252-0027-00: Einführung in die Programmierung

Übungsblatt 10

Abgabe: 3. Dezember 2019, 10:00

Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus. Vergessen Sie nicht, Tests zu schreiben! Auch Ausnahmefälle (Exceptions) können mit JUnit getestet werden, z.B. so:

```
boolean thrown = false;
try {
    // code
} catch (SomeException e) {
    thrown = true;
}
if (!thrown) {
    fail("expected some exception");
}
```

Aufgabe 1: Wahlen (Bonus!)

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe "Leistungskontrolle" im `www.vvz.ethz.ch`). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin.

In der letzten Bonusaufgabe sollten Sie zeigen, dass Sie mit der Theorie von Klassen und Vererbung umgehen können. In dieser Aufgabe geht es um die praktische Anwendung von Klassen und Interfaces. Sie werden ein Programm vervollständigen, welches Wahlergebnisse eines Kantons auswertet nach dem Doppeltproportionalen Zuteilungsverfahren (Wir geben weiter unten eine kurze Beschreibung). Ein Grossteil des Zuteilungsverfahrens ist bereits implementiert und wird von der Methode `wahlauswertung` aus der Datei "Wahlen.java" aufgerufen. Die Methode nimmt als Parameter einen zweidimensionalen Array `stimmen` (`stimmen[i][j]` enthält die Anzahl gezählter Stimmen aus dem Wahlkreis mit dem Index `i` für die Partei mit dem Index `j`), einen eindimensionalen Array `sitzeProWahlkreis` (`sitzeProWahlkreis[i]` enthält die Anzahl verfügbarer Sitze für den Wahlkreis `i`), eine Zuteilungsstrategie `verteilung` und eine Rundungsstrategie `runder`. Die Methode gibt einen zweidimensionalen Array zurück, welcher an der Stelle `[i][j]` für den Wahlkreis mit Index `i` die Sitze der Partei mit Index `j` enthält. Jeder Wahlkreis hat die gleiche Anzahl Parteien, wobei sowohl die Anzahl Wahlkreise als auch die Anzahl Parteien durch die beiden Dimensionen des Arrays `stimmen` gegeben sind.

Ihre Aufgabe ist es, den restlichen Code an den fehlenden Stellen hinzuzufügen, welche mit `TODO` markiert sind. An jeder dieser Stellen ist explizit geschrieben was verlangt wird. Man kann die Aufgabe mit ungefähr 30 Zeilen Code lösen. Die Hauptherausforderung ist es, mit Interfaces umzugehen und nicht von grösserem Code überwältigt zu werden. Sie können neue Interfaces, Klassen, Methoden und Felder hinzufügen, jedoch dürfen Sie keine der gegebenen Deklarationen ändern (ausser wenn es explizit geschrieben ist). Sie dürfen einer Klasse oder einem Interface immer weitere Interfaces oder Klassen in der `extends` oder `implements` Klausel hinzufügen.

