

Einführung in die Programmierung

WS 2017/2018, Blatt 4

Ausgabe: 15.01.2018

Prof. Norbert Siegmund
Nathalie Dittrich

Abgabetermin: Montag, 29.01.2018, 11:00 Uhr

Bitte lesen Sie die folgenden Informationen zum Übungsablauf **sorgfältig** durch.

Grundsätzlich – wenn nicht anders angegeben – sind die Lösungen zu den Übungen zu Programmierung I jeden **Dienstag bis spätestens 12:00 Uhr** an die jeweiligen Tutoren per E-Mail zu schicken.

Schreiben Sie bitte im Betreff Ihrer E-Mail Ihre **Teamnummer** sowie die Nummer des Übungsblattes. In der E-Mail schreiben Sie bitte zusätzlich Ihren **Namen** und **Matrikelnummer**. Die Lösungen für Sie bitte als Java Dateien als Anlage hinzufügen. Es werden **keine** kompilierten Dateien, wie *.class oder *.jar angenommen.

Übungen müssen von **minimal ein** und **maximal zwei** Studierenden aus derselben Übungsgruppe in einem festen Team bearbeitet werden (Ausnahmen nur auf Anfrage beim Übungsleiter). Pro Team soll die Lösung nur einmal abgegeben werden. Aufgaben sollen **im Team gelöst** und nicht nur vom Team abgegeben werden. Sie müssen mindestens **50%** dieser Punkte für eine Zulassung zur Prüfung erreichen. Das **Abschreiben** identischer Lösungen wird jeweils mit 0 Punkten bewertet.

Bei Fragen oder Unklarheiten wenden Sie sich bitte **vor der Abgabe** des Übungsblattes an den Übungsleiter (per E-Mail oder persönlich). Es soll nie jemand sagen müssen: „Wir haben die Aufgabe nicht verstanden und konnten sie daher nicht bearbeiten.“

Weitere Informationen, wie aktuelle Ankündigungen, die Angaben finden Sie online (<https://www.uni-weimar.de/de/medien/professuren/intelligente-softwaresysteme/lehre/>) unter Einführung in die Programmierung

Aufgabe 1 Polymorphie (6 Punkte)

Betrachten Sie das beigefügte Java Programm. Wo wird hier Polymorphie verwendet? Um welche Art handelt es sich jeweils? Es reicht, wenn Sie Kommentare in die entsprechenden Java-Klassen schreiben.

Bewertung:

- Polymorphie entdeckt: 3 Punkte
- richtige Art der Polymorphie genannt: 3 Punkte

Aufgabe 2 Listen (22 Punkte)

Erstellen Sie einen Datentyp (eine Klasse **DLList**) für doppelt verkettete Listen. In der Liste sollen Strings gespeichert werden. Die Knoten sollen eine Referenz auf den nachfolgenden als auch auf den vorhergehenden Knoten haben.

(a) Die Knoten Klasse (**Node**) soll folgende Eigenschaften implementieren:

- (i) `public Node(String data)` - der Konstruktor.
- (ii) `public String toString()` soll `{stringvalue}` ausgeben.
- (iii) `public void setNextNode(Node next)` setzt den nächsten Knoten auf `next`.
- (iv) `public void setPreviousNode(Node previous)` setzt den vorherigen Knoten auf `previous`.
- (v) `public Node getNextNode()` gibt die Referenz auf den nächsten Knoten oder `null` zurück, falls der Knoten der letzte der Liste ist.
- (vi) `public Node getPreviousNode()` gibt die Referenz auf den vorherigen Knoten oder `null` zurück, falls der Knoten der erste der Liste ist.
- (vii) `public String getValue()` gibt den gespeicherten Wert zurück.

(b) Folgende Methoden sollen von der Listen Klasse implementiert werden:

- (i) `public boolean isEmpty()` gibt zurück ob die Liste leer ist.
- (ii) `public void display()` gibt die Liste auf der Konsole aus.
- (iii) `public void add(int pos, String content)` fügt den `String content` an der Position `int pos` ein.
- (iv) `public void add(String content)` fügt `String content` am Ende der Liste ein.
- (v) `public void remove(String content)` löscht den ersten Knoten, der `String content` enthält aus der Liste.

- (vi) `public void removeFirst()` löscht den ersten Knoten der Liste.
- (vii) `public void removeLast()` löscht den letzten Knoten der Liste.
- (viii) `public void clear()` löscht alle Knoten aus der Liste.
- (ix) `public String getFirst()` gibt den String des ersten Knotens der Liste zurück.
- (x) `public String getLast()` gibt den String zurück, der im letzten Knoten der Liste gespeichert ist.
- (xi) `public String get(int pos)` gibt den String an der Position `int pos` zurück.
- (xii) `public void concat(DLList list)` fügt `DLList list` am Ende der Liste an.
- (xiii) `public int find(String content)` überprüft ob der `String content` in der Liste enthalten ist. Ist dies der Fall, so gibt die Methode die Knotennummer zurück oder `-1` falls der String nicht enthalten ist.
- (xiv) `public boolean contains(String content)` gibt zurück ob die Liste `String content` enthält.
- (xv) `public int size()` gibt die Anzahl der Knoten zurück.

Testen Sie Ihre Implementierung ausführlich und geben Sie das Testprogramm inkl. dessen Ausgaben mit ab.

Bewertung:

- 1 Punkt pro richtige Teilaufgabe