarien haben Sie hinzugefügt, um Fehlerfälle zu testen? Wozu?
- Welche Testszenarien haben Sie hinzugefügt, um die Resize-Logik zu testen? Wozu?

1.3.4 Behandlung von Fehlerfällen

Wie könnten Fehlerfälle nach außen signalisiert werden?

- Wie könnten Fehlerfälle durch den Wrapper behandelt werden, so dass solche Ausnahmesituationen von außen erkennbar sind?

- Welche Änderungen würden Sie an den Rückgabetypen bzw. -werten der Methoden vornehmen?

- Welche zusätzlichen Methoden könnten nach außen angeboten werden, damit die Nutzer:innen des Wrappers Fehlerfälle vermeiden bzw. besser darauf reagieren können?

2 `ArrayList<String>` statt `String[]`

Lernziele

- ☐ Sie verwenden die Klasse `ArrayList`, um dynamische Datenstrukturen zu implementieren.
- ☐ Sie migrieren von einem Array-basierten Ansatz zu einem `ArrayList`-basierten Ansatz.
- ☐ Sie diskutieren die Vor- und Nachteile von `ArrayList` im Vergleich zu Arrays.

In dieser Aufgabe wechseln Sie von einem Array-basierten und damit statischen Ansatz zu einem `ArrayList`-basierten Ansatz. Warum verwenden viele professionelle Entwickler:innen `ArrayList` statt Arrays? Dies hängt unter anderem mit der Flexibilität und den Funktionen zusammen, die `ArrayList` bietet.

Heads up! Vergessen Sie nicht, testgetriebene Entwicklung (TDD) zu verwenden! In dieser Aufgabe bedeutet dies, dass Sie zumindest die Tests vor dem Produktionscode schreiben.

2.1 Migration der Tests

Erstellen Sie die Testklasse `StringArrayListTestDrive` und migrieren Sie die Tests von `StringArrayTestDrive` aus Aufgabe 1 in diese neue Testklasse!

Hinweis: Unter der Annahme, dass Sie im bisherigen Testcode Variablen wie `wrapper` vom Typ `StringArray` verwendet haben, sollte es ausreichen, den Testcode zu kopieren, den Namen der Testklasse anzupassen und den Typ der Variablen in `StringArrayList` zu ändern. Schließlich soll `StringArrayList` die gleichen Methoden implementieren wie `StringArray`.

Erstellen Sie dann die Klasse `StringArrayList` und implementieren Sie die Methoden `get`, `set`, `size` und `remove` aus Aufgabe 1.1 und die Methode `add` Aufgabe 1.2 so, dass der Testcode kompiliert werden kann. Geben Sie als temporäre Maßnahme einfach Werte wie **`null`** oder `-1` zurück, wie Sie es u. a. aus der Vorlesung kennen.

Heads up! Damit haben Sie die Klasse `StringArrayListTestDrive`, bei deren Ausführung alle Tests fehlschlagen sollten. Gleichzeitig haben Sie das Sicherheitsnetz aus der

vorherigen Aufgabe, das Ihnen hilft, die bereits implementierte Funktionalität zu erhalten. Dies ist ein idealer Ausgangspunkt für den nächsten Teil dieser Aufgabe.

2.2 Austausch der Datenstruktur

Migrieren Sie den Produktionscode von `StringArray` nach `StringArrayList`!

Ändern Sie dazu die interne Datenstruktur von Array nach `ArrayList<String>`, indem Sie den Typ der Instanzvariablen, die die Zeichenketten speichert, entsprechend anpassen.[6]

Kopieren Sie dann den Code von `StringArray` Stück für Stück, z. B. Methode für Methode, nach `StringArrayList` und passen Sie ihn so an, dass `ArrayList` statt des Arrays verwendet wird. Rufen Sie die entsprechenden Methoden der API der von Java bereitgestellten `ArrayList`-Klasse auf, um die Funktionalität zu implementieren.

Heads up! Verwenden Sie die Tests in `StringArrayListTestDrive`, um sicherzustellen, dass Ihre Migration korrekt ist.

2.3 Reflexion im Coding-Team

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

Stellen Sie sich dazu vor, Sie entwickeln eine Anwendung, die Daten dynamisch verwalten muss. Wie würde sich die Wahl zwischen Arrays und `ArrayList` auf Ihre Implementierung auswirken?

