Heinrich Heine Universität Düsseldorf

Programmierung – WS 2025/26
Dr. Markus Brenneis
Abgabefrist für Feedback:
01.12.25, 10 Uhr

7. Übungsblatt

Denken Sie an die Anmeldefrist für den ersten Zulassungstest (Freitag, 28.11.25). Verspätete Anmeldungen sind nicht möglich. Für diesen Zulassungstest sind die Inhalte bis einschließlich Blatt 7 relevant.

Aufgabe 1: Maybe nach VL 10

Es gibt Berechnungen, die manchmal keinen sinnvollen Rückgabewert haben können. Was ist z. B. der Durchschnitt aller Zahlen in einem leeren `int`-Array?

Es gibt verschiedene Strategien, mit diesen „nicht vorhandenen“ Werten umzugehen. Eine Strategie finden Sie in der Codevorgabe zu dieser Aufgabe umgesetzt.

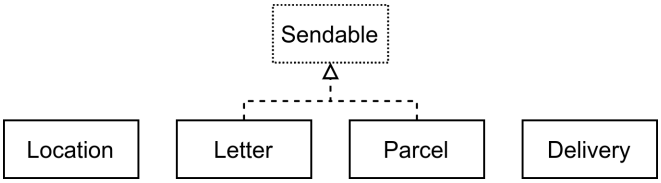
1. Schauen Sie sich die Codevorgabe an. Beantworten Sie dann die Fragen in der Datei `fragen.md`[1]. Wenn Sie individuelle Rückmeldungen zu Ihren Antworten haben wollen, schreiben Sie `// Feedback gewünscht` oben in die Datei und geben Sie die Datei `fragen.md` ab.
2. Fügen Sie eine Aktion zum Quadrieren von `MaybeInt`s hinzu:
   (a) Schreiben Sie eine Klasse `Square`, die `Int2IntFunction` implementiert. Die `run`-Methode soll das Quadrat der übergebenen Zahl zurückgeben.
   (b) Entkommentieren Sie den Code zum Testen in `Test.java` und prüfen Sie, ob er wir gewünscht funktioniert.
3. Fügen Sie eine Aktion hinzu, um zu prüfen, ob ein `MaybeInt` positiv ist:
   (a) Definieren Sie ein Interface `Int2BooleanFunction`, das eine Methode `boolean run(int)` vorschreibt.
   (b) Schreiben Sie eine Klasse `IsPositive`, die `Int2BooleanFunction` implementiert. Die `run`-Methode soll genau dann `true` zurückgeben, wenn die übergebene Zahl größer als 0 ist.
   (c) Erweitern Sie `MaybeInt` um eine Methode `boolean fulfills(Int2BooleanFunction f)`, die `f.run(value)` zurückgibt. Falls die `MaybeInt`-Instanz leer ist, soll `false` zurückgegeben werden.
   (d) Entkommentieren Sie den Code zum Testen in `Test.java` und prüfen Sie, ob er wir gewünscht funktioniert.

[1] Die Datei können Sie in Ihrem Texteditor öffnen.

Aufgabe 2: Post nach VL 13

Diese Aufgabe ist angelehnt an eine Aufgabe aus einer Altklausur und könnte so ähnlich auch in den Klausuren in diesem Jahr vorkommen. In dieser Aufgabe werden verschiedene Aspekte des objektorientierten Entwurfs und der Polymorphie geprüft.

In dieser Aufgabe sollen Sie Klassen und Methoden für den Versand von Briefen und Paketen zwischen verschiedenen Standorten eines Unternehmens programmieren.



Hinweise:

- Lesen Sie sich die Aufgabenstellung vor Beginn der Implementierung **vollständig** durch, um einen besseren Überblick über das Gesamtbild zu erhalten.
- Wählen Sie sinnvolle Datentypen für Ihre Variablen.
- Objekt- und Klassenvariablen müssen minimale Sichtbarkeit haben.
- Sofern nicht anders durch die Aufgabenstellung oder Java vorgegeben, sollen Interfaces, Klassen, Konstruktoren und Methoden **paket-privat** sein und Klassen *nicht* abstrakt.
- An Stellen, wo Klassen verlangt sind, dürfen Sie auch Records benutzen, wenn diese mit den Anforderungen der Aufgabenstellung kompatibel sind.
- Innerhalb dieser Aufgabe müssen Sie nur dann Fehlerfälle behandeln, wenn dies explizit verlangt wird.
- Das vorgegebene Interface darf nicht verändert werden.

