

Algorithmen, Datenstrukturen und Programmieren 2

SS 2003

1. Entwerfen Sie eine Klasse `ObjectTester` zur Analyse von Klassen zur Laufzeit. Beim Aufruf von `ObjectTester` soll das Programm den Namen einer Klasse als Kommandozeilenparameter übernehmen. Es soll eine Instanz der Klasse, deren Name eingelesen wurde, anlegen und feststellen, über welche Methoden die Klasse verfügt. Dann sollen alle Methoden der Klasse, die statisch sind und keine Eingabeparameter brauchen, aufgerufen werden (verwenden sie die `Modifier`-Methode `isStatic(int)`). Schreiben Sie auch eine Testklasse zur Überprüfung der Funktionalität von `ObjectTester`.
2. *Doppelt verkettete Listen*: Implementieren Sie eine doppelt verkettete Liste `DoublyLinkedList` wie folgt: Die Listenknoten `DLNode` sollen nun im Gegensatz zu einfach verketteten Listen auch eine Instanzvariable `prev` enthalten, die auf den jeweils vorhergehenden Knoten zeigt. `DoublyLinkedList` soll über alle Methoden verfügen, die auch `MyList` aus der Vorlesung besitzt. Überlegen Sie bei der Implementierung jeweils, welche Änderungen im Vergleich zu `MyList` durchzuführen sind. Außerdem sollen folgende neue Methoden geschrieben werden:

```
public boolean isInList (Object o)
// stellt fest, ob o in Liste vorhanden ist

public boolean insertAfter (Object what, Object where)
// fuegt what nach der Stelle, an der where liegt,
// in die Liste ein, retourniert bei Erfolg true

public boolean insertBefore (Object what, Object where)
// fuegt VOR where ein

public boolean removeFromList (Object what)
// entfernt das Objekt what aus der Liste
```

Schreiben Sie auch eine Testapplikation, die die Korrektheit *aller* neu geschriebenen Methoden demonstriert.

3. Modifizieren Sie die Klasse `MyTree` aus der Vorlesung so, dass die Datenfelder der Knoten des Baumes vom Typ `Comparable` sind. Wie

Sie der API-Dokumentation entnehmen können, erlaubt die entstehende Klasse `BinaryTree` nun, beliebige Klassen, die das Interface `Comparable` implementieren, in einen binären Suchbaum einzuordnen. Testen Sie diese Behauptung in einer Applikation an Hand von zwei verschiedenen Klassen, die `Comparable` implementieren (z.B. `String`).

4. Schreiben Sie eine Klasse `NonDuplicate`, die mehrfach vorkommende Worte aus einer Eingabe entfernt: der User soll zunächst über die Tastatur eine Anzahl von Wörtern eingeben können (jeweils durch Enter getrennt). Diese sollen in eine `ArrayList` eingelesen werden. Dann soll eine Methode `printNonDuplicates(Collection c)` auf diese `ArrayList` angewendet werden, die mit Hilfe eines `HashSet` die voneinander verschiedenen Einträge der Collection auf der Standardausgabe druckt.
5. Erstellen Sie eine Klasse `FirstLetter`, die eine Statistik über die Anzahl der Wörter mit jeweils gleichem Anfangsbuchstaben ausgibt. Die Wörter sollen genauso wie in Aufgabe 4 in eine `ArrayList` eingelesen werden. Anschließend sollen die Einträge dieser `ArrayList` in eine `HashMap` eingeordnet werden. Verwenden Sie als Keys die Anfangsbuchstaben (Typ `Character`), als Values die Anzahl des Vorkommens (Typ `Integer`). Falls ein Key zum ersten Mal eingeordnet wird, soll er den Value 1 erhalten, falls er schon in der `HashMap` ist, soll sein Value um 1 erhöht werden.
6. Schreiben Sie eine Applikation `SatzAnalyse` (als Unterklasse von `JFrame`), die einen Satz über ein `JTextField` einliest und dann folgende Daten ausgibt:
 - a) Die Zahl der mehrfach vorkommenden Wörter (dabei soll Groß- und Kleinschreibung ignoriert und Satz- und Trennungszeichen herausgefiltert werden).
 - b) Die voneinander verschiedenen Wörter des Satzes in alphabetischer Reihenfolge.
 - c) Die voneinander verschiedenen Wörter des Satzes in umgekehrter Reihenfolge.

Überlegen Sie zuerst, welche Datenstrukturen dem Problem angemessen erscheinen. Wahrscheinlich werden Sie die Klasse `StringTokenizer` verwenden wollen. Außerdem ist vielleicht die `Collections`-Methode `reverseOrder()` von Nutzen.

7. Verwenden Sie die Klasse `RandomFileTest` als Ausgangspunkt, um eine Klasse `PersonalVerwaltung` (wieder als Unterklasse von `JFrame`) zu programmieren: der User soll zunächst über Textfelder Name und Gehalt des Angestellten festlegen können. Das Programm soll dann die Nummer des jeweiligen Angestellten anzeigen und die Daten in einem File abspeichern.

Es soll möglich sein, über die Personalnummer auf die einzelnen Angestellten zuzugreifen, ihre Daten anzeigen zu lassen und ihr Gehalt zu verändern. Außerdem soll die Ausgabe des gesamten Datenfiles auf die Standardausgabe möglich sein. Bei Aufruf des Programmes soll die Zahl der derzeit im File abgespeicherten Angestellten angezeigt werden.

8. Modifizieren Sie das Programm `TableDisplay2.java` so, dass die Resultate der einzelnen Abfragen jeweils zusätzlich in Textdateien abgespeichert werden. Wenn der User den `Submit`-Button drückt, soll ein File namens `abfrageX.txt` angelegt werden, wobei `X` die Nummer der Abfrage ist. In dieses File soll zunächst der Abfrage-String geschrieben werden und danach als durch Kommata getrennte Liste das Resultat der Abfrage. Ein Blick auf die `showTable`-Methode des Programmes `MakeDB.java` ist eventuell von Nutzen...
9. Modifizieren Sie die Client-Server Applikation aus 5.4 so, dass der Client ein Textfile anfordern kann. Falls dieses File am Server nicht vorhanden ist, soll eine entsprechende Meldung an den Client gesendet werden. Ansonsten soll das gesamte File zeilenweise an den Client geschickt werden.