2.3.1 `ArrayList` als Alternative zu Arrays

Warum ist die `ArrayList` eine sinnvolle Alternative zu Arrays?

[6] Denken Sie daran, die Klasse `ArrayList` zu importieren: `import java.util.ArrayList;`

- Welche Herausforderungen gab es bei der Implementierung der Resize-Logik in Aufgabe 1.2?
- Wie begegnet die `ArrayList` diesen Herausforderungen?
- In welchen Anwendungsfällen würden Sie weiterhin ein statisches Array bevorzugen?
- Welche anderen Datenstrukturen könnte Java als Alternative zu `ArrayList` anbieten?

2.3.2 `ArrayList` als dynamische Datenstruktur

Welche Methoden von `ArrayList` erleichtern das Arbeiten mit dynamischen Datenstrukturen?

- Welche Methoden der `ArrayList` haben Sie in Ihrer Umsetzung verwendet?
- Worin unterscheidet sich die Verwendung von `add()` und `set()` in der `ArrayList`?
- Welche Methoden könnten in realen Anwendungen nützlich sein, z. B. `indexOf()` oder `contains()`?
- Wie hilft die Methode `isEmpty()` bei der Arbeit mit dynamischen Daten?

2.3.3 Flexibilität von `ArrayList`

Wie hebt sich die `ArrayList` von Arrays ab?

- Welche Vorteile bietet die `ArrayList` in Bezug auf Flexibilität und Wartbarkeit?
- Wie unterscheidet sich der Zugriff auf Elemente in einer `ArrayList` von dem Zugriff auf Elemente in einem Array?
- In welchen Szenarien können die Einschränkungen eines Arrays dennoch von Vorteil sein?

2.3.4 `ArrayList` und Fehlerbehandlung

Wie behandelt die `ArrayList` typische Fehlerfälle?

- Was passiert, wenn Sie versuchen, auf einen ungültigen Index in einer `ArrayList` zuzugreifen?
- Wie unterscheidet sich das Verhalten der `ArrayList` im Fehlerfall von Ihrer Implementierung in Aufgabe 1.1?
- Warum ist es sinnvoll, Fehlerfälle wie einen ungültigen Index explizit zu behandeln?
- Wie können Sie sicherstellen, dass Nutzer:innen Ihrer `ArrayList`-basierten Implementierung solche Fehler vermeiden können?

3 Erweiterung des Wrappers

Lernziele

- ☐ Sie erweitern Ihren Wrapper `StringArrayList` um zusätzliche Funktionalität und wenden dabei testgetriebene Entwicklung an.
- ☐ Sie verwenden zusätzliche Methoden der Klasse `ArrayList`, um mehr über die API der Klasse zu erfahren.
- ☐ Sie diskutieren die Vorteile der von Java bereitgestellten Datenstrukturen.

Stellen Sie sich vor, Sie verwalten eine dynamische To-Do-Liste, die regelmäßig bereinigt und überprüft werden muss.

In dieser Aufgabe erweitern Sie Ihren Wrapper `StringArrayList` um die dafür notwendigen Methoden, indem Sie zusätzliche Methoden der Klasse `ArrayList` verwenden. Auf diese Weise lernen Sie die API der Klasse weiter kennen.

Heads up! Vergessen Sie nicht, testgetriebene Entwicklung (TDD) zu verwenden! Erweitern Sie also zuerst Ihre Testklasse `StringArrayListTestDrive` und dann Ihre Klasse `StringArrayList`.

3.1 Herausfinden, ob etwas in den Daten enthalten ist

Implementieren Sie die Methode `boolean contains(String value)`!

Die Methode soll `true` zurückgeben, wenn der Wert `value` in den Zeichenketten enthalten ist, andernfalls `false`.

Tipp: Verwenden Sie zum Beispiel die Methode `contains()` der Klasse `ArrayList`.

3.2 Die Daten leeren

Implementieren Sie die Methode `void clear()`!

Die Methode soll alle Zeichenketten löschen. Setzen Sie dazu alle Werte auf **`null`**.

Tipp: Verwenden Sie zum Beispiel die Methode `clear()` der Klasse `ArrayList`.

3.3 Daten als Array zurückgeben

Implementieren Sie die Methode `String[] toArray()`!

