technische universität dortmund

Fakultät für Informatik
Lehrstühle 2, 11

Prof. Dr. C. Sohler, A. Krivošija
H. Blom, N. Kriege, H. Timm, B. Zey
http://tiny.cc/dap2p

Sommersemester 2014

# DAP2 Praktikum – Blatt 6

Ausgabe: 08. Mai — Abgabe: 19. Mai–23. Mai

### Hinweis zu den Langaufgaben

Für diese Aufgaben stehen Ihnen ca. zwei Wochen Bearbeitungszeit zur Verfügung. In den Praktika dieser Woche können (und sollen) Sie mit der Bearbeitung beginnen. Den Rest müssen Sie außerhalb des Praktikums erledigen. Neben den Lernräumen können Sie dafür auch die Poolräume (z.B. in einer Freistunde) nutzen, wenn dort gerade keine Veranstaltungen stattfinden. Im Praktikum in zwei Wochen präsentieren Sie Ihrem Tutor dann Ihre Lösungen auf Ihrem Praktikumsrechner im Pool und bekommen Punkte dafür (nach Absprache mit dem Tutor ist es auch möglich, die Lösung auf Ihrem eigenen Rechner zu präsentieren).
**Wichtig:** Sie müssen die Aufgaben **vor** Beginn des Praktikums (in dem Sie die Lösungen präsentieren) fertig haben.
`Sie dürfen die Langaufgaben in Gruppen von bis zu drei Studierenden bearbeiten und präsentieren.`

**Ausserdem Wichtig:** Der Quellcode ist natürlich mit sinnvollen Kommentaren und die Schleifen mit zugehörigen Invarianten (inkl. Bereich in dem die Invariante gilt) zu kommentieren. Des Weiteren ist der Quellcode (insbesondere die Invarianten der Schleifen) mit *Assertions* (siehe Hinweis auf Blatt 2) zu überprüfen!

### Langaufgabe 6.1 (4 Punkte)

*Lernziel: IntervalScheduling*

Implementieren Sie den Algorithmus *IntervalScheduling* aus der Vorlesung und testen Sie diesen auf den gegebenen Eingabedaten. Gehen Sie dazu wie folgt vor:

- Laden Sie die Datei `VorlagenBlatt06.zip` von der DAP 2 - Praktikumsseite herunter und entpacken Sie die Dateien in Ihr Arbeitsverzeichnis.
- Erstellen Sie eine Klasse `Interval`, die zwei Integer-Attribute verwaltet, mit folgenden Methoden:
  - Einen Konstruktor, der zwei Integer-Parameter erwartet, einen für den Start des Intervals und den anderen für das Ende.
  - Eine Methode `public int getStart()` und eine Methode `public int getEnd()`, die jeweils die Werte zurückgeben.
  - Eine Methode `public String toString()`, die eine geeignete Stringrepräsentation des Intervals zurückgibt.

- Erstellen Sie eine Klasse `Anwendung` mit:
  - Einer Methode `public static ArrayList<Interval> intervalScheduling (ArrayList<Interval> intervals)`, die den `IntervalScheduling`-Algorithmus aus der Vorlesung implementiert. (Beachten Sie die Hinweise zur ArrayList)
  - Einer `main`-Methode, die als Parameter einen Dateipfad, auf eine der vorgegebenen Textdateien erwartet. Die Datei soll eingelesen werden und die Werte sollen in ein Array der Klasse Interval geladen werden. Dabei sollen die möglichen Exceptions des StringParsers explizit behandelt werden. (Beachten Sie dazu die Hinweise)
  - Auf diese Werte sollen Sie dann den Algorithmus ausführen und die Ergebnisse den Vorgaben ähnlich ausgeben. (siehe `sollAusgabeBsp.txt`, `sollAusgabeMittel.txt` oder `sollAusgabeGross.txt`)

**Langaufgabe 6.2** **(4 Punkte)**
*Lernziel: LatenessScheduling*

Für die Implementierung des LatenessScheduling-Algorithmus gehen Sie wie folgt vor.

