Übung 04: Innere Klassen

Abgabetermin: 02.04.2020, 8:15

Aufgabe	Punkte	abzugeben über Moodle	Punkte
Übung 4	24	Java Programmcode in Zip-Datei Java Programmcode als pdf-Datei	

Übung 04: Data Processing (24 Punkte)

Da in den letzten Jahren überall Buzzwords wie Big Data, AI, Machine Learning, Data Science und Co. großgeschrieben werden, wollen Sie sich nun ebenfalls mit dieser Materie beschäftigen. Hierzu wollen Sie ein kleines Programm für Data Processing basteln, das Sie unterstützen soll, CSV-Dateien (comma separated values) einzulesen, diese mit unterschiedlichen Methoden zu bearbeiten und anschließend wieder als CSV zu speichern. Weil Sie unlängst von inneren Klassen gehört haben, wollen Sie dieses neue Konzept hier gleich verwenden.

a) Funktionsweise des Programms

Das Programm soll wie folgt funktionieren:

- In der `main`-Methode der Klasse `Main` wird ein `Data`-Objekt von einer Datei eingelesen.
- Es wird ein `Processor`-Array erzeugt, das mehrere `Processor`s enthält, die zur Verarbeitung des `Data`-Objekts verwendet werden. Die Implementierung dieser `Processor`s müssen Sie erledigen.
- Ein `Processor` verarbeitet nun das `Data`-Objekt und gibt ein neues, verarbeitetes `Data`-Objekt zurück, was wiederum als Input für den nächsten `Processor` dient.
- Das verarbeitete `Data`-Objekt wird am Ende wieder in eine Datei gespeichert.

Der Großteil dieses Programms ist bereits implementiert und steht in den Downloadmaterialien zur Verfügung. Sie müssen unterschiedliche Implementierungen des Interfaces `Processor` umsetzen, die in Punkt **b) Implementierung** beschrieben sind. Siehe dazu die TODOs in der Klasse `Processors`. Um den Umgang mit dem Code zu erleichtern, können Sie sich die JavaDoc durchlesen.

Folgende Klassen sind vorgegeben:

Klasse `Data`:

Die Klasse `Data` repräsentiert die Daten der CSV-Datei. Sie definiert eine öffentliche Schnittstelle wie folgt:

```
public class Data implements Iterable<Double> {
    public Data(double[] values)
    public int size()
    public Iterator<Double> iterator()
    public static Data readFromFile(String path) throws IOException
    public void writeToFile(String path) throws IOException
}
```

- Mit dem Konstruktor kann man ein `Data`-Objekt mit den Datenwerten erzeugen.
- Mit `size()` kann man die Anzahl der Werte abfragen
- Die Klasse implementiert `Iterable<Double>` und man kann dadurch mit der foreach-Schleife über die Datenwerte iterieren, z.B. wie folgt: `for (double val : data)`.
- Mit der statischen Methode `readFromFile(String path)` kann man die Daten aus einer Datei lesen und ein `Data`-Objekt aufbauen.
- Mit `writeToFile(String path)` kann man die Daten in eine Datei schreiben.

Beachten Sie, dass `Data` unveränderlich/immutable ist.

Interface `Processor`:

Ein `Processor` wird zur Verarbeitung eines `Data`-Objekts verwendet.

```
public interface Processor {
    Data process(Data data);
    String getName();
}
```

Das Interface `Processor` definiert nur zwei Methoden wie folgt:

- Die Methode `Data process(Data data)` nimmt ein `Data`-Objekt entgegen, verarbeitet dieses und gibt ein neues, verarbeitetes `Data`-Objekt zurück.
- Die Methode `String getName()` liefert einen beschreibenden Namen.

Klasse `Processors`:

In der Klasse `Processors` sollen Sie mehrere innere Klassen und statische Methoden zum Erzeugen von `Processor`-Objekten implementieren. Siehe **b) Implementierung**.

Klasse `Main`:

Ist der Einstiegspunkt mit bereits fertig implementierter `main`-Methode. Hier können Sie sehen, wie ein `Data`-Objekt von einer Datei geladen wird, unterschiedliche `Processor`s erzeugt werden, diese `Processor`s zur Verarbeitung des `Data`-Objekts verwendet werden und schließlich das verarbeitete `Data`-Objekt wieder in eine Datei geschrieben wird.

Testen Sie Ihr Programm mit der `main`-Methode. Siehe **c) Test**.

b) Implementierung

Diese Klasse `Processors` soll diverse `Processor`-Objekte mittels statischer Methoden liefern. Dazu müssen Sie mehrere unterschiedliche innere Klassen implementieren.

Folgendes sollen Sie in der Klasse `Processors` implementieren:

1) **Öffentliche, abstrakte, statische, innere Klasse** `Scaler`:

- Implementiert `Processor`.
- Hat zwei abstrakte Methoden `double getMin()` und `double getMax()`.
- Die konkrete Methode `process(Data data)` soll die Daten auf das Intervall [`getMin()`, `getMax()`] skalieren. Berechnen Sie hierzu zuerst das Minimum `dataMin` und Maximum `dataMax` der Datenelemente und skalieren Sie dann jeden Datenpunkt `val` von `data` mittels

  ```
  double scaled = (val - dataMin) / (dataMax - dataMin);
  double newVal = scaled * (getMax() - getMin()) + getMin();
  ```

  Ergebnis ist ein neues `Data`-Objekt mit den skalierten Daten.
- Überschreiben Sie `Processor.getName()` mit einem passenden Namen. Der Name soll die Werte von `getMin()` und `getMax()` enthalten.