Vorgegeben ist bereits ein Interface `Sendable`. Schauen Sie sich die Kommentare im vorgegebenen Interface an.

1. Schreiben Sie einen **Record** `Location`, der einen Standort mit Postleitzahl und Land repräsentiert. Der Konstruktor soll die Signatur `Location(int, String)` haben und die Getter `postcode()` und `country()` heißen.
2. Schreiben Sie eine Klasse `Letter`, die das `Sendable`-Interface sinnvoll implementiert. Der Konstruktor nimmt Absender- und Empfänger-Standort (in dieser Reihenfolge) in Form von `Location`-Objekten entgegen. Jeder Brief wiegt 80 g.
3. Schreiben Sie eine nicht abstrakte Klasse `Parcel`, die das `Sendable`-Interface sinnvoll implementiert. Der Konstruktor nimmt den Absender- und den Empfänger-Standort (`Location`s) und das Gewicht (`double`) in dieser Reihenfolge entgegen.
4. Schreiben Sie eine Klasse `Delivery`, die eine Lieferung bestehend aus einer oder mehreren Sendungen repräsentiert. Ein `Delivery`-Objekt speichert dazu ein Array von `Sendable`-Instanzen. Es gibt zwei Konstruktoren, über die die Sendungen der Lieferung gesetzt werden:
   - `Delivery(Sendable[])` nimmt ein Array von Sendungen entgegen.
   - `Delivery(Sendable)` nimmt eine einzelne Sendung entgegen.

Beide Konstruktoren lösen eine `NullPointerException` aus, wenn das übergebene Objekt oder ein Element im Array `null` sind. Wenn einem Konstruktor ein änderbares Objekt übergeben wird, wird außerdem eine defensive Copy angelegt.

Schreiben Sie eine **private** Klassenmethode `int postage(Sendable)`, die die Portokosten einer einzelnen Sendung in Euro berechnet:

- Sendungen unter 100 g: 1 €
- Sendungen ≥ 100 g: 2 €
- Sendungen, bei denen Empfangs- und Absende-**Land unterschiedlich** sind: zusätzlich 1 €

Schreiben Sie eine Instanzmethode `int postage()`, die das Gesamtporto für die Sendungen der Lieferung zurückgibt.

5. Ersetzen Sie in der `main`-Methode der Klasse `DispatchOffice` die beiden `null`s, sodass eine Lieferung bestehend aus dem angelegten `Letter`-Objekt erstellt wird, und das Gesamtporto mithilfe der `Delivery`-Instanz berechnet und ausgegeben wird.

Aufgabe 3: Widgets nach VL 13

Als Widgets bezeichnet man Elemente in grafischen Benutzeroberflächen (graphical user interfaces, GUIs), wie z. B. Textfelder oder Buttons. In dieser Aufgabe betrachten wir eine einfache[2] Widget-Sammlung, die auf StdDraw basiert. Kompilieren Sie zunächst die Klasse `Application` in den vorgegebenen Dateien und schauen Sie sich das Ergebnis an.

Machen Sie sich mit der Vererbungshierarchie der vorgegebenen Klassen vertraut. Beantworten Sie dann die Fragen, die Sie in der vorgegebenen Datei `fragen.md` finden. Wenn Sie individuell Rückmeldung zu Ihren Antworten erhalten wollen, schreiben Sie oben in die Datei `// Feedback gewünscht` und laden Sie sie im Abgebesystem hoch. Als Antwort reichen jeweils 1–3 Sätze oder nachvollziehbaren Stichpunkte.

[2] „einfach“ im Sinne von „sieht nur einigermaßen gut aus, kann man aber nicht anklicken oder sinnvoll verwenden“. Die Vererbungshierarchie, die Sie in dieser Aufgabe kennenlernen, ist aber typisch für „echte“ Klassen zum Erstellen grafischer Benutzeroberflächen (siehe z. B. https://openjfx.io/javadoc/21/javafx.controls/javafx/scene/control/package-tree.html). Wir begnügen uns hier mit der „einfacheren“ Variante, damit Sie die Implementierung aller Widgets noch nachvollziehen können.