Datenstrukturen PVL 3- SS2023

Professur Softwaretechnik

5| 2023

Administratives

Abgabe Termin

22.05.2023 - 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 3". 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 3" hoch. Hinweis für Java: Von uns vorgegebene Interfaces brauchen Sie 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.

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, in den Thread "PVL 3: Fragen" zu stellen.

Aufgabe

Erstellen Sie eine Double Linked List, die als Bank verwendet werden kann. Die Knoten der Liste sollen durch die Klasse **Konto** repräsentiert werden. Jeder Knoten in der Liste soll neben dem Vorgänger- und Nachfolgerknoten auch die Daten eines Kontos enthalten, einschließlich des Names und des aktuellen Kontostands.

Wichtig: Sie sollen hier eine eigene Datenstruktur manuell implementieren und *nicht* die bereits vorhandenen Listen der Standardbibliothek verwenden. Abgaben, welche einfach nur die bereits vorhandenen Listen 'neu verpacken' werden nicht gewertet.

Implementieren Sie für die Klasse **Konto** folgendes:

```
constructor(String name, int balance)
```

Welche dem Konstruktor Ihrer genutzten Sprache entspricht und die entsprechenden Attribute des Knotens auf 'name' und 'balance' setzt. Der Vorgänger und Nachfolger des Knotens sollen dabei mit 'null' bzw. mit 'None' initialisiert werden.

```
getName()
```

Welche den Namen des Kontoinhabers zurückgibt.

```
getBalance()
```

Welche den Kontostand zurückgibt.

```
getPrev()
```

Welche das vorherige Konto zurückgibt.

```
getNext()
```

Welche das nachfolgende Konto zurückgibt.

```
setPrev(Konto konto)
```

Welche das übergebene Konto als Vorgänger setzt. Hatte das ursprüngliche Konto bereits Vorgänger, sollen diese als Vorgänger des übergebenen Kontos gesetzt werden.

```
setNext(Konto konto)
```

Welche das übergebene Konto als Nachfolger setzt. Hatte das ursprüngliche Konto bereits Nachfolger, sollen diese als Nachfolger des übergebenen Kontos gesetzt werden.

Implementieren Sie für die Klasse **Bank** folgendes:

`constructor()`

Welche dem Konstruktor Ihrer genutzten Sprache entspricht und eine leere Liste erzeugt.

`getHead()`

Gibt den Kopf der Liste (das erste Element) zurück.

`append(String name, int balance)`

Fügt ein neues Konto mit den übergebenen Werten an der letzten Stelle der Bank hinzu.

`get(index)`

Welche das Konto am übergebenen Index zurückgibt. Ist der Index nicht in der Liste vorhanden, geben Sie null bzw. None zurück.

`insert(int index, String name, int balance)`

Welche ein neues Konto mit den übergebenen Werten erzeugt und dieses an dem gegebenen Index hinzufügt. Sollte der übergebene Index der aktuellen Größe der Liste entsprechen, wird das neue Konto am Ende der Liste hinzugefügt.
Falls das Hinzufügen erfolgreich ist, geben Sie den Wahrheitswert *True* zurück, ansonsten *False*.

`delete(int index)`

Welche das Konto an der Stelle des Indexes löscht. Konnte das Konto erfolgreich gelöscht werden, geben Sie den Wahrheitswert *True* zurück, ansonsten *False*.

`swap(int index1, int index2)`

Welche zwei Konten an den übergebenen Indizes in der Bank tauscht. Falls der Tausch erfolgreich ist, geben Sie den Wahrheitswert *True* zurück, ansonsten *False*.

`sortByBalance()`

Welche die Konten anhand ihres Kontostandes aufsteigend sortiert. Haben zwei Konten den gleichen Kontostand, ist die Reihenfolge egal.

`getMedian()`

Welche den Median der Kontostände bestimmt. Der Median ist der Wert, der genau in der Mitte einer sortierten Datenreihe liegt. Wenn die Anzahl der Konten ungerade ist, ist der Median also die 'balance' des Kontos in der Mitte der Liste. Wenn die Anzahl der Konten gerade ist, gibt es keinen eindeutigen mittleren Wert, und der Median ist der Durchschnitt der beiden mittleren Werte.

`printAccounts()`

Welche alle Konten der Bank auf der Konsole ausgibt. Achten Sie auf eine leserliche Formatierung der Kontodaten.

Beispiel

```
Bank bank = new Bank();

bank.append("Max", 100);
bank.append("Lilian", 150);
bank.append("Ulla", 200);

bank.insert(2, "Hans", 5000); \\ gibt 'true' zurück
bank.insert(10, "Alice", 400); \\ gibt 'false' zurück

bank.get(0); \\ gibt das Konto von Max zurück
bank.get(8); \\ gibt 'null' zurück

bank.delete(3); \\ löscht das Konto von Ulla, gibt 'true' zurück
bank.delete(5); \\ gibt 'false' zurück

bank.getMedian(); \\ gibt 150 zurück
```