Übung 8: Dynamische Datenstrukturen

Gegeben ist folgende Klasse `Student` mit den Feldern `name` (für den Namen der/des Studierenden), `subject` (für das Studienfach) und `ects` (für die Anzahl an bereits absolvierten ECTS-Punkten):

```
public class Student {
    private final String name;
    private final int ects;

    public Student(String name, int ects) {
        this.name = name;
        this.ects = ects;
    }

    public String getName() {
        return name;
    }

    public int getECTS() {
        return ects;
    }

    @Override
    public String toString() {
        return String.format("Student{%s: %d}", name, ects);
    }

}
```

Im Folgenden sollen Sie eine einfach verkettete, sortierte Liste entwickeln, die Objekte dieser Klasse speichern kann. Alle weiteren Anforderungen sind bei den jeweiligen Aufgaben detailliert beschrieben.

1. Listenknoten

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie die Implementierung von *Listenknoten* üben.

Entwickeln Sie eine Klasse `StudentListNode`, die ein `Student`-Objekt sowie einen Zeiger auf einen weiteren Knoten vom Typ `StudentListNode` speichert. Diese Knotenklasse soll als Basis zur Implementierung einer einfach verketteten, sortierten Liste von `Student`-Objekten dienen. Definieren Sie auch mindestens einen sinnvollen Konstruktor für `StudentListNode`.

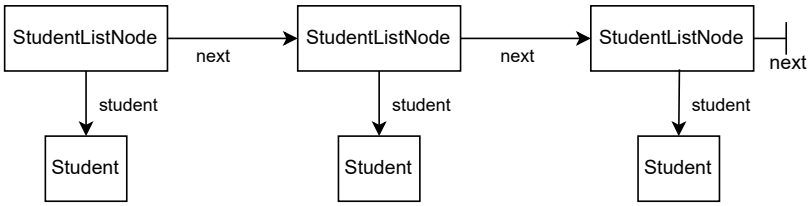

Abbildung 1: Beispiel für drei verkettete `Student`-Objekte mittels `StudentListNode`-Knoten

2. Liste

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie die Felder von *knotenbasierten Listen* kennenlernen.

Definieren Sie eine Klasse `StudentList`, die mit `StudentListNode`-Knoten arbeitet. Die Liste soll ein Feld `head` haben, das auf den ersten Knoten der Liste zeigt. Des Weiteren soll die Anzahl der Elemente der Liste über eine Methode `int getSize()` abgefragt werden können.

- Achten Sie auf eine effiziente Implementierung von `int getSize()`. Durchlaufen Sie beim Methodenaufruf also nicht jedes Mal die Liste, um die Anzahl der Elemente zu bestimmen.

3. Hinzufügen von Studierenden zur Liste

Punkte: 3

Lernziel: In dieser Aufgabe sollen Sie *absteigendes Einfügen in Listen* üben.

Implementieren Sie in der Klasse `StudentList` eine Methode `void add(Student student)`, die ein `Student`-Objekt zur Liste hinzufügt. Dabei soll das Einfügen **sortiert** erfolgen. Für die Sortierung sollen die ECTS-Punkte der Studierenden in **absteigender** Reihenfolge herangezogen werden (der/die Studierende mit den meisten abgeschlossenen ECTS steht also am Beginn der Liste). Bei gleicher ECTS-Anzahl ist es egal, ob das neue `Student`-Objekt vor oder nach der/dem bereits enthaltenen Studierenden eingefügt wird.

- Achten Sie darauf, die Liste nur so weit zu durchlaufen, wie wirklich nötig.
- Achten Sie auf die korrekte Zuweisung an das Feld `head`, falls der neue Knoten als erster der Liste eingefügt wird.
- Achten Sie darauf, dass sich die Anzahl der Listenelemente beim Einfügen ändert.

4. Leeren der Liste

Punkte: 1

Lernziel: In dieser Aufgabe sollen Sie *das komplette Leeren von Listen* üben.

Implementieren Sie in der Klasse `StudentList` eine Methode `void clear()`, die alle Listenelemente löscht (die Liste ist danach leer, das Feld `head` hat also den Wert `null`).

- Achten Sie darauf, dass sich die Anzahl der Listenelemente beim Löschen ändert.

