Klausur zur Vorlesung

Allgemeine Informatik II

Prof. J. Fürnkranz
Technische Universität Darmstadt — Sommersemester 2007 (Klausur WS07/08)
Termin: 28. 2. 2008

Name: **Vorname:** **Matrikelnummer:**

Fachrichtung: **Wiederholer:** ☐ ja ☐ nein

☐ Diplom ☐ Master ☐ Bachelor

Punkte: (1) .... (2) .... (3) .... (4) .... (5) .... **Summe:**

- **Aufgaben:** Diese Klausur enthält auf den folgenden Seiten 5 Aufgaben zu insgesamt 80 Punkten. Jede Aufgabe steht auf einem eigenen Blatt. Kontrollieren Sie *sofort*, ob Sie alle sechs Blätter erhalten haben!
- **Zeiteinteilung:** Die Zeit ist knapp bemessen. Wir empfehlen Ihnen, sich zuerst einen kurzen Überblick über die Aufgabenstellungen zu verschaffen, und dann mit den Aufgaben zu beginnen, die Ihnen am besten liegen.
- **Papier:** Verwenden Sie nur Papier, das Sie von uns ausgeteilt bekommen. Bitte lösen Sie die Aufgaben auf den dafür vorgesehenen Seiten (auch auf den Rückseiten). Falls der Platz nicht ausreicht, vermerken Sie dies bitte und setzen die Lösung auf einem Zusatzblatt fort. Brauchen Sie zusätzlich Papier (auch Schmierpaper), bitte melden.
- **Fragen:** Sollten Sie Teile der Aufgabenstellung nicht verstehen, bitte fragen Sie!
- **Abschreiben:** Sollte es sich (wie in den letzten Jahren leider immer wieder) herausstellen, daß Ihre Lösung und die eines Kommilitonen über das zu erwartende Maß hinaus übereinstimmen, werden beide Arbeiten negativ beurteilt (ganz egal wer von wem in welchem Umfang abgeschrieben hat).
- **Ausweis:** Legen Sie Ihren *Studentenausweis* und *Lichtbildausweis* sichtbar auf Ihren Platz. Füllen Sie das Deckblatt sofort aus!
- **Hilfsmittel:** Zur Lösung der Aufgaben sind keine Unterlagen erlaubt. Gedruckte Wörterbücher sind für ausländische Studenten erlaubt, elektronische Hilfsmittel (Taschenrechner, elektronische Wörterbücher, Handy, etc.) sind verboten! Sollten Sie etwas anderes verwenden wollen, bitte klären Sie das *bevor* Sie zu arbeiten beginnen.
- **Aufräumen:** Sonst darf außer Schreibgerät, Essbarem, von uns ausgeteiltem Papier und eventuell Wörterbüchern nichts auf Ihrem Platz liegen. Taschen bitte unter den Tisch!

Gutes Gelingen!

Aufgabe 1 (12 Punkte)

Gegeben seien folgende Klassendefinition:

```
1   public class A {
2      public static int sta_method();
3      public        int pub_method();
4      protected     int pro_method();
5      private       int pri_method();
6
7      public int x () {
8        return ...;   // Aufgabe a
9      }
10  }
11
```

```
12  public class B extends A {
13
14     public int x () {
15        return ...;  // Aufgabe b
16     }
17
18     public static int y () {
19       int n = 0;
20       ...          // Aufgabe c
21       return n;
22     }
23  }
```

Beantworten Sie folgende Fragen:

Anmerkung: Für falsch angekreuzte Antworten erhalten Sie Abzugspunkte! (In Summe können Sie aber natürlich nicht weniger als 0 Punkte für die gesamte Aufgabe erhalten.)

1–a Welche Werte können anstelle der drei Punkte (. . .) in Zeile 8 eingesetzt werden, ohne einen Fehler zu produzieren?

ja	**nein**	
☐	☐	`pri_method()`
☐	☐	`pub_method()`
☐	☐	`pro_method()`

1–b Welche Werte können anstelle der drei Punkte (. . .) in Zeile 15 eingesetzt werden, ohne einen Fehler zu produzieren?

ja	**nein**	
☐	☐	`pri_method()`
☐	☐	`pub_method()`
☐	☐	`pro_method()`

1–c Geben Sie für jeden der folgenden Aufrufe an, ob er in Zeile 20 gültig wäre.

