ulm university universität ulm

Programmierung von Systemen

Institut für Softwaretechnik und Programmiersprachen | Sommersemester 2019
Prof. Dr. Matthias Tichy, Stefan Götz

Übungsblatt 2: Interfaces, Datenstrukturen und Java API

Abgabetermin: 12.5.2017, 23.59 Uhr

Aufgabe 1: Paare und Interfaces

(0 + 1 + 2 + 2 + 2 + 4)

Die Java-Community hat eine lange Diskussion darüber geführt ob Tupel/Paare ein teil der Java Standardbibliothek sein sollen. Am Ende wurde sich aus verschiedensten Gründen dagegen entschieden. Den Entscheidenden zum trotz werden wird in dieser Aufgabe eine Tupel-Klasse erstellt werden.

Die zugehörigen Felder und Methoden kannst du der Abbildung 1 entnehmen. Da Tupel ihre Inspiration aus der *funktionalen*-Programmierung ziehen soll eine weitere wichtige Eigenschaft gelten: Tupel sollen **`immutable`** sein.

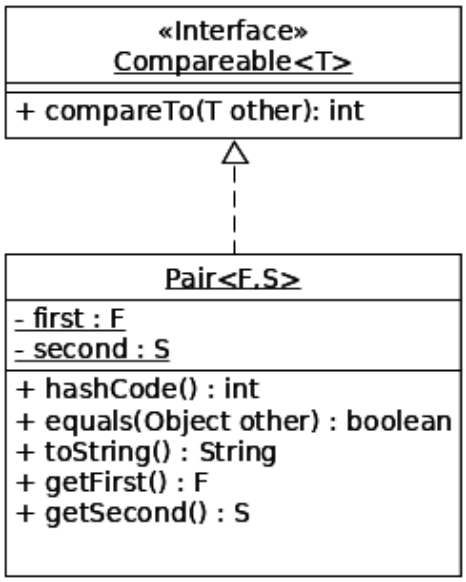

Abbildung 1: Klassendiagramm für Pair

1. Informiere dich darüber was du bei der Erstellung einer Klasse beachten musst um sie **`immutable`** zu machen.
2. Aus welchen Gründen ist es sinnvoll immutable Klassen statt "normalen"Klassen zu verwenden?
3. Aus welchen Gründen ist es schwer zu garantieren, dass eine generische Pair-Klasse (`Pair<F,S>`) komplett immutable ist?
4. Implementiere die Klasse `Pair` mit **allen Feldern**, dem **Konstruktor** und der **`toString`** Methode.
5. Verwende deine IDE um automatisch `Getter`, aber keine `Setter` (siehe `immutable`), für alle Felder der Klasse zu erzeugen.

   Verwende deine IDE um automatisch eine `equals` und `hashCode` Methode zu erzeugen.

   Achte darauf welche Einstellungen jeweils gemacht werden können, zum Beispiel für die Formatierung oder die verwendeten Felder.
6. Informiere dich auf https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html über das `Comparable` Interface und implementiere es in der `Pair` Klasse. Dabei soll das `first` Feld stärker gewichtet sein als das `second` Feld.

Aufgabe 2: `Collections` und Datenstrukturen (2 + 1 + 2)

Die Klasse `java.util.Collections` (https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html) bietet viele Methoden an, die das Arbeiten mit Listen und im Allgemeinen mit Klassen, die das Interface `Collection` implementieren, erleichtern.

1. Finde heraus, welche Möglichkeiten es gibt, eine Liste zu sortieren und warum es nicht möglich ist, eine `Collection` (Achtung: kein s am Ende!) zu sortieren.
2. Du hast bereits das `Comparable<T>`-Interface in der Klasse `Pair` implementiert. Erzeuge eine Datenstruktur, die du mit `Collections.sort` sortieren kannst und fülle sie mit mindestens sieben Instanzen deiner `Pair`-Klasse, dabei soll mindestens ein Paar doppelt vorkommen. Sortiere die Datenstruktur und gib sie aus.

   Teste die Korrektheit deiner Implementierung des `Comparable`-Interfaces, indem du weitere Testdaten in die Liste einfügst.
3. Sieh dir die Klasse `HashSet` (https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html) an. Verwende eine Schleife um alle Instanzen deiner Klasse `Pair` aus der vorherigen Teilaufgabe in ein `HashSet` einzufügen. Suche während dem Einfügen nach doppelten Daten und gib sie aus.

   Die Verwendung von `HashSets` ist ein Standard Verfahren um Duplikate zu finden. Hier ist es nützlich, dass du in der vorigen Aufgabe `equals` und `hashCode` implementiert hast.

Aufgabe 3: `LocalDate`: Woher bekommt man das heutige Datum? (1 + 1 + 1)

Java bietet eine große Standardbibliothek, die bereits viele nützliche Klassen und Funktionen enthält, nicht aber die Pair Klasse. Dazu zählen unter anderem auch Datenstrukturen zur Darstellung eines Datums und auch Klassen um mit Daten und Zeit umzugehen. Im Zweifelsfall sollte immer eher eine Klasse aus der Standardbibliothek benutzen werden, weil sie die meisten Java Programmierer kennen und sie besser getestet ist als dein eigener Code.