5. Extrahieren von Studierenden mit hoher ECTS-Anzahl

Punkte: 3

Lernziel: In dieser Aufgabe sollen Sie üben, *den Anfang einer Liste* zu extrahieren.

Implementieren Sie in der Klasse `StudentList` eine Methode
`StudentList studentsWithECTSAbove(int minECTS)`, die eine **neue** Liste zurückliefert, in welcher nur jene Studierenden enthalten sind, die mindestens die angegebene Anzahl an ECTS absolviert haben ($\geq$ `minECTS`). Die bestehende Liste bleibt unverändert.

- Achten Sie darauf, die bestehende Liste nur so weit zu durchlaufen, wie nötig (bedenken Sie, dass die Liste absteigend sortiert ist).

6. Entfernen von Studierenden mit hoher ECTS-Anzahl

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie üben, *Teile vom Anfang einer Liste* zu entfernen.

Implementieren Sie in der Klasse `StudentList` eine Methode
`void removeStudentsWithMoreECTSThan(int maxECTS)`, die die bestehende Liste so ändert, dass nur noch jene Studierenden enthalten sind, die höchstens die angegebene Anzahl an ECTS absolviert haben ($\leq$ `maxECTS`).

- Achten Sie darauf, die Liste nur so weit zu durchlaufen, wie nötig (bedenken Sie, dass die Liste absteigend sortiert ist).
- Achten Sie auf die korrekte Zuweisung des Listenkopfes `head`.
- Achten Sie darauf, dass sich die Anzahl der Listenelemente beim Entfernen ändert.

7. Index-basierter Listenzugriff

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie üben, über einen *Index* auf bestimmte Listenelemente zuzugreifen.

Implementieren Sie in der Klasse `StudentList` eine Methode `Student getAtIndex(int index)`, die die/den Studierenden am spezifizierten Index zurückliefert.

Falls der Index ungültig ist (< 0 oder $\geq$ Anzahl der Listenelemente), soll `null` zurückgeliefert werden.

8. Studierender mit höchster und niedrigster ECTS-Anzahl

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie üben, das *erste und letzte Element* einer Liste zurückzugeben.

8.1. Studierender mit höchster ECTS-Anzahl

Implementieren Sie in der Klasse `StudentList` eine Methode `Student studentWithMostECTS()`, die die/den Studierenden mit den meisten absolvierten ECTS zurückliefert. Die bestehende Liste bleibt unverändert.

Falls die Liste kein einziges Element enthält, soll `null` zurückgeliefert werden.

8.2. Studierender mit niedrigster ECTS-Anzahl

Implementieren Sie in der Klasse `StudentList` eine Methode `Student studentWithFewestECTS()`, die die/den Studierenden mit den wenigsten absolvierten ECTS zurückliefert. Die bestehende Liste bleibt unverändert.

Falls die Liste kein einziges Element enthält, soll `null` zurückgeliefert werden.

9. `toString()`

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie üben, Listen in eine *textuelle Repräsentation* umzuwandeln.

Implementieren Sie in der Klasse `StudentList` eine Methode `String toString()`, die eine Stringrepräsentation der Liste zurückliefert. Sie können die bereits bestehende Methode `student.toString()` der Klasse `Student` verwenden, um eine Stringrepräsentation von Studierenden zu erhalten.

Die Listenelemente sollen innerhalb eckiger Klammern mit Beistrichen getrennt werden.

Beispiele:

- Leere Liste: `"[]"`
- Ein Element: `"[Student{Gabe: 180 ECTS}]"`
- Zwei Elemente: `"[Student{Gabe: 180 ECTS}, Student{Harold: 42 ECTS}]"`

Verwenden Sie zum Aufbau des Rückgabestrings einen `StringBuilder`.

10. Statische Fabrikmethode

Punkte: 2

Lernziel: In dieser Aufgabe sollen Sie üben, eine Liste basierend auf einem *Vararg-Parameter* zu erzeugen und zu füllen.

Implementieren Sie in der Klasse `StudentList` eine **statische** Methode
`static StudentList from(Student... students)`. Diese Methode soll eine neue Liste anlegen, in diese mittels wiederholtem `add` alle Studierenden aus `students` einfügen, und die Liste anschließend zurückliefern.

