Fachbereich Informatik

Klausur Einführung in die Informatik II für Elektrotechniker 21. April 2001

Name: .....

Matr.-Nr. .....

Bearbeitungszeit: 120 Minuten

Bewertung
(bitte offenlassen :–)

Aufgabe	Punkte	Erreichte Punkte
1	6	
2	6	
3	5	
4	6	
5	11	
6	10	
Summe	44	

Spielregeln (**Jetzt lesen!**):

- Benutzen Sie für die Lösung der Aufgaben **nur** das mit diesem Deckblatt ausgeteilte Papier. Lösungen, die auf anderem Papier geschrieben werden, können **nicht** bewertet werden. Schreiben Sie ihre Lösung auch auf die Rückseiten der Blätter; benötigen Sie für eine Lösung mehr als ein Blatt, finden Sie am Ende der Klausur Leerblätter. Zusätzliches Papier können Sie von den Tutoren bekommen.
- Tragen Sie jetzt (vor Beginn der eigentlichen Bearbeitungszeit !!!) auf *allen* Blättern ihren Namen und ihre Matrikelnummer ein.
- Schreiben Sie deutlich! Unleserliche oder zweideutige Lösungen können nicht gewertet werden.
- Schreiben Sie **nicht** mit Bleistift.
- Bitte schreiben Sie nicht mit rotem oder grünem Stift (das sind die Farben für die Korrektur).
- Lesen Sie die Aufgaben jeweils bis zum Ende durch; oft gibt es hilfreiche Hinweise!
- Wir weisen noch einmal darauf hin, daß die Benutzung von Taschenrechnern nicht gestattet ist.

Viel Erfolg!

- **AUFGABE 1 (6 Punkte) Theorie.**

1. (2 Punkte) Was versteht man unter *Vererbung* und was hat es mit *dynamischem Binden* zu tun?

2. (2 Punkte) Beschreiben Sie *kurz* das Grundprinzip von *Quicksort*. Welchen (Zeit-)Aufwand und welches Speicherplatzverhalten hat es?

3. (1 Punkt) Was versteht man unter einen *Stack*?

4. (1 Punkt) Was versteht man in GUIs unter einem *Container* und was unterscheidet ihn von anderen grafischen Komponenten?

- **AUFGABE 2 (6 Punkte) JAVA.**

1. (2 Punkte) Was ist eine Referenz in *JAVA*? Erläutern Sie zwei verschiedene Arten von *Gleichheit*, die in JAVA definiert sind.

2. (1 Punkt) Wie können Sie in *JAVA* Methoden schreiben, die ohne die Erzeugung eines entsprechenden Objektes aufgerufen werden können? Welche Einschränkungen gelten für solche Methoden?

3. (3 Punkte) Wo sind die Fehler im folgenden *JAVA*-Code? Beschreiben Sie die Fehler und geben Sie die Zeilenzahlen des Auftretens an.

```
 1 class Counter {
 2     public int counter;
 3
 4     NewCounter(int counter) {
 5         this.counter = counter;
 6     }
 7
 8     Counter() {
 9         clear();
10     }
11 }
12
13 class Sum extends Counter {
14     void clear() {
15         counter = 0;
16     }
17
18     int sum(int[] values) {
19         final int result = 0;
20         for (int i = 0; i < values.length; i++) {
21             result = result + values[i];
22         }
23         return result;
24     }
25 }
```

• AUFGABE 3 (5 Punkte) Numerik.

Schreiben Sie eine *JAVA*-Methode `double sin(double x, int minsteps)`, welche den Sinus per Approximation gemäß folgender Reihenentwicklung berechnet:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

Beenden Sie den Approximationsprozeß, wenn eine Genauigkeit von 10^{-8} erreicht ist. Formulieren Sie diese Bedingung mit Hilfe einer zu erstellenden Hilfsfunktion `boolean close( double x, double y, double eps )`. Um mögliche „Einschwingvorgänge“ zu umgehen, soll Ihr Approximationsprozeß mindestens die ersten `minsteps` Glieder der Reihe berechnen.

Ihre Methode soll möglichst keine überflüssigen, doppelten Berechnungen ausführen. Deshalb sollen Sie die Methode `Math.pow` für die Potenzfunktion *nicht* verwenden.

• AUFGABE 4 (6 Punkte) Binärbäume.

Bei der Bearbeitung dieser Aufgabe können Sie voraussetzen, daß Ihnen die Klasse `BinTree` aus dem Skript zur Verfügung steht. Sie stellt Ihnen die folgenden Methoden bereit:

- Die Konstruktoren `BinTree()`, `BinTree(Object o)` und `BinTree(Object o, BinTree l, BinTree r)`
- Die Zugriffsmethoden `Object value()`, `BinTree left()` und `BinTree right()`
- Die Hilfsfunktionen `boolean isEmpty()`, `boolean isLeaf()` und `boolean isNode()`

Erweitern Sie die Klasse `BinTree` um die folgenden Methoden:

1. (3 Punkte) Fügen Sie (zur Klasse `BinTree`) eine Methode `int Leafs()` hinzu, die die Anzahl der Blätter des Baumes berechnet.
2. (3 Punkte) Schreiben Sie ferner eine Methode `BinTree swapTree()`, welche einen „gespiegelten“ Baum erzeugt, bei dem in allen Knoten der linke und rechte Unterbaum (gegenüber dem Ausgangsbaum) vertauscht sind.

