Datenstrukturen PVL 5- SS2023

Professur Softwaretechnik

6| 2023

Administratives

Abgabe Termin

19.06.2023 - 23:59

Abgabe

Ihre Abgabe darf entweder in Java oder Python geschehen. Nutzen Sie für die Abgabe das Code Template aus dem OPAL-Bereich "Prüfungsvorleistung 5". Nichtbeachtung des Templates kann zu Punktabzügen führen! Sie dürfen zusätzliche Klassen und Methoden anlegen, um die Aufgabe zu lösen.

Laden Sie alle .java oder .py Dateien, die zum Ausführen Ihres Codes nötig sind, im OPAL-Bereich "Prüfungsvorleistung 5" hoch. Bitte geben Sie keine ganzen Ordner ab, die Quelldateien reichen. Hinweis für Java: Von uns vorgegebene Interfaces brauchen Sie ebenfalls nicht mit abzugeben.

Erlaubte Klassen

Sie dürfen Klassen und Methoden aus der Standardbibliothek von Java bzw. Python nutzen. Bitte benutzen Sie keine Bibliotheken von Dritten.

Information zu Plagiaten

Es ist **erlaubt**, Lösungen aus dem Internet in Ihre Abgabe zu integrieren. Bitte **markieren** Sie jeglichen Code, der nicht von Ihnen selbst stammt, indem Sie einen Kommentar mit der Quelle des Codes (URL ist ausreichend) an die entsprechende Stelle setzen. Dies soll verhindern, dass Ihre Lösung fälschlicherweise als Plagiat erkannt wird.
Da es sich hierbei um eine Prüfungsvorleistung handelt, ist es notwendig, dass Sie die Aufgaben selbständig bearbeiten. Daher werden alle Abgaben untereinander auf Plagiate überprüft. Sollten Plagiate erkannt werden, gilt die Abgabe von **sämtlichen** beteiligten Parteien als ungültig.

Kompilierbarkeit und Clean Code

Es werden nur Abgaben gewertet, welche sich in einem ausführbaren Zustand befinden!
Achten Sie außerdem auf die Leserlichkeit des Codes. Unverständliche oder unangemessene Bezeichner von Klassen, Variablen, etc. können zu Punkteabzügen führen.
Beachten Sie, dass Methodennamen in Java in camelCase und in Python in snake_case geschrieben werden, auch wenn das Aufgabenblatt nur die Java-Variante zeigt.

Fragen

Da es sich um eine Prüfungsvorleistung handelt, ist es wichtig, dass jeder Teilnehmer Zugang zu allen notwendigen Informationen hat. Deswegen bitten wir Sie, Fragen direkt ins Forum, im Thread "PVL 5: Fragen" zu stellen. Bitte beachten Sie, dass Sie ihre Fragen möglichst frühzeitig stellen. Kurz vor der Deadline können wir keine rechtzeitigen Antworten mehr garantieren.

Aufgabe

Bäume sind besonders gut dazu geeignet, Daten mit klaren hierarchischen Strukturen darzustellen, wie zum Beispiel den Stammbaum einer Familie. Für diese Aufgabe sollen Sie einen simplen Stammbaum implementieren, welcher nur eine Seite einer Familiengeschichte darstellt, Partner werden zu diesem Zweck ignoriert.

Nutzen Sie als Knoten für den Baum die Klasse **Person**. Implementieren Sie die folgenden Methoden:

1 `Konstruktor(String name)`

Implementieren Sie einen Konstruktor in ihrer entsprechenden Sprache, welcher den Namen der Person entsprechend des übergebenen Werts setzt. Eine Person sollte außerdem eine Sammlung von weiteren Personen besitzen (um ihre Kinder darstellen zu können).

1 `getName()`

Geben Sie den Namen der Person zurück.

1 `getChildren()`

Geben Sie die Kinder der Person als Liste von Personen zurück.

Implementieren Sie die Klasse **FamilyTree** als Baum-Struktur. Alle Personen, die Teil des Baums sind, werden mit einer eindeutigen ID gekennzeichnet. Implementieren Sie die folgenden Methoden:

1 `Konstruktor(Person root)`

Implementieren Sie einen Konstruktor in Ihrer entsprechenden Sprache, welcher die übergebene Person als Wurzel des Baums setzt. Die Wurzel erhält die ID 0.

1 `insertAsChildOf(Person toInsert, Integer parentID)`

Fügen Sie die Person *toInsert* als Kind von der Person mit der ID *parentID* in den Baum ein. Teilen Sie ihr dabei eine eindeutige ID zu. IDs werden nach der Reihenfolge, nach der die Personen dem Stammbaum hinzugefügt werden, vergeben (root=0, nächste eingefügte Person =1, usw.). Geben Sie die ID zurück.

1 `getRoot()`

Geben Sie die ID der Wurzel zurück.

1 `getParentOf(Integer id)`

Geben Sie die ID des Elternteils der übergebenen *id* zurück.

1 `getSiblingsOf(Integer id)`

Geben Sie eine Liste von IDs der Geschwister von *id* zurück.

1 `getChildrenOf(Integer id)`

Geben Sie eine Liste von IDs der Kinder von *id* zurück.

1 `getNiecesNephewsOf(Integer id)`

Geben Sie eine Liste von IDs der Nichten und Neffen von *id* zurück.

1 `getCousinsOf(Integer id)`

Geben Sie eine Liste von IDs der Cousinen und Cousins von *id* zurück.

1 `getPersonById(Integer id)`

Geben Sie die Person mit der übergebenen *id* zurück.

Beispiel

```
Person grandpa = new Person("Johann");

FamilyTree buddenbrooksTree = new FamilyTree(grandpa);
getRoot(); //gibt 0 zurück

insertAsChildOf(new Person("Gotthold"), 0); //gibt 1 zurück
insertAsChildOf(new Person("Jean"), 0); //gibt 2 zurück
insertAsChildOf(new Person("Olly"), 0); //gibt 3 zurück

insertAsChildOf(new Person("Friederike"), 1); //gibt 4 zurück
insertAsChildOf(new Person("Henriette"), 1); //gibt 5 zurück
insertAsChildOf(new Person("Pfiffi"), 1); //gibt 6 zurück

insertAsChildOf(new Person("Thomas"), 2); //gibt 7 zurück
insertAsChildOf(new Person("Antonie"), 2); //gibt 8 zurück
insertAsChildOf(new Person("Christian"), 2); //gibt 9 zurück
insertAsChildOf(new Person("Clara"), 2); //gibt 10 zurück

getParentOf(3); //gibt 0 zurück
getSiblingsOf(3); // gibt [1,2] zurück
getChildrenOf(3); //gibt eine leere Liste zurück
getNiecesNephewsOf(3); // gibt [4,5,6,7,8,9,10] zurück
```

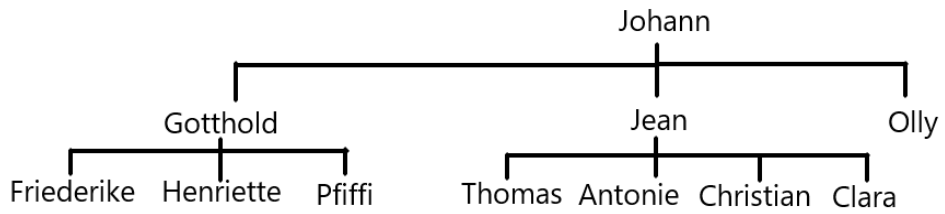

Figure 1: Visualisierung des entstandenen Stammbaums