Ostfalia - Hochschule für angewandte Wissenschaften
Institut für Software Engineering
Prof. Dr. F. Seutter

WS 2018/19

Laboraufgaben Algorithmen und Datenstrukturen

Labortermine und späteste Abgabetermine:

Aufgabe	Labor	Abgabe
	02.10.18	
	16.10.18	
1+2	30.10.18	30.10.18
	13.11.18	
3	27.11.18	27.11.18
4	11.12.18	11.12.18

Allgemeines:

- Sie arbeiten in Zweier-Gruppen.
- Jede Laborabgabe muss spätestens an dem angegebenen Termin erfolgen. Bei jeder Aufgabe checken Sie Ihre Lösung im SVN ein und stellen den Laborbetreuern die Lösung der Aufgabe vor, welche beide Mitglieder der Gruppe erläutern können.
- Pfad für das SVN: `code.ostfalia.de/svn/i-audws2018/`
- Sie melden sich für die Gruppeneinteilung unter folgender URL bis zum 09.10.2018 elektronisch an: `https://Laborga-i.ostfalia.de`
- Für jede Aufgabe erstellen Sie ein neues Projekt in Eclipse. Die Projektnamen lauten entsprechend `AlgoAufgabe1`, `AlgoAufgabe2`, `AlgoAufgabe3` und `AlgoAufgabe4`.
- Die zu implementierenden Interfaces und die öffentlichen JUnit-Tests können unter

  ```
  L:\fbi\gruendel\Algorithmik\java\Serie1
  L:\fbi\gruendel\Algorithmik\java\Serie2
  L:\fbi\gruendel\Algorithmik\java\Serie3
  L:\fbi\gruendel\Algorithmik\java\Serie4
  ```

  heruntergeladen werden. Die nicht-öffentlichen JUnit-Tests laufen auf unserem Sever und überprüfen Ihre ins SVN hochgeladenen Lösungen.
- Alle JUnit-Tests (öffentlich und nicht-öffentlich) müssen mit Ihren Implementierungen ohne Fehler laufen. Ein Teil der Aufgabenstellungen ist dann damit gelöst. Die restlichen Teile der Aufgabenstellungen sind mit der Präsentation Ihrer Implementierungen und der jeweils zugehörigen Dokumentation zu lösen.
- Ein Beispielprogramm für das Einlesen von Datensätzen aus einer Textdatei kann unter

  ```
  L:\fbi\gruendel\Algorithmik\java\
  ```

  heruntergeladen werden.

- Testdateien für die Aufgaben können unter

  `L:\fbi\gruendel\Algorithmik\Materialien\Aufgabe\`

  heruntergeladen werden.

- Verwenden Sie für Ihre Implementierung die folgenden Namen und halten Sie auch die vorgegebene Verzeichnisstruktur ein. $X \in \{1, 2, 3, 4\}$.

  Projektname: `AlgoAufgabe`X

  Verzeichnisstruktur für das jeweilige Eclipse-Projekt:

```
+---AlgoAufgabeX
|    \---src
|        \---de
|            \---ostfalia
|                \---algo
|                    \---ws18
|                        +---base
|                        |       Gender.java
|                        |       IManagement.java
|                        |       IMember.java
|                        |       KindOfSport.java
|                        |       Member.java
|                        |
|                        +---sX
|                            |   Management.java
|                            |   ....
|                            |
|                            \---test
|                                    ManagementTest.java
|
+--------lib
|            JUnit4AddOn.jar
|
\--------Materialien
             Mitglieder10000.txt
             ...
```

Aufgaben

Es ist eine Mitgliederverwaltung eines Sportvereins zu programmieren. Hierzu sind diverse Datenstrukturen zu definieren und zu implementieren. In den ersten drei Aufgaben sind diese Datenstrukturen selbst zu programmieren, in der vierten Aufgabe sollen dazu die Standardimplementierungen der Java Collections verwendet werden. Die unterschiedlichen Implementierungen sind für bestimmte Operationen auf ihre Effizienz hin zu analysieren und zu vergleichen.

Die für jedes Mitglied zu speichernden Daten sind:

- Schlüssel (Long)
- Name (String)
- Vorname (String)
- Geburtsdatum (jjjj-mm-tt)
- Geschlecht (m/w)
- Sportart (Fussball, Handball, Schwimmen, Leichtathletik, Reiten, Fechten, Turnen, Radsport, Tanzen, Rudern)

Dabei ist der Schlüssel wie folgt aufgebaut:

- Anfangsbuchstabe des Namens, codiert in zwei Dezimalziffern
  a $\mapsto$ 01, b $\mapsto$ 02, ..., z $\mapsto$ 26
  Diese Zuordnung gilt für Groß- und Kleinbuchstaben, Umlaute werden als ae, oe, ue dargestellt.
- Anfangsbuchstabe des Vornamens, codiert in zwei Dezimalziffern, siehe Name
- Geburtsdatum, ttmmjjjj

Sie können davon ausgehen, dass doppelte Schlüssel nicht vorkommen.