Für gültige Aufrufe geben Sie den Wert an, der in Zeile 21 zurückgegeben wird, für ungültige Aufrufe geben Sie eine kurze Begründung, warum der Aufruf nicht möglich ist.

ja	**nein**		
☐	☐	`n = pub_method();`	_____
☐	☐	`pub_method();`	_____
☐	☐	`n = sta_method();`	_____
☐	☐	`sta_method();`	_____

1–d Nehmen Sie an, daß Sie in einem Programm zwei Instanzen `a` und `b` wie folgt definiert haben:

```
1    A a = new B();
2    B b = new B();
```

Welche der folgenden Aufrufe wären in einer darauffolgenden Zeile 3 gültig?

ja	**nein**		**ja**	**nein**	
☐	☐	`a.x();`	☐	☐	`b.x();`
☐	☐	`A.x();`	☐	☐	`B.x();`
☐	☐	`a.y();`	☐	☐	`b.y();`
☐	☐	`A.y();`	☐	☐	`B.y();`

Aufgabe 2 (12 Punkte)

Die Folge der Fibonacci-Zahlen ist dadurch definiert, daß jede Zahl f_n die Summe der beiden vorhergehenden Zahlen ist, also $f_n = f_{n-1} + f_{n-2}$. Die ersten beiden Zahlen der Folge sind $f_0 = 0$ und $f_1 = 1$. Der Rest der Folge ergibt sich daraus deterministisch:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

Schreiben Sie zwei verschiedene Versionen einer Methode `static int fibonacci(int n)`, die die `n`-te Fibonacci-Zahl f_n berechnet und als Rückgabewert zurückgibt.

2–a Implementieren Sie `fibonacci` nicht-rekursiv.

2–b Implementieren Sie `fibonacci` rekursiv.

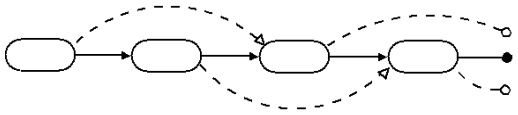
Aufgabe 3 (22 Punkte)

Gegeben sei folgende rekursive Datenstruktur für Listen, die in jedem Element zwei Verweise abspeichern:

- einen Verweis auf das nächste Element (`n1`, voll gezeichnete Linien)
- einen Verweis auf das übernächste Element (`n2`, gestrichelte Linien)

Zusätzlich wird in jedem Element noch eine Integer-Zahl `z` abgespeichert.

```
1    class NNList {
2       public int z;
3       public NNList n1;
4       public NNList n2;
5    }
```



Wenn kein nächstes oder übernächstes Element vorhanden ist, zeigt der Zeiger dementsprechend auf `null` (in der Skizze durch Kreise gekennzeichnet).

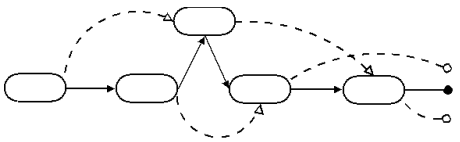
3–a Schreiben Sie einen Konstruktur, der eine Zahl übergeben bekommt, und eine Liste mit einem Element erzeugt, und dieses mit der übergebenen Zahl initialisiert.

3–b Schreiben Sie unter Verwendung des in a) definierten Konstrukturs einen weiteren Konstruktor, der einen Array von Integer-Zahlen übergeben bekommt, und eine `NNList` mit diesen Zahlen initialisiert.

Hinweis: Ein einfacher Lösungsweg besteht darin, die Liste zuerst nur mit den korrekten Zeigern auf die nächsten Elemente zu initialisieren, und erst danach in einem weiteren Durchlauf die Zeiger auf die jeweils übernächsten Elemente zu setzen. Sie können aber natürlich auch einen anderen Lösungsweg wählen.

3–c Schreiben Sie eine Methode `insert` mit einem Parameter `i`. Die Methode soll ein neues Listenelement mit der Zahl `i` an der übernächsten Stelle der Liste einfügen.

