Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Matthias Renz
Dr. Erich Schubert, Daniel Kaltenthaler

München, 14.4.2015

Algorithmen und Datenstrukturen

SS 2015

Übungsblatt 1: Wiederholung Grundlagen

Besprechung: 23.4.–28.4.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 22.4.2015 13:25

Aufgabe 1-1 Abgabesystem — Hausaufgabe

Machen Sie sich mit UniWorX vertraut, insbesondere mit dem Abgabesystem für Hausaufgaben.

- Ihre Lösung **muss** als `zip`-Archiv mit dem Namen `loesung01.zip` hochgeladen werden.
- Als Abgabe werden aus Kompatibilitätsgründen ausschließlich `.pdf` und `.java` Dateien akzeptiert, sofern dies nicht anders angegeben wird.
- Sie können jede Aufgabe in einer separaten Datei bearbeiten.
- Informieren Sie sich rechtzeitig, wie Sie aus Ihrer bevorzugten Software ein PDF exportieren
  Beispiel LibreOffice: Datei, Exportieren als PDF.
  Beispiel Linux: hier existiert i.d.R. die Option in eine PDF-Datei zu drucken.
- Eingescannte Lösungen müssen eine downloadfreundliche Dateigröße (möglichst $\ll 1$ MB) haben, aber dennoch für die Korrektoren gut lesbar sein. Hier ist es ggf. notwendig Kontrast und Helligkeit ihres Scanners sinnvoll einzustellen.
- Das Bearbeiten von Programmieraufgaben auf Papier ist nicht zulässig. Installieren Sie sich eine aktuelle Entwicklungsumgebung, oder verwenden Sie die am CIP-Pool vorinstallierte eclipse-Entwicklungsumgebung.
- Compilieren und testen Sie Ihre Programmieraufgaben, um Fehler zu finden.
- *Keinesfalls* geben Sie OpenOffice oder Word-Dateien direkt ab. Diese werden von UniWorX fälschlicherweise als zip-Dateien erkannt, und *können danach nicht mehr geöffnet werden*. Auch solche Dateien *müssen* Sie als `.pdf` exportieren, damit Sie vom Tutor korrigiert werden können!
- Nicht lesbare (sowie nicht öffnebare) Lösungen werden nicht korrigiert.
  Programmier-Aufgaben müssen mit Java 1.7 oder Java 1.8 compilierbar sein.

Aufgabe 1-2 Eigenschaften von Algorithmen nicht bewertet

Stellen Sie sich einen Karteikasten vor, in dem Sie nach einer Karteikarte mit einem bestimmten Titel suchen. Gehen Sie dabei von folgenden Annahmen aus:

- Der Karteikasten enthält nicht mehrere Karteikarten mit dem selben Titel.
- Der Karteikasten enthält beim Aufruf des Algorithmus mindestens eine Karte.
- Eine gültige Eingabe ist auch ein Titel, der nicht im Karteikasten vorkommt.

Suchverfahren 1:
(1) Sie nehmen die erste Karte aus dem Karteikasten heraus.
(2) Sie geben diese Karte zurück.

Suchverfahren 2:
(1) Sie greifen zufällig eine Karte heraus.
(2) Hat diese den gewünschten Titel, so sind Sie fertig, ansonsten legen Sie sie zurück und wiederholen das Verfahren ab Schritt 1.

Suchverfahren 3:
(1) Sie greifen zufällig eine Karte heraus.
(2) Hat diese den gewünschten Titel, so sind Sie fertig, ansonsten legen Sie sie zurück.
(3) Sind seit Beginn der Suche zehn Minuten vergangen, geben Sie auf und beenden die Suche.
(4) Ansonsten wiederholen Sie das Verfahren ab Schritt 1.

Suchverfahren 4:
(1) Sie greifen zufällig eine Karte heraus.
(2) Trägt diese den gewünschten Titel, so legen Sie die Karte auf einen Ergebnisstapel, ansonsten legen Sie sie auf die Seite.
(3) Ist der Kasten leer, so beenden Sie die Suche.
(4) Ansonsten wiederholen Sie das Verfahren ab Schritt 1.

Suchverfahren 5: (Vorbedingung: Karteikasten ist lexikographisch sortiert)
(1) Sie greifen die Karte, die in der Mitte liegt.
(2) Trägt diese den gewünschten Titel, legen Sie die Karte auf den Ergebnisstapel und beenden die Suche, ansonsten:
(3) Ist der gesuchte Titel alphabetisch vor dem Titel auf der ausgewählten Karte, so suchen Sie nur noch in der ersten Hälfte nach dem gleichen Verfahren ab Schritt 1 weiter.
(4) Ist der gesuchte Titel alphabetisch hinter dem Titel auf der ausgewählten Karte, so suchen Sie nur noch in der zweiten Hälfte nach dem gleichen Verfahren ab Schritt 1 weiter.
(5) Das Verfahren ist erfolglos beendet, wenn der letzte Kartenabschnitt leer ist.

