Übungsblatt 3 (Block 1)

Prof. Dr. Olaf Hellwich und Mitarbeiter

Suchen und Komplexitätsberechnung

Verfügbar ab:	**07.05.18**
Abgabe bis:	**14.-18.05.18**

Aufgabe 1: Binäre Suche im VideoOnDemand-Angebot — 6 Punkte

In dieser Aufgabe vervollständigen Sie die Implementierung der Angebotsübersicht eines VideoOnDemand-Anbieters.

Das Angebot ist wie folgt modelliert:

- Der Anbieter hat einen Bestand, implementiert als Array von Filmen
- Ein Film besteht aus Titel (String), Preis [EUR] (double), Länge [min] (int), Beschreibung (String) und Erscheinungsdatum [JJJJ-MM-TT] (String).
- Alle Filme sind stets nach ihrem Erscheinungsdatum sortiert gespeichert. Zur Vereinfachung soll davon ausgegangen werden, dass keine zwei verschiedenen Filme am gleichen Tag erschienen sein können.
- Eine Schnittstelle zur Datenbank zum Hinzufügen und Entfernen von Filmen ist nicht Bestandteil dieser Implementierung.

Die vorgegebenen Dateien sind:

- `videos.db` – Textdatei, enthält den Filmbestand zum Programmstart und Programmende
- `Terminal.java` – Java-Klasse zur Eingabe von Daten
- `Parser.java` – Java-Klasse zum Einlesen der Initialartikeldaten
- `VideoOnDemand.java` – Hauptprogramm

Die folgenden Klassen sind teilweise vorgegeben und zu vervollständigen:

- `Film.java` – Java-Klasse, die einen Film repräsentiert
- `Suche.java` – Java-Klasse, die die Durchsuchung von Film-Arrays ermöglicht

Klasse `Film`

1. Konstruktor
   - Vervollständigen Sie in der Klasse den erweiterten Konstruktor, der alle Attribute mit den übergebenen Parametern initialisiert.
2. `public String toString()`
   - Implementieren Sie in `Film` eine Methode `public String toString()`, die einen String mit allen relevanten Informationen des Film-Objekts zurückliefert.
3. `public int compareTo(Film o)`
   - Die Klasse `Film` soll eine Methode `public int compareTo(Film o)` besitzen, welche das jeweilige Objekt dieser Klasse mit einem übergebenen `Film`-Objekt `o` vergleicht.

- Der Rückgabewert der Methode `int compareTo(Film o)` ist durch den Erscheinungsdatumsvergleich beider Filme bestimmt[1].
- Es soll gelten bzgl. des Rückgabewertes von `int compareTo(Film o)`:
  - Rückgabewert > 0: Das Erscheinungsdatum dieses `Film`-Objekts ist später als das von `o`
  - Rückgabewert = 0: Das Erscheinungsdatum von diesem `Film`-Objekt und `o` sind gleich
  - Rückgabewert < 0: Das Erscheinungsdatum dieses `Film`-Objekts ist früher als das von `o`
  - Hinweis: Die Java-Klasse `String` verfügt über eine `compareTo`-Methode, mit der sich Strings untereinander vergleichen lassen.

Klasse Suche

1. Methode `public static int binaereSuche(Film suchObjekt, Film[] array)`
   - Zurückgegeben wird der gefundene Index des übergebenen Artikelobjekts `suchObjekt` im Array `array`. Falls der Film nicht gefunden wurde, gibt die Methode -1 zurück.
   - Verwenden Sie für den Vergleich von zwei `Film`-Objekten die von Ihnen implementierte `compareTo`-Methode, die die Erscheinungsdaten vergleicht.
   - Implementieren Sie den Algorithmus zur binären Suche.
   - Testen Sie Ihre Implementierung, indem Sie die fertig vorgegebene Klasse `VideoOnDemand` kompilieren und ausführen. Suchen Sie nach verschiedenen Filmen. Ist der Film in der Übersicht, muss Ihre Suche ihn finden.

Musterlösung:

```
1  public class Suche {
2     // Suchmethode
3     public static int binaereSuche(Film suchObjekt, Film[] array){
4        // Indexvariablen fuer den kleiner werdenden Suchintervall im
             Array
5        int anfang = 0;
6        int ende = array.length -1;
7        int mitte;
8
9        while (true) {
10          mitte = (anfang + ende) / 2; // abgerundet
11
12          if (array[mitte].compareTo(suchObjekt) < 0) { // rechts
                weitersuchen
13             anfang = mitte + 1;
14             if (anfang > ende) return -1; // Rand war bereits erreicht
15          } else if (array[mitte].compareTo(suchObjekt) > 0) { // links
                weitersuchen
16             ende = mitte - 1;
17             if (anfang > mitte) return -1; // Rand war bereits erreicht
18          } else { // Treffer
19             return mitte; // index zurueckgeben
20          }
21       }
```