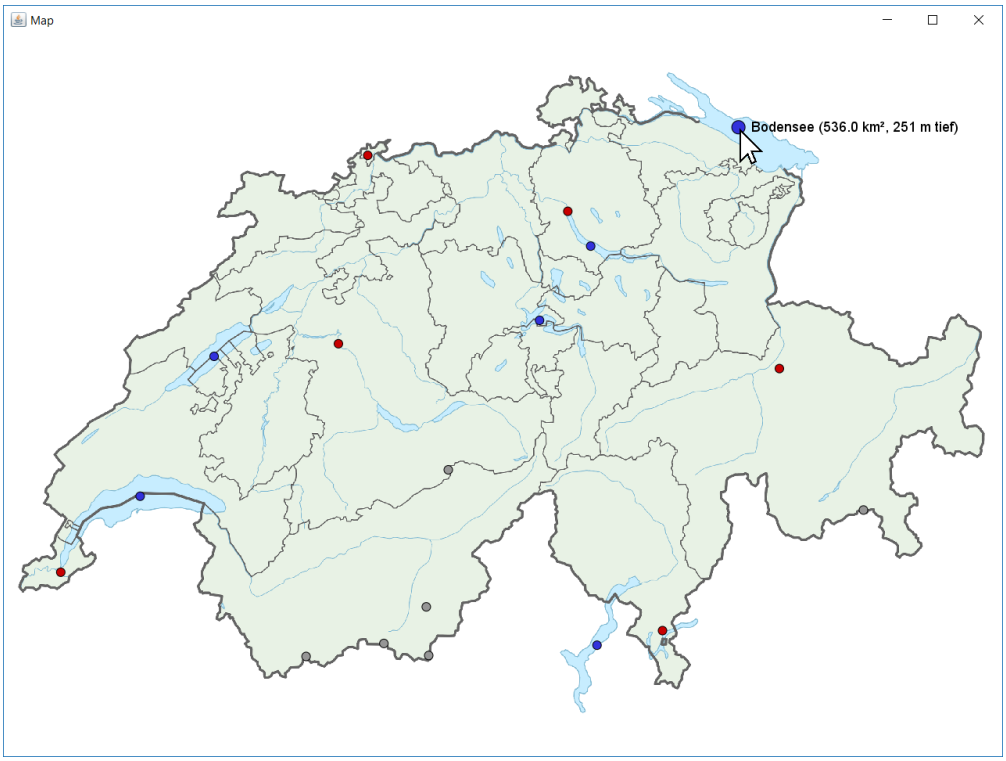
Die Datei "WahlenTest.java" enthält Tests, um Ihre Lösung zu testen. Das Verfahren liefert nicht für alle Inputs eine Lösung. In der Realität würde eine Kommission Uneindeutigkeiten auflösen. Für uns reicht es jedoch in diesen Fällen `null` zurückgegeben. Sie müssen das Verfahren nicht anpassen und können davon ausgehen, dass nur Inputs mit einem eindeutigen Ergebnis zum testen verwendet werden.

Nun zur Erklärung des Verfahrens: Wir geben hier eine kürzere Erklärung; bei Fragen folgen Sie dem oberen Link (enthält auch Beispiele) oder sehen Sie sich den zur Verfügung gestellten Code und die Tests an. Das Doppeltproportionale Zuteilungsverfahren ist in zwei Phasen unterteilt, die Oberzuteilung und die Unterzuteilung. In der Oberzuteilung werden jeder Partei eine Anzahl Sitze zugeteilt. Dafür werden zuerst für jede Partei die Summe aller Stimmen über alle Wahlkreise berechnet. Zusätzlich wird die Summe k aller gesamt verfügbaren Sitze berechnet. Danach wird ein sogenanntes Höchstzahlverfahren angewendet: Jede Zahl der Stimmen einer Partei wird durch eine Reihe von Divisoren geteilt (zum Beispiel $0.5, 1.5, 2.5, 3.5, ...$), wobei die Ergebnisse aufgelistet werden. Eine Partei erhält dann i Sitze, wenn die Partei i der k grössten aufgelisteten Einträge hat. Wir parametrisieren die Oberzuteilung mit einer Sitzzuteilungsstrategie, welche die verwendete Reihe der Divisoren vorgibt und in der `wahlauswertung` Methode als Parameter übergeben wird. Definieren Sie hierfür Methoden in dem Interface `SitzzuteilungStrategy` und implementieren Sie zwei Klassen welche das Sainte-Laguë-Verfahren (Divisoren Reihe $0.5, 1.5, 2.5, 3.5, ...$) und das D'Hondt-Verfahren (Divisoren Reihe $1, 2, 3, 4, ...$) implementieren. Zusätzlich implementieren Sie die Methoden `getSainteLague` und `getDHondt` der `SitzzuteilungStrategyFactory` Klasse, damit wir in unseren Tests die entsprechenden Strategien instanzieren können. All diese Anweisungen stehen auch im Code, mit `TODO` markiert.