Die Methode soll ein Array zurückgeben, das alle Zeichenketten enthält. Die Reihenfolge der Zeichenketten muss erhalten bleiben.

Tipp: Verwenden Sie zum Beispiel die Methode `toArray()` der Klasse `ArrayList`.

3.4 Herausfinden, ob die Daten leer sind

Implementieren Sie die Methode **`boolean`** `isEmpty()`!

Die Methode soll **`true`** zurückgeben, wenn keine Zeichenketten enthalten sind, andernfalls **`false`**.

Tipp: Verwenden Sie zum Beispiel die Methode `isEmpty()` der Klasse `ArrayList`.

3.5 Reflexion im Coding-Team

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

Denken Sie zu diesem Zweck noch einmal an das Szenario einer dynamischen To-Do-Liste, wie es eingangs beschrieben wurde.

3.5.1 Weitere Methoden der Klasse `ArrayList`

Wie können die neuen Methoden die Arbeit mit dynamischen Datenstrukturen in einem To-Do-Listen-Szenario erleichtern?

- Warum ist die Methode `contains()` nützlich, um zu prüfen, ob eine bestimmte Aufgabe bereits in der Liste enthalten ist?
- Wie könnte die Methode `clear()` helfen, wenn eine To-Do-Liste für einen neuen Tag komplett geleert werden muss?
- In welchen Fällen könnte die Methode `toArray()` verwendet werden, um die To-Do-Liste an eine andere Komponente (z. B. ein E-Mail-System) zu übergeben?
- Warum ist es wichtig, mit `isEmpty()` schnell prüfen zu können, ob sich noch Aufgaben in der Liste befinden, z. B. am Ende eines Arbeitstages?

3.5.2 Vorteile von Java-Bibliotheken

Warum ist es nützlich, dass Java eine umfangreiche Bibliothek mit Datenstrukturen wie `ArrayList` zur Verfügung stellt?

- Wie kann die Verwendung von `ArrayList` im Vergleich zu einem Array den Entwicklungsprozess einer To-Do-Liste erleichtern?
- Warum ist es vorteilhaft, auf von Java getestete Standardklassen zurückzugreifen, anstatt eine eigene Datenstruktur für die To-Do-Liste zu entwickeln?
- Inwiefern erleichtern Standardklassen wie `ArrayList` die Zusammenarbeit im Team, z. B. durch einheitliche Verwendung und Pflege des Codes?
- Wie können Sie in zukünftigen Projekten von den umfangreichen Java-Bibliotheken profitieren, wenn es um dynamische Datenstrukturen geht?

4 Imports und Packages

Lernziele

- ☐ Sie verwenden `import`-Anweisungen, um Klassen aus anderen Paketen zu verwenden.
- ☐ Sie unterscheiden zwischen der Verwendung von Klassen ohne und mit `import`-Anweisungen.
- ☐ Sie organisieren Ihren Code in Paketen, um ihn zu strukturieren und zu verwalten.
- ☐ Sie kompilieren und führen Ihren Code auf der Konsole aus.

In dieser Aufgabe beschäftigen Sie sich mit Imports und Packages zur Strukturierung und Organisation von Java-Programmen. Dazu wechseln Sie im letzten Teil der Aufgabe vom OneCompiler auf die Konsole, um den Code auf dem eigenen Rechner zu kompilieren und auszuführen.

Tipp: Wenn Sie Schwierigkeiten mit der Konsole im Allgemeinen, mit der Ausführung der Befehle `java` und `javac` im Besonderen oder mit der Strukturierung in Pakete haben, benutzen Sie bitte das Forum Gemeinsamer Austausch für Ihre Fragen!

Heads up! Vergewissern Sie sich, dass Ihre Lösungen richtig sind, indem Sie sie testen!

4.1 Imports

In den Aufgaben 2 und 3 haben Sie die Klasse `ArrayList` verwendet, nachdem Sie diese Klasse aus dem Paket `java.util` importiert haben:

```
import java.util.ArrayList;
```

Entfernen Sie die Import-Anweisung aus Ihrer Lösung zu Aufgabe 3 (alternativ können Sie auch die Lösung zu Aufgabe 2 verwenden) und bringen Sie den Code ohne die Import-Anweisung zum Kompilieren!

Heads up! Stellen Sie durch Ihre Tests aus der vorherigen Aufgabe sicher, dass die Funktionalität erhalten bleibt.