[1] Denkbar wäre z.B. auch, Filme nach Preis oder nach Titel zu vergleichen, falls wir dies wünschen würden. Unsere Filme sind für diese Übung jedoch stets nach dem Erscheinungsdatum sortiert.

```
    }
}
```

```
public class Film {

    // Attribute
    public String titel;
    public double preis; // in EUR
    public int laenge; // in min
    public String beschreibung;
    public String erscheinungsdatum; // ISO-8601 (JJJJ-MM-TT)

    // Konstruktor
    public Film(String titel, double preis, int laenge, String
        beschreibung, String erscheinungsdatum) {
        this.titel = titel;
        this.preis = preis;
        this.laenge = laenge;
        this.beschreibung = beschreibung;
        this.erscheinungsdatum = erscheinungsdatum;
    }

    // Methoden
    public String toString() {
        return "Film={" +
               "titel=" + this.titel + "," +
               "preis=" + this.preis + "," +
               "laenge=" + this.laenge + "," +
               "beschreibung=" + this.beschreibung + "," +
               "erscheinungsdatum=" + this.erscheinungsdatum +
               "}";
    }

    public int compareTo(Film other) {
        int rc;
        if (this.erscheinungsdatum.compareTo(other.erscheinungsdatum) > 0)
            rc = 1;
        else if (this.erscheinungsdatum.compareTo(other.erscheinungsdatum)
            < 0) rc = -1;
        else rc = 0;
        return rc;
    }

}
```

Aufgabe 2: Komplexität

4 Punkte

Laufzeitabschätzung `void work(int[] arr)`

Betrachten Sie die folgenden Java-Methoden:

```
public void work(int[] arr) {

    int versuche = 10;
    int[] ergebnis = new int[versuche];

```

```
6     for (int i = 0; i < versuche; i++) {
7        ergebnis[i] = eins(arr, 2);
8        System.out.println("" + (i+1) +": " + ergebnis[i]);
9     }
10
11 }
12
13 public int eins(int[] array, int modus) {
14
15    if (modus == 1) {
16       int ergebnis = 0;
17       for (int i = 0; i < array.length; i++) ergebnis += array[i];
18       return ergebnis;
19    } else if (modus == 2) {
20       if (array.length % 2 == 0) {
21          return zwei(array[0]) + zwei(array[array.length-1]);
22       } else {
23          int ergebnis = 1;
24          for (int i = 0; i < array.length; i++) {
25             ergebnis += zwei(array[i]);
26          }
27          return ergebnis;
28       }
29    } else if (modus == 3) {
30       int ergebnis = 1;
31       for (int i = 0; i < array.length; i++) ergebnis *= array[i];
32       return ergebnis;
33    } else {
34       return -1;
35    }
36
37 }
38
39 public int zwei(int n) {
40    // Nicht dargestellt. Hat einen Aufwand von 5n.
41 }
```

In dieser Aufgabe soll der Laufzeitaufwand der Methode `void work(int[] arr)` abgeschätzt werden. Die Problemgröße n ist hier die Länge des übergebenen Arrays `arr`.

Als Aufwand soll nur das Aufrufen der Hilfsmethode `int zwei(int n)` berücksichtigt werden[2]. Pro Aufruf dieser Hilfsmethode entsteht ein Aufwand von $5n$.

1. Die Auswertung von welchem Ausdruck entscheidet, ob der Worst-Case vorliegt?
2. Geben Sie die Worst-Case-Laufzeitabschätzung samt Lösungsweg und Einordnung in die kleinstmögliche Komplexitätsklasse an.
3. Geben Sie die Best-Case-Laufzeitabschätzung samt Lösungsweg und Einordnung in die kleinstmögliche Komplexitätsklasse an.
4. Gehen Sie nun davon aus, dass der Methode `int work(int[] arr)` in 75% der Aufrufe mit einem Array mit ungerader Länge aufgerufen wird. Geben Sie diese Average-Case-Laufzeitabschätzung samt Lösungsweg und Einordnung in die kleinstmögliche Komplexitätsklasse an.

[2]Zählen Sie also keine Ausdrücke wie Vergleiche oder arithmetische Operationen mit.

Musterlösung:

1. Zeile 20 unterscheidet. Der Programmfluss führt auf immer dem gleichen Weg bis zum Ausdruck `if (array.length % 2 == 0)` und verzweigt sich hier. Ist die Arraylänge ungerade, wird ein anderer (und aufwandsmäßig längerer) Weg eingeschlagen, als wenn die Arraylänge gerade ist.
2. Aufwand: $10 * n * 5n = 50 * n^2$, Aufwandsklasse $O(n^2)$
3. Aufwand: $10 * 3 * 5n = 150n$, Aufwandsklasse $O(n)$
4. Aufwand: $10 * (0.75 * (n * 5n) + 0.25 * (3 * 5n)) = 10 * ((3,75n^2) + 3,75n) = 37,5n^2 + 37,5n$, Aufwandsklasse $O(n^2)$