Bei der Unterzuteilung werden nach der Oberzuteilung die genauen Sitze jeder Partei für jeden Wahlkreis zugeordnet. Es ist ein iteratives Verfahren, welches angewendet wird, bis eine Lösung gefunden wird. Zuerst wird für jeden Wahlkreis ein Divisor gesucht, so dass wenn die Stimmen der Parteien in dem Wahlkreis durch den Divisor geteilt werden, gerundet werden, und dann aufaddiert werden, man die Anzahl verfügbarer Sitze in dem Wahlkreis erhält. Beachten Sie, dass nach diesem Schritt die Stimmen Dezimalzahlen sein können (im gegebenen Code nennen wir diese meist "Bruchteile von Sitzen"), wobei der gerundete Wert der Anzahl Sitze für eine Partei in einem Wahlkreis entsprechen wird. Als nächstes wird das Gleiche für jede Partei gemacht, nur dass die Summe danach der Sitze der Partei entsprechen soll. Ein gutes Beispiel finden Sie hier unter "Unterzuteilung". Da beim Anpassen einer Folge für einen Wahlkreis die Folge für eine Partei geändert werden kann, wird dieses Verfahren wiederholt, bis die Summe der gerundeten Bruchteile von Sitzen in jeder Folge für jeden Wahlkreis und jede Partei der Anzahl verfügbarer Sitze entspricht. Ähnlich wie die Zuteilungsstrategie, wird die Unterzuteilung parametrisiert mit einer Rundungsstrategie, welche vorgibt wie gerundet werden soll. Definieren Sie dafuer die noetigen Methoden in dem Interface `RundungStrategy` und implementieren Sie die getter in der `RundungStrategyFactory` Klasse. Weitere Anweisungen stehen im Code.

Aufgabe 2: Interaktive Karte

In dieser Übung verwenden Sie die `Window`-Klasse um eine interaktive Karte zu erstellen. Auf der Karte werden verschiedene *points of interest* (POI) angezeigt. Wenn der Benutzer mit der Maus auf einen solchen POI zeigt, sollen einige Informationen dazu angezeigt werden:



In der Vorlage befindet sich bereits ein Skelett für `SwissMap`, welches eine Window-Instanz erstellt und ein Hintergrundbild darin anzeigt. Ausserdem finden Sie die `PointOfInterest`-Klasse, welche die Basis-Klasse für verschiedene Arten von POIs bildet. Ihre Aufgabe ist es, drei Subklassen von `PointOfInterest` (d.h. Klassen, die von `PointOfInterest` erben) zu erstellen: `City`, `Lake` und `Mountain`. Danach sollen Sie einige Instanzen dieser Klassen erstellen und auf Ihrer Karte anzeigen.

a) Erstellen Sie eine neue Klasse `City`, welche von `PointOfInterest` erbt. Diese Klasse soll zusätzlich zu den (geerbten) Feldern von `PointOfInterest` noch Felder für Einwohnerzahl und Fläche haben. Erstellen Sie auch einen Konstruktor, welcher Werte für alle geerbten und für alle eigenen Felder von `City` entgegen nimmt. Um die geerbten Felder zu initialisieren, sollen Sie mithilfe von `super(...)` den Superkonstruktor aufrufen.

   Gehen Sie analog für `Lake` (mit Fläche und Tiefe) und `Mountain` (mit Höhe) vor.

b) Überschreiben Sie in allen drei Klassen `City`, `Lake` und `Mountain` die beiden geerbten Methoden `color()` und `description()`.

   Die erste Methode, `color()`, soll die Farbe zurückgeben, in der ein POI angezeigt werden soll. Seen sollen blau angezeigt werden, Städte rot, usw. Beachten Sie, dass der Rückgabetyp `Color`

eine Klasse ist, die von der "Window.java"-Datei kommt und einfach drei RGB-Werte speichert. `Color`-Objekte können direkt an `Window.setColor()` übergeben werden.

