Übung 06: Collections

Abgabetermin: 28. 4. 2016, 8:15

Name: _____ **Matrikelnummer**: _____

Informatik: ☐ G1 (Grimmer) ☐ G2 (Prähofer) ☐ G3 (Prähofer)

WIN: ☐ G1 (Khalil) ☐ G2 (Khalil) ☐ G3 (Hummel)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 6	24	Java-Programm, Ausgabe der Testläufe	Java-Programm	☐	

Verwandtschafts-Datenbank (24 Points)

a) Relatives (18 Punkte)

Erstellen Sie eine Klasse zur Verwaltung von Verwandtschaftsbeziehungen (`Relatives.java`). Diese Relatives-Klasse soll eine Gruppe von Personen (`Person.java`) verwalten können und verschiedene Abfragen darauf zur Verfügung stellen. Sie können davon ausgehen, dass die Namen der Personen in den Eingabedaten immer eindeutig sind.

Die `Person`-Klasse sollen dabei folgende Informationen beinhalten:

- Vorname (`String`)
- Geschlecht (`enum Gender`)
- Vater (`Person`)
- Mutter (`Person`)
- PartnerIn (`Person`)

Implementieren Sie `equals` und `hashCode` auf Basis des eindeutigten Namens. Des Weiteren soll `Person Comparable<Person>` implementieren und nach dem Namen vergleichen.

Die `Relatives`-Klasse soll folgende Abfragen erlauben:

- `getPersons()` liefert alle Personen in der Datenbank als `SortedSet`, wobei die Elemente nach der natürlichen Ordnung sortiert werden.
- `getPersonMap()` liefert eine (nicht veränderbare) Abbildung von Name (`String`) auf Person.
- `getAncestors(Person person)` liefert alle Vorfahren der angegebenen Person
- `getDescendants(Person person)` liefert alle Nachkommen der angegebenen Person
- `getNumberOfAncestors()` liefert eine Abbildung von Person auf Anzahl Vorfahren (`Integer`)
- `getNumberOfDescendants()` liefert eine Abbildung von Person auf Anzahl Nachkommen (`Integer`)
- `getCommonAncestors(Collection<Person> pers)` liefert für mehrere Personen die gemeinsamen Vorfahren
- `getCommonDescendants(Collection<Person> pers)` liefert für mehrere Personen die gemeinsamen Nachkommen
- `getAncestorComparator()` liefert einen `Comparator<Person>`, der zuerst nach Anzahl Vorfahren (absteigend) und dann nach Namen (aufsteigend) sortiert
- `getDescendantComparator()` liefert einen `Comparator<Person>`, der zuerst nach der Anzahl Nachkommen (absteigend) und dann nach Namen (aufsteigend) sortiert

Vorfahren sind hierbei die Eltern + die Eltern der Eltern + …

Nachfahren sind die Kinder + die Kinder der Kinder + …

Ziel dieser Aufgabe ist es, die Funktionen der Collections – Bibliothek zu verwenden. Sie sollten also versuchen, die Aufgabenstellung soweit möglich unter Verwendung der Collections zu lösen. Gerne dürfen Sie auch die Stream API verwenden.

Achten Sie beim Erstellen der Schnittstelle der Klasse darauf, möglichst allgemeine Typen zu verwenden (z.B. `Map` und nicht `HashMap`) und die in der richtigen Form zurückzugeben (z.B. immutable).

Eine Klasse `PersonData` ist vorgegeben. In ihr befindet sich ein Feld namens `DATA`, das Beispieldaten enthält. Die Daten sind in einem Array von Strings gespeichert, wobei jeder String eine Person beschreibt. Der String enthält, jeweils durch Leerzeichen getrennt:

- Vorname
- Geschlecht („F“ oder „M“)
- Vorname des Vaters (oder „-“ falls nicht in der Datenbank)
- Vorname der Mutter (oder „-“ falls nicht in der Datenbank)
- Vorname der Partnerin / des Partners (oder „-“ falls nicht in der Datenbank/nicht vorhanden)

```
package at.jku.ssw.swe.relatives;

public class PersonData {
   public static final String[] DATA = new String[] {
      "Adelia F Demarcus Allena -",
      "Allena F Jae Joni Demarcus",
      "Aron M Demarcus Allena -",
      "Bernadette F Demarcus Allena -",
      "Darius M - - Elena",
      "Daron M Darius Elena Tamekia",
      "Demarcus M Darius Elena Allena",
      "Elena F - - Darius",
      "Jae M - - Joni",
      "Jesenia F Jae Joni -",
      "Joni F - - Jae",
      "Karl M Darius Elena -",
      "Kathlene F Monty Pauletta -",
      "Kraig M Daron Tamekia -",
      "Lelah F Daron Tamekia -",
      "Monty M - - Pauletta",
      "Pauletta F - - Monty",
      "Tamekia F Monty Pauletta Daron",
      "Vernell F Daron Tamekia -"
   };
}
```

Der Konstruktor der `Relatives`-Klasse soll ein derartiges Array von Strings entgegennehmen, auswerten und die Datenbank mit diesen Daten initialisieren. Sie können davon ausgehen, dass die Eingabedaten korrekt sind, also keine doppelten Namen enthalten und alle Väter/Mütter/PartnerInnen in der Datenbank existieren.

b) Beispielabfragen (6 Punkte)

Erstellen Sie ein Programm, das die `Relatives`-Klasse benutzt um folgende Abfragen auf die Beispieldaten auszuführen (es ist nicht notwendig eine interaktive Konsolenanwendung zu programmieren):

- Was sind die Nachfahren von Pauletta? Was sind die Nachfahren von Darius? Was sind die Nachfahren von Joni?
- Was sind die gemeinsamen Nachfahren von Darius und Joni? Was sind die gemeinsamen Vorfahren von Adelia und Lelah?
- Eine nach Anzahl Nachfahren und Name sortierte Liste aller Personen.
- Wer hat genau zwei Vorfahren?
- Wie groß ist die Familie (Vorfahren + Nachfahren) von Allena?
- Eine nach Anzahl Vorfahren und Name sortierte Liste aller Frauen, die keine Kinder in der Datenbank haben.

Ihr Programm soll diese Fragen und die Antworten textuell auf der Konsole ausgeben.

```
descendents of Pauletta: [Lelah, Vernell, Kathlene, Kraig, Tamekia]
descendents of Darius: [Daron, Lelah, ...
...
```

Schließen sie die Konsolenausgabe in Ihre Übungsabgabe mit ein.