Sieh dir die Dokumentation der Klasse `LocalDate` an (https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html). Sie gehört zum Java Date Time Package, das es seit Java 8 gibt (https://docs.oracle.com/javase/tutorial/datetime/index.html).

Schreibe eine Klasse und implementiere die folgenden Punkte in ihrer `main` Methode.

1. Gib das aktuelle Datum mithilfe von `LocalDate` aus.
2. Gib das Datum vor vier Wochen und vor einem Monat aus. Sind die Daten gleich?
3. Gib den Wochentag der obigen Daten aus.

Aufgabe 4: Sokoban (3 + 4 + 1 + 4 + 1)

Sokoban ist ein Rätselspiel, in dem man Kisten auf bestimmte Zielfelder schiebt. Dadurch, dass man Kisten nur schieben, aber nicht ziehen kann, ist es möglich sich selbst Lösungen zu verbauen. Weitere Infos gibt es zum Beispiel hier: https://en.wikipedia.org/wiki/Sokoban.

In dieser Aufgabe geht es darum Übung in dem Arbeiten mit mehrdimensionalen Arrays in Java zu bekommen. Dazu sollst du die minimal notwendigen Methoden implementieren um Sokoban spielen zu können.

Ein Sokoban Spielfeld soll als zweidimensionales Array vom Typ `char` dargestellt werden. Dabei stellt `#` Wände dar, `@` ist der Spieler, `.` ein leeres Feld und `$` ist eine Kiste. Auf diesem Übungsblatt gibt es noch keine Zielfelder.

Hier ist Beispielcode, der ein 7x7 Sokoban Array erzeugt:

```
char[][] sokoban = new char[7][];
sokoban[0] = "#######".toCharArray();
sokoban[1] = "#.....#".toCharArray();
```

```
sokoban[2] = "#..$..#".toCharArray();
sokoban[3] = "#.$@$.#".toCharArray();
sokoban[4] = "#..$..#".toCharArray();
sokoban[5] = "#.....#".toCharArray();
sokoban[6] = "#######".toCharArray();
```

Implementiere die folgenden Methoden in einer Klasse namens `Sokoban`.

1. Schreibe eine Methode `findPlayer` die ein `char[][]` (zweidimensionales `char`-Array) erhält und ein `Pair<Integer,Integer>` zurückgibt, der die Position des Spielers im Array darstellt.

   Verwende zwei geschachtelte `for`-Schleifen um das Array nach dem `@` zu durchsuchen.

   Beim obigen Beispiel muss `x=3, y=3` herauskommen.

2. Schreibe eine Methode `moveNorth`, die ein `char[][]` erhält und einen `boolean` zurückgibt. Wenn sich der Spieler nach Norden bewegen kann, soll die Methode das Array entsprechend ändern und `true` zurückgeben. Ansonsten soll das Array gleich bleiben und `false` zurückgegeben werden.

   Der Spieler kann sich nur bewegen wenn:

   - das Zielfeld leer ist (`.`).
   - das Zielfeld eine Kiste (`$`) ist und das Feld hinter der Kiste leer (`.`) ist. In diesem Fall schiebt der Spieler die Kiste vor sich her. Der Spieler kann also nur eine Kiste auf einmal verschieben.

   Verwende die `findPlayer` Methoden von oben um den Spieler zu finden. Untersuche danach die angrenzenden Felder in der entsprechenden Himmelsrichtung.

3. Schreibe eine Methode `sokobanToString`, die ein Sokoban-Array in einen String umwandelt.

   Hier eine Beispielausgabe für das obige Array vor und nach einem Aufruf von `moveNorth`.

```
Eingabe: #######   moveNorth: #######
         #.....#              #..$..#
         #..$..#              #..@..#
         #.$@$.#              #.$.$.#
         #..$..#              #..$..#
         #.....#              #.....#
         #######              #######
```

4. Implementiere die restlichen Bewegungsmethoden `moveEast`, `moveSouth` und `moveWest`.

   Hinweis: Dir wird auffallen, dass diese Methoden sehr repetitiv sind. Es ist möglich, die Bewegungsrichtung als zusätzlichen Parameter zu übergeben und sich auf diese Weise viel Code zu sparen.

5. Teste deine Sokoban Implementierung indem du abwechselnd Bewegungsmethoden auf deinem Array aufrufst und das Spielfeld ausgibst. Zum Beispiel so:

```
System.out.println(sokobanToString(sokoban));
moveNorth(sokoban);
System.out.println(sokobanToString(sokoban));
moveEast(sokoban);
System.out.println(sokobanToString(sokoban));
moveSouth(sokoban);
System.out.println(sokobanToString(sokoban));
```

Abgabe

Beachte bei der Abgabe folgendes:

- Die Bearbeitung kann in Gruppen erfolgen, muss jedoch nicht. Wichtig ist, dass, sollte die Abgabe in einer Gruppe erfolgen die Namen der Mitglieder erkenntlich sein sollten.
- Die Abgabe erfolgt über Moodle.
- Der abgegebene Code muss kompilieren.
- Alle von Ihnen erzeugten Klassen und Methoden müssen ausreichend mit JavaDoc und normalen Kommentaren versehen sein.
- Alle Fragen müssen hierbei sinnvoll beantwortet sein.

Fragen

Bei Fragen können das Moodle-Forum oder die Tutoren zu rate gezogen werden.