Die zweite Methode, `description()`, soll eine Beschreibung zurückgeben, welche angezeigt wird, wenn der Benutzer die Maus über einen POI bewegt. Die Version in `PointOfInterest` gibt einfach den Namen des Punktes zurück. Überschreiben Sie diese Methode in den drei Subklassen, um eine spezifischere Beschreibung für jeden Typ von POI zurückzugeben (siehe Beispiel oben).

c) Vervollständigen Sie die `SwissMap`-Klasse. Erstellen Sie erst einige POIs, z.B. so:

```
PointOfInterest[] pois = new PointOfInterest[] {
        new City("Zürich",         683354, 247353, 396030, 91.88),
        new Lake("Bodensee",       744895, 277632,    536, 251),
        new Mountain("Matterhorn", 617049,  91670,   4478)};
```

In Anhang A finden Sie eine längere Liste von POIs, die Sie kopieren können. Um die Koordinaten von weiteren POIs herauszufinden, besuchen Sie am einfachsten die entsprechende Wikipedia-Seite und kopieren die "CH1903"-Koordinaten oben rechts des Artikels.

Ergänzen Sie dann die `show()`-Methode so, dass diese POIs angezeigt werden. Um die "realen" Koordinaten in GUI-Koordinaten umzuwandeln, können Sie die `toGuiX()`- und `toGuiY()`-Methoden verwenden. Die Informationen eines POIs sollen nur angezeigt werden, wenn sich der Mauszeiger in der Nähe befindet. Verwenden Sie dazu die `getMouseX()`- und `getMouseY()`-Methoden der `Window`-Klasse, welche die aktuelle Position des Mauszeigers angeben.

Aufgabe 3: Exceptions

Das Programm `BmiChecker` kennen Sie bereits von Übungsblatt 6. Es liest Daten über Personen ein und gibt eine Liste der "ungesunden" Personen aus (gemäss Body Mass Index und WHO). Leider ist der Datensatz dieses Mal nicht ganz fehlerfrei. Es ist nun Ihre Aufgabe, das Programm so zu verbessern, dass Fehler in den Daten erkannt und behandelt werden.

a) Vergewissern Sie sich, dass das Programm, wenn Sie es unverändert ausführen, mit einem Fehler abbricht (`FileNotFoundException`). Eine solche Fehlermeldung zu verstehen ist für den Benutzer des Programms nicht einfach. Deshalb wollen wir diesen Fehler abfangen und stattdessen eine einfachere Fehlermeldung ausgeben, z.B. "Error: The file body1.dat.txt could not be found".

Aus der Vorlesung wissen Sie, wie Sie mit `try-catch` diese Exception direkt in der `main()`-Methode behandeln können. Ändern Sie nun die Methode so ab, dass die obige Meldung erscheint (vergessen Sie nicht, die `throws`-Deklaration zu entfernen) und beheben Sie dann den eigentlichen Fehler, sodass die Datei gefunden wird.

b) Bei erneutem Ausführen des Programms sehen Sie den nächsten Fehler (`InputMismatchException`). Diese Exception wird von `Scanner.nextInt()` im Konstruktor der Klasse `Person` geworfen und bedeutet, dass der Scanner keinen `int` als nächstes Token finden konnte. Anscheinend enthielt eine Zeile (Parameter `dataRow`) in der Datei "body.dat.txt" fehlerhafte Daten. Nun

können wir uns leicht weitere Fehler in den Daten vorstellen: Einerseits wirft `nextInt()` noch die weitere wichtige Exception `NoSuchElementException` (zu wenig Daten in der Zeile) und andererseits können auch die eingelesenen Werte sinnlos sein (z.B. eine negative Körpergrösse). Wir möchten alle diese Fehler unter einer Exception `IllegalPersonFormatException` (schon im Projekt vorgegeben) zusammenfassen.

