Universität Augsburg
Institut für Informatik

Übung zur Vorlesung Informatik 2 SoSe 2024
Fakultät für Angewandte Informatik
Institut für Informatik
Prof R. Lorenz, L. Petrak, J. Linne, V. Le Claire, P. Schalk 31.05.2024

Übungsblatt 5

Abgabe: 10.06.2024, 10:00 Uhr (im Digicampus via VIPS: .java-Dateien für Code, .uxf für UML, .pdf für alles andere)

- Dieses Übungsblatt muss im Team abgegeben werden (Einzelabgaben sind nicht erlaubt!).
- Die **Zeitangaben** geben zur Orientierung an, wie viel Zeit für eine Aufgabe später in der Klausur vorgesehen wäre; gehen Sie davon aus, dass Sie zum jetzigen Zeitpunkt wesentlich länger brauchen und die angegebene Zeit erst nach ausreichender Übung erreichen.

* leichte Aufgabe / ** mittelschwere Aufgabe / *** schwere Aufgabe

Aufgabe 17 ** *(Polymorphismus / Schnittstellen, 20 Minuten)*

a) *(10 Minuten)*
Erklären Sie die Begriffe **Substitutionsprinzip**, **spätes Binden** und **Polymorphismus** anhand des Eingabeparameters und der Implementierung der `contains`-Methode der Klasse `ArrayList`.

b) *(10 Minuten)*
Erklären Sie die Vorteile der Benutzung von Schnittstellen als Datentypen anhand des Eingabeparameters der `removeAll`-Methode der Klasse `ArrayList`.

Aufgabe 18 ** *(Eigene Klasse und Containerklasse, 30 Minuten)*

In dieser Aufgabe sollen Sie eine eigene Datenklasse für die Verwaltung von Gepäckstücken implementieren. Dabei üben Sie auch das Werfen von Ausnahmen.

a) *(**, 24 Minuten)*
Implementieren Sie eine eigene Klasse `Luggage` für Gepäckstücke mit passenden set-, get- und check-Methoden:

- Ein Gepäckstück hat eine Flug-ID, einen Besitzer und ein Gewicht.
- Der Besitzer wird durch eine nicht-leere Zeichenkette dargestellt.
- Das Gewicht ist eine positive reelle Zahl.
- Die Flug-ID ist von der Form `"XXYY"`, wobei `X` für einen beliebigen Großbuchstaben und `Y` für eine beliebige Ziffer steht. Benutzen Sie einen geeigneten regulären Ausdruck[1], um diese Einschränkung darzustellen.
- Ist eine der Datenstruktur-Invarianten verletzt, soll die zugehörige set-Methode eine `IllegalArgumentException` werfen, in welcher der aufgetretene Fehler **kurz und knapp** beschrieben ist.
- Zwei `Luggage`-Objekte gelten als gleich, wenn ihre Flug-ID und ihr Besitzer übereinstimmen.
- Der Hashcode eines `Luggage`-Objekts soll aus seiner Flug-ID und dem Namen seines Besitzers berechnet werden. Überschreiben Sie dafür die `hashCode`-Methode und verwenden Sie dafür eine geeignete Klassenmethode der Klasse `Objects` im Paket `java.util`.

b) *(**, 8 Minuten)*
Implementieren Sie eine Klasse `LuggageContainer` zum Verwalten aller Gepäckstücke: Der Container bietet...

- mit `void addLuggage(Luggage l)` eine Möglichkeit an, ein `Luggage`-Objekt hinzuzufügen.
- mit `void removeLuggage(Luggage l)` eine Möglichkeit an, ein `Luggage`-Objekt zu entfernen.
- eine Mögichkeit an, über alle enthaltenen Elemente zu iterieren und implementiert dafür eine geeignete Schnittstelle.
- mit `Stream<Luggage> stream()` die Rückgabe eines Streams aller enthaltenen `Luggage`-Objekte an.

c)
Testen Sie Ihre Implementierungen mit der im DigiCampus zur Verfügung gestellten Programmklasse `LuggageMain.java`.

[1] Siehe https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/regex/Pattern.html. Für diese Aufgabe sind die Zusammenfassungen zu **Character classes** und **Greedy Quantifiers** vollkommen ausreichend.

Aufgabe 19 ** *(Exceptions abfangen und werfen, 25 Minuten)*

In den folgenden Teilaufgaben sollen Sie lernen, wie Sie Exceptions abfangen und werfen können.

a) *(**, 10 Minuten)*
Laden Sie die im Digicampus gegebene Klasse `FaultyCode` herunter. Sorgen Sie durch maximal spezifisches Abfangen aller auftretenden Exceptions dafür, dass das Programm fehlerfrei terminiert. Führen Sie dazu das Programm aus und verwenden Sie die ausgegebenen Stack Traces, um nach und nach alle Anweisungen, die Exceptions werfen, mit einem try-catch Block umgeben und mit geeigneten Methoden die Klasse und die enthaltene Fehlermeldung aller geworfenen Ausnahmen auf der Kommandozeile ausgeben. Umgeben Sie jeden Aufruf, der eine Exception wirft, mit einem eigenen try-catch Block.
Lassen Sie die Anweisungen des Programms sonst unverändert!

b) *(***, 15 Minuten)*
Schreiben Sie eine Programmklasse, die einen primitiven Taschenrechner implementiert. Der Taschenrechner soll in der Lage sein, zwei reellwertige Zahlen zu addieren, subtrahieren, multiplizieren und zu dividieren. Die Ein- und Ausgabe soll über die Kommandozeile erfolgen. Pro Zeile soll eine Operation eingegeben werden, die Verwendung von Klammern ist nicht erlaubt. Das Programm soll

- zu Beginn kurz über seine Benutzung informieren,
- solange zeilenweise Rechnungen einlesen, bis die Zeichenkette `"EXIT"` eingegeben wird,
- die Eingabe in zwei Zahlen und den verwendeten Operator aufspalten und die Rechnung mit ihrem Ergebnis ausgeben.

