

## Übungsblatt 4

*Abgabe: Vorlesung am 26.06.2006*

### 1 Implementierung des Comparable-Interfaces

Gegeben sei die folgende Klasse Student:

```
class Student {  
    int matrikelnr;  
    String vorname;  
    String nachname;  
    String studiengang;  
    int fachsemester;  
}
```

Ergänzen Sie die Klasse Student um eine Implementierung des Interfaces Comparable, die eine natürliche Ordnung nach folgenden Regeln realisiert: (4 Punkte)

- Verwende die alphabetische Ordnung der Nachnamen.
- Bei gleichen Nachnamen: Verwende die alphabetische Ordnung der Vornamen.
- Bei gleichen Nachnamen und Vornamen: Verwende die numerische Ordnung der Matrikelnummern.

### 2 Verwendung des Comparable-Interfaces

Im Laufe der Vorlesung haben Sie mit der Klasse Heap bereits eine Implementierung von HeapSort kennen gelernt, die auf einem Feld von Ganzzahlen arbeitete.

Wie muss diese Klasse angepasst werden, um auf einem Feld von Objekten zu arbeiten, die eine durch Implementierung von Comparable definierte natürliche Ordnung haben? Notieren Sie die Implementierung der veränderten Attribute und Methoden. (7 Punkte)

### 3 Adjazenzmatrix-Implementierung

Die folgende Klasse soll einen ungerichteten Graphen als Adjazenzmatrix modellieren:

```
public class Graph {  
  
    private byte[][] matrix;  
  
    public Graph(int maxKnoten)  
        throws IllegalArgumentException {...}  
  
    public void knotenEinfuegen(int nr)  
        throws IllegalArgumentException {...}  
    public boolean knotenExistiert(int nr)  
        throws IllegalArgumentException {...}  
  
    public void kanteEinfuegen(int start, int ende)  
        throws IllegalArgumentException {...}  
    public boolean kanteExistiert(int start, int ende)  
        throws IllegalArgumentException {...}  
  
    public boolean istZusammenhaengend() {...}  
  
}
```

Durch Aufruf des Konstruktors soll eine Adjazenzmatrix initialisiert werden, die maximal `maxKnoten` viele Knoten umfassen kann. Unmittelbar nach der Initialisierung soll der Graph jedoch noch keine Knoten enthalten. In der Matrix soll für nicht existierende Knoten `-1`, für durch eine Kante verbundene Knoten `1` und für nicht verbundene Knoten `0` stehen.

Mit der Methode `knotenEinfügen` sollen Knoten dem Graphen hinzugefügt werden. Die übergebene Nummer soll dem Index des betreffenden Knotens in der Adjazenzmatrix entsprechen. Die Methode `knotenExistiert` soll genau dann `true` zurückgeben, wenn der Knoten existiert, anderenfalls `false`. Die Methoden für die Kanten sollen analog für ungerichtete Kanten zwischen den Knoten mit den angegebenen Indizes funktionieren. Die Methode `istZusammenhaengend` soll genau dann `true` zurückgeben, wenn der Graph zusammenhängend ist, anderenfalls `false`.

Implementieren Sie die Methodenrumpfe des obigen Coderahmens. Prüfen Sie dabei die übergebenen Werte auf Plausibilität – kann eine Operation aufgrund ungültiger Parameter nicht durchgeführt werden, so lassen Sie den Graphen unverändert und werfen Sie eine `IllegalArgumentException`. (9 Punkte)

*Abgaben bitte mit Namen, Matrikelnummer und Übungsgruppennummer versehen!*