Hinweis: Die Hilfsklasse `DataUtil` in der Klasse `Processors` stellt Methoden zur Berechnung von Minimum und Maximum zur Verfügung.

2) Statische Methode `Processor scale(double min, double max)`:

- Liefert ein Objekt einer **anonymen Klasse** basierend auf `Scaler`, in der `getMin()` das Argument `min` und `getMax()` das Argument `max` liefern sollen.

3) **Öffentliche, statische, innere Klasse** `PercentScaler`:

- Erbt von `Scaler`, wobei `getMin()` den Wert 0 und `getMax()` den Wert 100 liefern sollen. Die Klasse wird in der `main`-Methode der Klasse `Main` erzeugt und muss daher öffentlich sein.

4) Methode `Processor standardize()`:

- Liefert ein Objekt einer **anonymen Klasse** basierend auf `Processor`, in der die Methode `Processor.process(Data data)` die übergebenen Daten standardisiert. Standardisieren heißt, dass von jedem Datenpunkt `val` der Mittelwert `dataAvg` abgezogen wird und durch die Standardabweichung `dataStd` dividiert wird:

```
double newVal = (val - dataAvg) / dataStd;
```

- Überschreiben Sie `Processor.getName()`, verwenden Sie `"Standardizer"` als Namen.

Hinweis: Die Hilfsklasse `DataUtil` in der Klasse `Processors` stellt Methoden zur Berechnung von Mittelwert und Standardabweichung zur Verfügung.

5) **Private, statische, innere Klasse** Clipper:

- Implementiert `Processor`.
- Ein Konstruktor `Clipper(boolean clipLower, boolean clipUpper, double lower, double upper)` dient zur Einstellung des Clippers.
- Die Methode `Processor.process(Data data)` soll die übergebenen Daten entsprechend abschneiden. Wenn `clipLower` gesetzt ist, müssen alle Datenpunkte in `data`, die kleiner als `lower` sind, auf `lower` gesetzt werden. Wenn `clipUpper` gesetzt ist, müssen alle Datenpunkte in `data`, die größer als `upper` sind, auf `upper` gesetzt werden. Wenn beide boolean-Werte gesetzt sind, muss beides durchgeführt werden.
- Überschreiben Sie `Processor.getName()` mit einem passenden Namen, der die Werte von `lower` und `upper` enthält, sofern die entsprechenden boolean-Werte gesetzt sind.

6) Methoden `clip`

- `public static Processor clip(double lower, double upper)`
- `public static Processor clipLower(double lower)`
- `public static Processor clipUpper(double upper)`

die entsprechende `Clipper`-Objekte erzeugen und zurückgeben.

Hinweise:

- In der Klasse `Processors` steht Ihnen eine statische innere Hilfsklasse `DataUtil` zur Verfügung, die das Minimum, Maximum, den Mittelwert und die Standardabweichung eines `Data`-Objekts berechnet.
- Verwenden Sie zur Iteration der Datenpunkte eines `Data`-Objekts die foreach-Schleife (siehe oben).
- Bedenken Sie, dass ein `Data`-Objekt unveränderlich/immutable ist. Sie müssen also immer ein neues double-Array anlegen, in das Sie die verarbeiteten Werte speichern, und dann ein neues `Data`-Objekt mit diesem neuen Array erzeugen.

c) Test

Führen Sie die `main`-Methode der Klasse `Main` aus. Am Ende sollte in etwa diese Ausgabe herauskommen:

```
processing data with 'Scaler(min=0.0,100.0)'
  before: 2.0,4.0,4.0,4.0,5.0,5.0,7.0,9.0
   after: 0.0,28.57142857142857,28.57142857142857,28.57142857142857,42.857142857142854,42.857142857142854,71.42857142857143,100.0
processing data with 'Scaler(min=-20.0,123.0)'
  before: 0.0,28.57142857142857,28.57142857142857,28.57142857142857,42.857142857142854,42.857142857142854,71.42857142857143,100.0
   after: -20.0,20.857142857142854,20.857142857142854,20.857142857142854,41.285714285714285,41.285714285714285,82.14285714285714,123.0
processing data with 'Standardizer'
  before: -20.0,20.857142857142854,20.857142857142854,20.857142857142854,41.285714285714285,41.285714285714285,82.14285714285714,123.0
   after: -1.4999999999999998,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0
processing data with 'Clipper(lower=-1.0)'
  before: -1.4999999999999998,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0
   after: -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0
processing data with 'Clipper(upper=1.1)'
  before: -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0
   after: -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,1.1
processing data with 'Clipper(lower=-0.7,upper=1.0)'
  before: -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,1.1
   after: -0.7,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,1.0
```

Geben Sie Ihre Ausgabe im PDF mit ab.

Download:

In den Downloadmaterialien finden Sie das zur Verfügung gestellte Klassenkonstrukt, sowie die zur Ausführung der `main`-Methode nötige Testdatei `data.csv`.