4.2 Packages

Teilen Sie Ihren Test- und Produktionscode aus Aufgabe 3 (alternativ können Sie auch die Lösung zu Aufgabe 2 verwenden) in geeignete Pakete auf!

Heads up! Dazu müssen Sie vom OneCompiler auf die Konsole wechseln, da der OneCompiler keine Paketstrukturen unterstützt. Falls Sie die notwendigen Programmierwerkzeuge noch nicht installiert haben, laden Sie die OpenJDK-Distribution Eclipse Temurin von Adoptium herunter und installieren Sie diese!

4.2.1 `javac` und `java`

Schreiben Sie den Java-Code mit einem Texteditor Ihrer Wahl und verwenden Sie die Konsole, um den Code zu kompilieren und auszuführen:

Kompilieren: `javac <Klassenname1>.java <Klassenname2>.java ...`

Ausführen: `java <Klassenname>.java`

Nehmen wir an, Sie haben diese Verzeichnisstruktur:

```
main/
    Calculator.java
test/
    CalculatorTestDrive.java
```

Verwenden Sie dann die folgenden Befehle im Verzeichnis, das die Ordner `main` und `test` enthält:

Kompilieren: `javac main/Calculator.java test/CalculatorTestDrive.java`

Ausführen: `java test.CalculatorTestDrive`

4.2.2 Code in Packages organisieren

Folgen Sie dem obigen Beispiel und organisieren Sie Ihren Code in den Paketen `main` und `test`!

Nehmen Sie dazu die beiden Klassen aus Aufgabe 3 (oder Aufgabe 2) und legen Sie sie in den entsprechenden Verzeichnissen bzw. Paketen ab. Vergessen Sie nicht, die notwendigen **`package`**- und **`import`**-Anweisungen hinzuzufügen!

Heads up! Stellen Sie durch Ihre Tests sicher, dass die Funktionalität erhalten bleibt.

4.3 Reflexion im Coding-Team

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

4.3.1 Technische Grundlagen

Wie unterstützen Imports und Packages die Organisation und Verwendung von Java-Code auf technischer Ebene?

- Warum ist es wichtig, den Zusammenhang zwischen Paketstruktur und Verzeichnispfaden zu verstehen?
- Wie unterscheidet sich die Verwendung von Import-Anweisungen von der direkten Verwendung vollständiger Paketnamen (z. B. `java.util.ArrayList`)?
- Wie wirkt sich die Verwendung oder Nichtverwendung von Import-Statements auf die Lesbarkeit und Wartbarkeit des Codes aus?
- Wie können Sie sicherstellen, dass Ihre Paketstruktur korrekt aufgebaut ist, um Fehler beim Kompilieren und Ausführen zu vermeiden?

4.3.2 Organisation und Skalierbarkeit

Wie unterstützen Imports und Packages die Organisation, Wartbarkeit und Skalierbarkeit von Java-Projekten?

- Warum ist es sinnvoll, Produktions- und Testcode in getrennten Paketen (z. B. `main` und `test`) zu organisieren?
- Welche Vorteile hat eine durchdachte Paketstruktur für größere Projekte mit vielen Klassen und Teams?
- Welche Herausforderungen können entstehen, wenn Pakete unklar oder inkonsistent organisiert sind?
- Inwiefern trägt eine klare Paketstruktur zur Übersichtlichkeit und Wiederverwendbarkeit des Codes bei?

4.3.3 Praktische Anwendung und Herausforderungen

Welche Herausforderungen und praktischen Erfahrungen haben Sie bei der Arbeit mit der Konsole, Imports und Packages gemacht?

- Was waren die größten Schwierigkeiten bei der Arbeit mit der Konsole (z. B. `javac` und `java`) und wie konnten Sie diese lösen?
- Welche typischen Fehler (z. B. “Package does not exist” oder “Cannot find symbol”) sind aufgetreten und wie haben Sie diese behoben?
- Warum ist es nützlich, die Grundlagen der Java-Kompilierung zu verstehen, auch wenn man normalerweise eine IDE (Abkürzung für Integrated Development Environment) oder ein Werkzeug wie OneCompiler verwendet?
- Was sind die Vor- und Nachteile der Arbeit mit der Konsole im Vergleich zu einer IDE oder dem OneCompiler?