Vergleichen Sie die oben angeführten Suchverfahren.
Welcher Algorithmus erfüllt welche der folgenden Eigenschaften?

(a) terminierend
(b) deterministisch
(c) determiniert
(d) partiell korrekt
(e) total korrekt

Begründen Sie Ihre Entscheidung kurz.

Aufgabe 1-3 Verkettete Listen nicht bewertet

Die Datenstruktur einer **einfach verankerten verketteten Liste** haben Sie bereits in der Vorlesung "Einführung in die Programmierung" kennen gelernt.

Rufen Sie sich diese Datenstruktur in Erinnerung, und ergänzen Sie folgende Implementierung:

- In dem unten folgenden Codefragment finden sie eine äußere Klasse, eine innere Klasse. An zwei Stellen werden Generics definiert und ein Interface wird implementiert. Identifizieren Sie, welche Stellen im Programm damit gemeint sind.
- Warum haben wir `Liste`, `Knoten`, `Iterable` mit einem Großbuchstaben, `kopf`, `daten`, `naechstes` klein, und `INHALT` komplett groß geschrieben? Ist dies notwendig?
- Ergänzen Sie die Methoden `size`, `prepend` und `append`:

```
import java.util.*;

/* Verkettete Liste für Daten vom Typ INHALT */
public class Liste<INHALT> implements Iterable<INHALT> {
  /* Kopf der Liste */
  Knoten<INHALT> kopf;

  /* Konstruktor für eine leere Liste */
  public Liste() {
    kopf = null; // Leer
  };

  /* Länge der Liste berechnen */
  public int size() {
    // TODO: Ergänzen
  }

  /* Element am Anfang einfügen */
  public void prepend(INHALT daten) {
    // TODO: Ergänzen
  }

  /* Element am Ende einfügen */
  public void append(INHALT daten) {
    // TODO: Ergänzen
  }

  /* Klasse, die ein Element der Liste speichert */
  protected static class Knoten<INHALT> {
    /* Gespeichterte Daten */
    protected INHALT daten;

    /* Nächstes Element */
    protected Knoten<INHALT> naechstes;

    /* Konstruktor für Knoten */
    public Knoten(INHALT daten, Knoten<INHALT> naechstes) {
      this.daten = daten;
      this.naechstes = naechstes;
    }
  }
}
```

Hinweis: auch mit den ergänzten Methoden wird dieser Code noch nicht compilieren.
Welchen Fehler meldet Ihr Compiler? Was bedeutet der Fehler?

- Ergänzen Sie einen `Iterator`, um den Inhalt der Liste auslesen zu können:

```
public class Liste<INHALT> implements Iterable<INHALT> {
  // ... wie zuvor

  /* Einen Iterator für die Liste erzeugen */
  public Iterator<INHALT> iterator() {
    return new Iter<INHALT>(kopf);
  }

  /* Implementierung eines Iterators */
  protected static class Iter<INHALT> implements Iterator<INHALT> {
    /* Aktuelle position */
    Knoten<INHALT> position;

    /* Konstruktur */
    public Iter(Knoten<INHALT> position) {
      this.position = position;
    }

    /* Zum nächsten Element gehen */
    public INHALT next() {
      // TODO: Ergänzen
    }

    /* Test, ob die Liste ein nächstes Element hat */
    public boolean hasNext() {
      // TODO: Ergänzen
    }

    /* Entfernen nicht erlaubt. */
    public void remove() {
      throw new UnsupportedOperationException();
    }
  }
}
```

- Testen Sie ihre Implementierung (mindestens) mit folgendem Code:

```
public class Liste<INHALT> implements Iterable<INHALT> {
  // ... wie zuvor

  /* Methode zum Testen der Implementierung */
  public static void main(String[] args) {
    Liste<Integer> liste = new Liste<Integer>();
    liste.prepend(2);
    liste.prepend(1);
    liste.append(3);
    liste.append(4);
    liste.prepend(0);
    System.out.println("Ist die Länge 5? "+liste.size());
    System.out.println("Die nächstes Zeile sollte dies sein: '0 1 2 3 4'!");
    for (Integer i : liste) {
      System.out.print(i+" ");
    }
    System.out.println();
    System.out.println("Die nächste Zeile sollte leer sein, aber keinen Fehler verursachen!");
    for (Integer i : new Liste<Integer>()) {
      System.out.print(i+" ");
    }
    System.out.println();
  }
}
```