Beispieldatensatz:
`130205082001, Mueller, Bernd, 2001-08-05, m, Fussball`

Die Daten sind zunächst aus einer Textdatei in eine initial leere Datenstruktur einzulesen. Über den Schlüssel oder über den Vornamen und den Nachnamen ist ein Zugriff auf den jeweiligen Datensatz zu ermöglichen. Achtung: Vor- und Nachname sind ggf. nicht eindeutig. Anfragen nach Anzahl aller Mitglieder, aller Mitglieder einer Sportart und deren Auflistung in einem Feld (array) sind ebenfalls zu ermöglichen. Die Bedienerführung der Anwendung erfolgt über die Konsole.

Die unterschiedlichen Implementierungen sind bzgl. ihren Zeit-Komplexitäten und Laufzeiten miteinander zu vergleichen. Dies hat durch Zählen der jeweiligen grundlegenden Operationen und der realen Laufzeiten (mit Hilfe der Methode `System.nanoTime()`) zu erfolgen. Die Ergebnisse sind tabellarisch darzustellen und auch in Beziehung zu den jeweiligen theoretisch hergeleiteten Ergebnissen aus der Vorlesung zu setzen.

Um aussagekräftige und vergleichbare Werte zu bekommen werden zwei Test-Datenbestände als Textdateien zur Verfügung gestellt.

- Mitglieder-Datensätze zum Einfügen in die Datenstrukturen, wie oben beschrieben.
  Die jeweiligen Schlüssel sind entsprechend zu generieren.
  Anzahl 10.000, darunter keine Duplikate.
- Schlüssel-Datensätze zum Suchen in den Datenstrukturen.
  Anzahl 10.000, davon 5.000, die nicht enthalten sind.

Dokumentieren Sie alle von Ihnen implementierten Klassen, einschließlich aller Attribute und Methoden, in `JavaDoc`. Überschreiben Sie für diese Klassen die Methode `toString()`.

1. **Unsortierte Verkettete Listen**

   - Für die Mitglieder-Datensätze ist die Klasse `Member` im Package `de.ostfalia.algo.ws18.base` zu erstellen und das vorgegebene Interface `IMember` zu implementieren. Der Konstruktor erhält Name, Vorname, Geburtsdatum, Geschlecht und Sportart in einem String, durch Kommas getrennt, als Argumente (so wie aus der Textdatei eingelesen). Der Schlüssel ist daraus zu generieren.
   - Für die Mitgliederverwaltung ist die Klasse `Management` im Package `de.ostfalia.algo.ws18.s1` zu erstellen und das vorgegebene Interface `IManagement` zu implementieren. Der Konstruktor bekommt den Dateinamen der Textdatei mit den Mitglieder-Datensätzen als Argument und erstellt eine Datenstruktur, in die die Datensätze einzulesen sind.
   - Als Datenstruktur für die Mitgliederverwaltung wählen Sie in dieser Aufgabe eine unsortierte verkettete Liste und fügen Sie die Datensätze ein. Eingefügt wird am Kopf der Liste. Auch das Suchen erfolgt ausgehend vom Kopf der Liste.
   - Die Klassen aus dem `java.util.collection` Framework dürfen hier nicht verwendet werden.
   - Bestimmen Sie die Zeitkomplexität (in Anzahl grundlegender Operationen) und die Laufzeit (in Millisekunden) jeweils für das initiale Einfügen der Datensätze in diese Datenstruktur, das Suchen nach bestimmten Datensätzen anhand der bereitgestellten Testdatensätze und für die Bestimmung der Anzahl der Mitglieder der Sportart Fussball durch Traversierung des gesamten Datenbestandes. Benennen Sie dazu auch die jeweils relevanten grundlegenden Operationen und dokumentieren Sie die Ergebnisse entsprechend der im Anhang aufgeführten Tabelle. Das Lesen der Testdatensätze aus den Textdateien ist von der Zeitmessung auszunehmen.

2. **Sortierte Verkettete Listen**

   - Verwenden Sie die in Aufgabe 1 erstellten Klassen soweit möglich weiter.
   - Als Datenstruktur für die Mitgliederverwaltung wählen Sie in dieser Aufgabe nun die Datenstruktur einer sortierten verketteten Liste (aufsteigend nach Schlüsseln) und fügen Sie die Datensätze sortiert ein.
   - Die Klasse `Management` für die Mitgliederverwaltung ist im Package `de.ostfalia.algo.ws18.s2` zu erstellen und das Interface `IManagement` zu implementieren.
   - Die Klassen aus dem `java.util.collection` Framework dürfen hier nicht verwendet werden.
   - Die Zeitkomplexitäten und die Laufzeiten sind entsprechend Aufgabe 1 zu bestimmen und entsprechend tabellarisch für diese Datenstruktur zu dokumentieren.