Beachten Sie außerdem folgende Vorgaben zur Implementierung:

- Implementieren Sie für die Berechnung des Ergebnisses eine eigene Klassenmethode, die auch das Ausgeben des Ergebnisses übernimmt.
- Verwenden Sie für das Aufteilen der Eingabe den regulären Ausdruck `"[\\x2B\\x2D\\x2A\\x2F]"`[2]
zusammen mit einer geeigneten Objektmethode der Klasse `String`.
- Bei der Nutzereingabe können verschiedene Fehler auftreten. Behandeln Sie die folgenden Fehler jeweils mit dem Werfen der angegebenen Ausnahme:
  - Es wird kein gültiger Operator übergeben: `IllegalArgumentException`
  - Es werden zu viele gültige Operatoren übergeben `IllegalArgumentException`
  - Es werden keine gültigen Zahlen als Operanden übergeben: `NumberFormatException`

  Wenn möglich, reichen Sie die von API-Methoden geworfenen Ausnahmen weiter.
- Behandeln Sie die auftretenden Ausnahmen in der `main`-Methode maximal spezifisch und geben Sie eine jeweils eine passende Fehlermeldung auf dem Standard-Fehlerstrom aus.

[2] \x2B, 2D, 2A, 2F sind die Hexadezimalwerte der Zeichen `'+'`, `'-'`, `'*'`, `'/'`.

Aufgabe 20 ** *(Eigene Datenklassen implementieren, 30 Minuten)*

In dieser Aufgabe sollen Sie nach einer Systembeschreibung mehrere eigene Datenklassen implementieren und diese mit einer eigenen Programmklasse testen.

a) *(**, Klassen `RealEstate`, `Parcel` und `Apartment` nach Systembeschreibung implementieren, 22 Minuten)*
Implementieren Sie eigene Datenklassen `RealEstate`, `Parcel` und `Apartment` für eine Anwendung zum Anbieten von **Immobilien** (real estate), d.h. Grundstücke (parcel) und Apartments, gemäß folgender Systembeschreibung.

Es sollen Grundstücke und Apartments angeboten werden können. Für Grundstücke ist eine Größe anzugeben. Es dürfen nur Grundstücke, die nicht weniger als 250 *Quadratmeter haben, angeboten werden. Für Apartments ist die Wohnfläche anzugeben – diese darf nicht weniger als* 20 *und nicht mehr als* 120 *Quadratmeter betragen. Außerdem haben alle Immobilien einen Preis, der bei mindestens* 10 000 *Euro liegen muss, und bei dem nur Vielfache von* 1000 *erlaubt sind. Zudem soll es möglich sein, für jede Immobilie den Preis pro Quadratmeter zu berechnen. Schließlich ist für jeden Verkauf eine Maklerprovision in Höhe von* 3.45 *Prozent fällig. Es soll jeweils eine geeignete Möglichkeit geben, Grundstücke und Apartments anzulegen. Ungültige Preise sollen eine geprüfte Ausnahme vom Typ `IllegalPriceException` und ungültige Grundstücksgrößen und Wohnflächen eine geprüfte Ausnahme vom Typ `IllegalSizeException` auslösen (die Sie in der nächsten Teilaufgabe selbst implementieren).*

Hinweis: Da sich in der Systembeschreibung dazu keine Vorgaben befinden, müssen Sie sich hier nicht um eine spezifische Zeichenketten-Ausgabe und nicht um einen geeigneten Test auf Gleichheit kümmern.

b) *(*, Eigene Ausnahmeklassen implementieren, 4 Minuten)*
Implementieren Sie geprüfte Ausnahmeklassen `IllegalPriceException` und `IllegalSizeException` jeweils mit einem Konstruktor, der das Setzen einer individuellen Fehlermeldung erlaubt (weitere Informationen sollen nicht gespeichert werden).

c) *(*, Programmklasse `ImmoApp` implementieren, 4 Minuten)*
Implementieren Sie gemäß folgender Vorgaben eine eigene Programmklasse `ImmoApp` zum Test der vorher implementierten Klassen:

- Erstellen Sie ein Grundstück mit einem Preis von 500 000 Euro und einer Größe von 1000 Quadratmetern und geben Sie den daraus in der Klasse berechneten Preis pro Quadratmeter aus. Sorgen Sie hierbei für eine geeignete Ausnahmebehandlung. Für ungültige Größen soll dabei der Stapel der zur Ausnahme führenden Methodenaufrufe ausgegeben werden, für ungültige Preise nur die Fehlermeldung.
- Erstellen Sie ein Apartment mit einem Preis von 450 000 Euro und einer Wohnfläche von 150 Quadratmetern und geben Sie den daraus in der Klasse berechneten Preis pro Quadratmeter aus. Sorgen Sie hierbei für eine geeignete Ausnahmebehandlung. Für ungültige Größen soll dabei der Stapel der zur Ausnahme führenden Methodenaufrufe ausgegeben werden, für ungültige Preise nur die Fehlermeldung.