Nebenstehende Skizze zeigt das Resultat nach Einfügen eines neuen Elements, wenn `insert` für das Element ganz links in obiger Liste aufgerufen wird.



3–d Schreiben Sie eine Methode `add_ungerade`, die, beginnend mit dem momentanen ersten Listenelement, die Zahlen aller Elemente der Liste, die an ungeraden Positionen vorkommen, aufaddiert, und das Ergebnis als Rückgabewert zurückliefert.

Aufgabe 4 (28 Punkte)

Sie haben die Aufgabe, ein Programm zur Verwaltung von Sportresultaten zu schreiben.

4–a Definieren Sie eine abstrakte Klasse `Team`, in der der Name einer Mannschaft und eine reellwertige Punkteanzahl abspeichert werden kann. Der Name der Mannschaft soll für beliebige Klassen direkt einsichtig und veränderbar sein, die Punkteanzahl soll nur von Unterklassen von `Team` verändert werden können.

4–b Definieren Sie für die Klasse `Team` eine konkrete Methode `getPunkte`, die die aktuelle Punkteanzahl zurückliefert, und eine abstrakte Methode `updatePunkte`, die ein Objekt vom Typ `Spiel` (siehe Aufgabe c) übergeben bekommt.

4–c Definieren Sie ein Interface `Spiel`. Ein Spiel soll die folgenden vier Methoden zur Verfügung stellen:

- `heimTeam`, `gastTeam`: retournieren Objekte vom Typ `Team` und sind dafür gedacht, das jeweilige Heim und Auswärtsteam des Spiels zurückzuliefern.
- `heimScore`, `gastScore`: retournieren reelle Zahlen und sind dafür gedacht, das im Spiel erzielte Score für das Heim- und Auswärtsteam zurückzulieferen (z.B. die Anzahl der von jedem Team erzielten Tore).

4–d Definieren Sie die Klasse `FussballTeam` als eine Unterklasse von `Team`. Zusätzlich zur Punkteanzahl sollen in dieser Klasse noch die Anzahl der Siege und die Anzahl der Unentschieden, sowie die Tordifferenz (Differenz zwischen geschossenen und erhaltenen Toren) abgespeichert werden.

Weiters soll diese Klasse noch das Interface `Comparable` umsetzen (siehe Aufgabe f).

4–e Implementieren Sie eine statische Methode `updatePunkte`. Der Methode wird ein `Spiel`-Objekt übergeben, deren Methoden `heimScore`, `gastScore` die Anzahl der für das Heim- bzw. Auswärts-Team erzielten Tore retourniert. Die Methode soll, abhängig vom Spielergebnis, für jedes der beiden im Spiel involvierten Teams die Anzahl der Siege und Unentschieden, sowie die Tordifferenz anpassen.

4–f Implementieren Sie die Methode `public int compareTo(Object t)`, die vom Interface `Comparable` vorgeschrieben wird (siehe Aufgabe d). Die Methode soll

- $+1$ retournieren, wenn das eigene Team besser ist als `t`
- 0 retournieren, wenn beide gleich gut sind
- -1 retournieren, wenn das eigene Team schlechter als `t` ist

Ein Fussball-Team ist besser als ein anderes Fussball-Team, wenn es mehr Punkte erzielt hat oder bei gleicher Punktzahl eine höhere Tordifferenz hat.

Die Methode soll eine `ClassCastException` werfen, wenn das übergebene Objekt `t` nicht den dynamischen Typ `FussballTeam` hat (Überprüfung durch den Operator `instanceof`).

Aufgabe 5 (6 Punkte)

Nehmen Sie an, Sie haben Aufgabe 4 erfolgreich gelöst, d.h. Sie haben die Klasse `FussballTeam` definiert, die das Interface `Comparable` implementiert.

Sie wollen nun eine Tabelle aller Fussball-Teams erstellen.

5–a Welche der Collection Interfaces würde sich dafür eignen? Begründung?

5–b Wie müssen Sie vorgehen, um die Tabelle sortiert zu halten (die besten Mannschaften vorne)?

Hinweis: Beantworten Sie nur die Fragen, Sie müssen in dieser Aufgabe nichts implementieren!