11. Testen der Implementierung

Punkte: 3

Lernziel: In dieser Aufgabe sollen Sie üben, die Methoden von Listen richtig *anzuwenden*.

Implementieren Sie in einer Klasse `StudentListApplication` eine `main`-Methode, in der Sie Ihre Listenimplementierung testen. Führen Sie dabei **alle** Listenmethoden mindestens einmal aus.

Sie können als Beispieldaten folgendes Array an Studierenden verwenden, auf Basis derer Sie eine Liste mittels der Fabrikmethode `StudentList.from` anlegen können:

```
Student[] students = {
    new Student("Gabe", 180),
    new Student("Harold", 42),
    new Student("Maxine", 72),
    new Student("Bob", 3),
    new Student("Lilly", 144),
    new Student("Susi", 144)
};
```

Beispielausgabe:

```
Initialization

List size: 6
List content: [Student{Gabe: 180}, Student{Susi: 144}, Student{Lilly: 144}, Student{Maxine: 72}, Student{Harold: 42}, Student{Bo
Most ECTS: Student{Gabe: 180}
Fewest ECTS: Student{Bob: 3}
Student at index 2: Student{Lilly: 144}
Student at index 5: Student{Bob: 3}
Student at index 1337: null

Filter ECTS >= 50 (new list):
Filtered list - Size: 4
Filtered list - Content: [Student{Gabe: 180}, Student{Lilly: 144}, Student{Susi: 144}, Student{Maxine: 72}]
Filtered list - Most ECTS: Student{Gabe: 180}
Filtered list - Fewest ECTS: Student{Maxine: 72}
Filtered list - Student at index 2: Student{Susi: 144}
Filtered list - Student at index 5: null
Filtered list - Student at index 1337: null
Size and content after clearing filtered list: 0, []

Remove ECTS > 100 (modify list)

List size: 3
List content: [Student{Maxine: 72}, Student{Harold: 42}, Student{Bob: 3}]
Most ECTS: Student{Maxine: 72}
Fewest ECTS: Student{Bob: 3}
Student at index 2: Student{Bob: 3}
Student at index 5: null
Student at index 1337: null

Remove ECTS > 0 (modify list)

List size: 0
List content: []
Most ECTS: null
Fewest ECTS: null
Student at index 2: null
Student at index 5: null
Student at index 1337: null
```

Zusätzliche Anweisungen

- Verwenden Sie zur Implementierung der Ein- und Ausgabe für alle Programme nur die Funktionen der beiden bereitgestellten Klassen `In` und `Out`, wie in der Übung gezeigt. Die entsprechenden Java-Dateien samt HTML-Dokumentation sind in Moodle unter *InOut.zip* zu finden.
- Implementieren Sie formatierte Ausgaben, indem Sie `String.format` in Kombination mit `Out.print` und `Out.println` verwenden, wie in der Übung gezeigt. Verwenden Sie als Referenz den Foliensatz `StringFormat.pdf` im Moodle.
- Sie dürfen, falls nötig, `Math`-Funktionen der Java-Bibliothek nutzen.
- Verwenden Sie für Variablen passende Datentypen Ihrer Wahl.
- Alle Namen (Klassennamen, Variablennamen, Methodennamen, etc.) sind in Englisch zu wählen.
- Formatierung, Namenswahl, etc. fließen in die Bewertung mit ein.
- Vermeiden Sie Codeduplikation und unnötige Operationen (beispielsweise zu viele oder unnötigen Verzweigungen).

Abzugeben

Geben Sie *eine .zip Datei* mit dem Namen kxxxxxxxx_UExx.zip ab (Beispiel: k01234567_UE08.zip). Darin muss enthalten sein:

- Der *Source Code* aller Aufgaben der aktuellen Übung (alle `.java` Dateien, keine `.class` Dateien).
- Zu der Aufgabe ist ein Testprotokoll anzufertigen (`Testprotokoll.txt` oder `Testprotokoll.pdf`). Testen Sie nicht nur einen Fall, sondern mehrere allgemeine Fälle sowie typische Grenzfälle (negative Zahlen, null, etc.).