- Erstellen Sie eine Klasse `Job`, analog zu der Klasse `Interval` aus Aufgabenteil 1. Statt des Startzeitpunktes soll die Klasse die Dauer und statt des Endzeitpunktes die Deadline des Jobs beinhalten.
- Erweitern Sie Ihre Klasse Anwendung wie folgt:
- Statt eines Parameters soll die `main`-Methode zwei übergeben bekommen. Der Erste soll von der Form "Interval" oder "Lateness" sein und für die Auswahl zwischen Interval- und LatenessScheduling sorgen. Der Zweite soll diesmal der Dateipfad sein.
- Benutzen Sie wieder die gleichen Dateien als Eingabe, allerdings sollen die Werte je nach Auswahlparameter entweder als Intervallgrenzen für das IntervalScheduling oder als Dauer und Deadline eines Jobs für das LatenessScheduling interpretiert werden.
- Schreiben Sie eine Methode `public static int[] latenessScheduling(ArrayList<Job> jobs)`, die den `LatenessScheduling`-Algorithmus aus der Vorlesung implementiert.
- Passen Sie Ihre Ausgabe entsprechend der Vorgaben an und geben Sie vor allem auch die maximale Verspätung mit aus.

**Beachten Sie die Hinweise und Tipps auf den folgenden Seiten.**

# Hinweise und Tipps

## 6.1 Einlesen einer Datei

Um in Java eine Text-Datei einzulesen, kann die Klasse `RandomAccessFile` benutzt werden. Zuerst erstellen Sie mittels des Dateipfades ein Objekt der Klasse `RandomAccessFile`. Der Konstruktor der Klasse benötigt zusätzlich einen zweiten String-Parameter, der die Zugriffsrechte auf die Datei definiert. Mit der Methode `readLine()` können Sie sich jeweils die nächste Zeile aus der Datei herausziehen. Die vorgegebenen Dateien sind in jeder Zeile so formatiert, dass zwei Integer Zahlen nur mit einem Komma getrennt sind. Um aus einer Zeile die beiden Integer zu parsen, kann die Klasse `StringTokenizer` benutzt werden. Im Konstruktor erwartet die Klasse als erstes den zu trennenden String und als zweites einen String mit den Zeichen, welche für die Trennung sorgen. Mit der Funktion `nextToken()` kann der jeweils nächste Teilstring rausgeholt werden. Beachten Sie die möglichen Exceptions die hierbei geworfen werden können. Diese sollen abgefangen werden und durch eine geeignete Ausgabe beschrieben werden. Für detaillierte Informationen über die Klassen besuchen Sie bitte die Java-API:

- `http://download.oracle.com/javase/6/docs/api/java/io/RandomAccessFile.html`
- `http://download.oracle.com/javase/6/docs/api/java/util/StringTokenizer.html`

**Beispiel:**

```
import java.io.RandomAcessFile;
import java.util.StringTokenizer;
...
RandomAccessFile file = new RandomAccessFile(path,"r");
...
String zeile = file.readLine();
...
StringTokenizer st = new StringTokenizer(zeile,",");
int start = Integer.parseInt(st.nextToken());
int ende = Integer.parseInt(st.nextToken());
Interval ivall = new Interval(start,ende);
...
```

## 6.2 ArrayList

Die Klasse ArrayList wird mittels `import java.util.ArrayList;` für Sie verfügbar. Die wichtigsten Methoden sind:

- add(E e) zum Einfügen eines Elements.
- isEmpty() zur Überprüfung ob die Liste leer ist.
- size() gibt die Größe der ArrayList zurück.
- get(int index) um das Element an Position index der Liste übergeben zu bekommen.

Für detaillierte Informationen schauen Sie bitte in der Java-Api nach:

`http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html`

## 6.3 Sortieren von Objekten

Objekte einer selbst geschriebenen Klasse können mit Hilfe der Methode `Collections.sort` sortiert werden, wenn die zu sortierende Klasse das `Comparable`-Interface implementiert. Dafür müssen sie (am Beispiel der Klasse Interval) eine Methode `public int compareTo(Interval other)` erstellen, die:

- einen positiven Integerwert zurückgibt, wenn `other` größer als das Element ist.
- Null (als Integer, nicht `null`!) zurückgibt, wenn `other` genau so groß ist wie das Element.
- einen negativen Integerwert zurückgibt, wenn `other` kleiner als das Element ist.

Für detaillierte Informationen schauen Sie bitte in der Java-Api nach:

`http://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html`

Eine `ArrayList` mit Objekten die diese Anforderung erfüllen kann wie folgt sortiert werden:

```
import java.util.Collections;
...
ArrayList<Interval> array = new ArrayList<Interval>();
...
Collections.sort(array);
```