3. **Binäre Bäume**

   - Verwenden Sie die in Aufgabe 1 erstellten Klassen soweit möglich weiter.
   - Als Datenstruktur für die Mitgliederverwaltung wählen Sie in dieser Aufgabe nun die Datenstruktur eines binären Suchbaums und fügen Sie die Datensätze ein. Ordnungskriterium sind wiederum die Schlüssel.
   - Die Klasse `Management` für die Mitgliederverwaltung ist im Package `de.ostfalia.algo.ws18.s3` zu erstellen und das Interface `IManagement` zu implementieren.
   - Die Klassen aus dem `java.util.collection` Framework dürfen hier nicht verwendet werden.
   - Die Zeitkomplexitäten und die Laufzeiten sind entsprechend Aufgabe 1 zu bestimmen und entsprechend tabellarisch für diese Datenstruktur zu dokumentieren.
   - Implementieren Sie auch die Methode `height()` aus dem Interface `IManagement`. Sie liefert die Höhe des Binärbaums.

4. **Dynamische Datenstrukturen und das Java Collections Framework**

- Modifizieren Sie Aufgabe 1 dahingehend, dass Sie für die Datenstruktur der Mitgliederverwaltung, Klasse `ManagementList`, statt der selbst programmierten Klasse die Klasse `LinkedList` aus den Java Collections verwenden.
- Modifizieren Sie Aufgabe 3 dahingehend, dass Sie für die Datenstruktur der Mitgliederverwaltung, Klasse `ManagementTree`, statt der selbst programmierten Klasse die Klasse `TreeSet` aus den Java Collections verwenden.
- Erstellen Sie die Mitgliederverwaltung, Klasse `ManagementMap`, als eine Hashtabelle und verwenden Sie dafür die Klasse `HashMap` aus den Java Collections. Der Hashwert für einen Datensatz ist aus seinem Schlüssel zu berechnen.
- Die Klassen `ManagementList`, `ManagementTree` und `ManagementMap` sind im Package `de.ostfalia.algo.ws18.s4` zu erstellen und jeweils das Interface `IManagement` zu implementieren.
- Auf das Benennen und zählen der grundlegenden Operationen werde hier bei der Komplexitätsanalyse verzichtet. Bestimmen Sie analog zu den vorangehenden Aufgaben nur die Laufzeiten und dokumentieren Sie diese entsprechend.

Anhang

Datenstruktur	Anzahl Datensätze	Einfügen			
		grundlegende Operation	Anzahl Operationen	Laufzeit in Millisek.	Komplexität in $O()$
unsortierte verkettete Liste	10.000				
sortierte verkettete Liste	—				
binärer Suchbaum	—				
`LinkedList`	—	—	—		
`TreeSet`	—	—	—		
`HashMap`	—	—	—		

Datenstruktur	Anzahl Datensätze	Suchen			
		grundlegende Operation	Anzahl Operationen	Laufzeit in Millisek.	Komplexität in $O()$
unsortierte verkettete Liste	10.000				
sortierte verkettete Liste	—				
binärer Suchbaum	—				
`LinkedList`	—	—	—		
`TreeSet`	—	—	—		
`HashMap`	—	—	—		

Datenstruktur	Anzahl Datensätze	Traversieren			
		grundlegende Operation	Anzahl Operationen	Laufzeit in Millisek.	Komplexität in $O()$
unsortierte verkettete Liste	10.000				
sortierte verkettete Liste	—				
binärer Suchbaum	—				
`LinkedList`	—	—	—		
`TreeSet`	—	—	—		
`HashMap`	—	—	—		

Bewertung

Voraussetzung: Alle JUnit-Tests müssen ohne Fehlermeldung laufen.

Aufgabe	Funktionalität	Punkte
1)	Einlesen der Textdateien	1
	Klasse Member implementiert	1
	Klasse verkettete Liste implementiert	1
	Member Schlüsselgenerierung und Einfügen in verkettete Liste	1
	Suchen in verkettete Liste	1
	Iterieren und Sportartenausgabe	1
	Anzahl Op. und Zeit jeweils in Größenordnung richtig	1
2)	Einfuegen in sortierte verkettete Liste	1
	Anzahl Op. und Zeit jeweils in Größenordnung richtig	1
3)	Klasse binärer Suchbaum implementiert	1
	Einfügen in binärer Suchbaum	1
	Suchen in binärer Suchbaum	1
	Iterieren und Sportartenausgabe	1
	Anzahl Op. und Zeit jeweils in Größenordnung richtig	1
4)	Klasse LinkedList richtig verwendet	1
	Laufzeit jeweils in Größenordnung richtig	1
	Klasse TreeSet richtig verwendet	1
	Laufzeit jeweils in Größenordnung richtig	1
	Klasse HashMap richtig verwendet	1
	Laufzeit jeweils in Größenordnung richtig	1
	Summe	20