Das Zusammenfassen aller Fehler unter einer Exception ist insofern sinnvoll, dass der Klient der Klasse `Person` nur mit einer Art Fehler umgehen muss. Wir teilen dem Klienten mit, dass etwas beim Erstellen der Person schief gelaufen ist. Ausserdem enthält die `IllegalPersonFormatException` noch eine Nachricht, die den Fehler genauer beschreibt und vom Klienten über die Methode `getMessage()` von der `IllegalPersonFormatException`-Instanz erfragt werden kann.

Fangen Sie nun mithilfe von `try-catch` die oben genannten Exceptions ab, die von `nextInt()` und `next()` geworfen werden können, und werfen Sie dann selbst eine Exception:

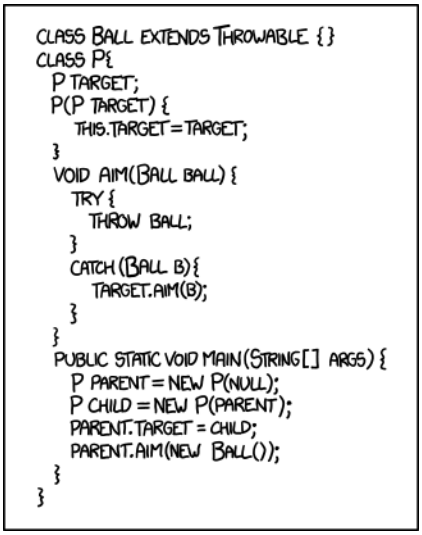
```
throw new IllegalPersonFormatException("Fehlerbeschreibung");
```

Hinweis: Die `IllegalPersonFormatException` ist ein Untertyp der Klasse `Exception` und somit eine "checked" Exception. Das Werfen dieser Exception muss deshalb vom Konstruktor `Person()` per `throws`-Deklaration angekündigt werden.

c) Überlegen Sie sich nun, welche Bedingungen für `age`, `weight`, `height` und `isMale` sinnvoll sind. Fügen Sie entsprechende Checks ein und werfen Sie eine `IllegalPersonFormatException`, falls eine Bedingung nicht erfüllt ist.

d) Da `IllegalPersonFormatException` eine checked Exception ist, muss sie nun auch vom Klienten der Klasse `Person` behandelt werden. Der Klient ist die Methode `readPersons()` der Klasse `BmiChecker`. Schreiben Sie diese Methode so um, dass sie illegale Datensätze **überspringt** und dabei eine Fehlermeldung für jeden solchen Datensatz ausgibt. Verwenden Sie dabei die Methode `getMessage()` der `IllegalPersonFormatException`.

Ihr Programm sollte nun wieder ohne Probleme alle ungesunden Personen ausgeben.



xkcd: Bonding by Randall Munroe (CC BY-NC 2.5)

Anhang B: Mehr POIs

```
PointOfInterest[] pois = new PointOfInterest[] {
        new City("Zürich", 683354, 247353, 396030, 91.88),
        new City("Genf",   500532, 117325, 201810, 15.89),
        new City("Basel",  611220, 267503, 175130, 22.75),
        new City("Bern",   600670, 199655, 141660, 51.60),
        new City("Lugano", 717505,  96295,  63580, 75.80),
        new City("Chur",   759662, 190702,  37110, 28.09),

        new Lake("Bodensee",           744895, 277632, 536,    251),
        new Lake("Genfersee",          529160, 144713, 580,    310),
        new Lake("Neuenburgersee",     555829, 195103, 217.9,  152),
        new Lake("Lago Maggiore",      693884,  91043, 212.5,  372),
        new Lake("Vierwaldstättersee", 673175, 208048, 113.72, 214),
        new Lake("Zürichsee",          691603, 234802,  88.17, 136),

        new Mountain("Dufourspitze", 633220,  87321, 4634),
        new Mountain("Dom",          632330, 104856, 4545),
        new Mountain("Matterhorn",   617049,  91670, 4478),
        new Mountain("Grand Combin", 589008,  86994, 4314),
        new Mountain("Jungfrau",     640278, 154213, 4158),
        new Mountain("Piz Bernina",  789947, 139751, 4049)};
```