• AUFGABE 5 (11 Punkte) Listen.

In dieser Aufgabe sollen sie eine einfache *Kundenliste* einer Bank verwalten. Dabei sollen die Kunden nach ihrem Namen sortiert sein, zusätzlich ist ihr jeweiliger Kontostand gespeichert.

1. (3 Punkte) Erstellen Sie eine Klasse `Kunde`, welche die Daten eines Kunden enthält. Dies sind sein Name (als String) sowie sein Kontostand (als long), welche geeignet mittels Konstruktor setzbar sein sollten.

   Außerdem soll eine Methode `boolean kleiner(Kunde k)` vorhanden sein, welche angibt, ob der Name des aktuellen Kunden kleiner als der eines übergebenen anderen ist.

   Hinweis: Sie können zwei Strings mit der Methode `int String.compareTo(String other)` vergleichen. Das Ergebnis ist negativ/Null/positiv, falls der entsprechende String kleiner/gleich/größer als der andere ist.

2. (1 Punkt) Schreiben Sie ferner eine Klasse `ListCell`, welche ein einzelnes Element einer einfach verketteten Liste repräsentiert.

3. (7 Punkte) Verwenden Sie obige Klassen, um eine Klasse `KundenListe` zu implementieren. Diese soll folgende Operationen bereit stellen:

   - eine Methode `void insert(Kunde k)`, welche einen Kunden an der durch `kleiner` bestimmten Stelle in die Kundenliste einfügt.

     Falls es diesen Kunden bereits gibt, soll eine `Exception(String message)` ausgelöst werden.

   - eine Methode `void addBonus(long bonus)`, welche allen Kunden mit Guthaben, d.h. positivem Kontostand, als Ostergeschenk einen Bonus zahlt, d.h. deren Kontostand entsprechend erhöht.

• AUFGABE 6 (10 Punkte) Vererbung.

In dieser Aufgabe sollen Sie einige Operationen (Quadratwurzel und Multiplikation) auf reellen bzw. komplexen Zahlen implementieren.

Komplexe Zahlen werden dabei in Polarkoordinaten $z = re^{i\phi}$ mit ihrem Absolutbetrag r ($r \geq 0$) und dem Winkel ϕ ($0 \leq \phi < 2\pi$) dargestellt. Dann gelten für komplexe Zahlen $z_i = r_i e^{i\phi_i} \triangleq (r_i, \phi_i)$ die Beziehungen

$$z_1 * z_2 = (r_1 * r_2) * e^{i*(\phi_1+\phi_2)} \triangleq (r_1 r_2, \phi_1 + \phi_2)$$

sowie

$$\sqrt{z} = \sqrt{r} e^{i\frac{\phi}{2}} \triangleq (\sqrt{r}, \frac{\phi}{2})$$

Bei reellen Zahlen ist der Winkel ϕ Null oder π und aus Vereinfachungsgründen wird hier nur der eigentliche reelle Wert (r bei $\pi = 0$ oder $-r$ bei $\phi = \pi$) gespeichert. Die Multiplikation einer reellen Zahl mit einer komplexen Zahl ergibt deshalb als Spezialfall

$$r_1 * z_2 = r_1 * r_2 * e^{i*\phi_2} \triangleq (r_1 r_2, \phi_2)$$

Die Wurzel aus einer negativen, reellen Zahl liefert eine komplexe Zahl ($r \geq 0$):

$$\sqrt{-r} = r * e^{i\pi} \triangleq (r, \pi)$$

Hinweis: In `Math` sind für reelle Zahlen `Math.sqrt` und `Math.PI` bereits vordefiniert.

Diese Unterscheidung wird mit einer abstrakten Klasse `Number` realisiert, deren Objekte entweder reelle Zahlen `Real` oder komplexe Zahlen `Complex` sind.

1. (1 Punkt) Schreiben Sie eine abstrakte Klasse `Number`, welche die Operationen
   - `Number sqrt()` zum Bilden der Quadratwurzel einer Zahl
   - `Number mult(Number other)` zum Multiplizieren der Zahl mit einer anderen

   deklariert, aber noch nicht implementiert.

2. (3 Punkte) Schreiben Sie eine Klasse `Real`, die von der Klasse `Number` abgeleitet ist. Sie soll als Attribut den Wert der Zahl als `double` enthalten, welcher in einem geeigneten Konstruktor gesetzt wird.

   Außerdem sollen die Operationen (`sqrt` und `mult`) aus `Number` implementiert werden.

   Beachten Sie, daß beim Wurzelziehen gegebenfalls eine komplexe Zahl (vom Typ `Complex`, siehe nächste Teilaufgabe) entstehen kann, wenn die Wurzel aus einer negativen Zahl gezogen wird. Da beim Multiplizieren das Argument vom Typ `Number` ist, kann es sowohl eine reelle als auch eine komplexe Zahl sein, wobei im letzteren Fall das Ergebnis dann ebenfalls eine komplexe Zahl ist.

   Beachten Sie ferner, daß beim Wurzelziehen und Multiplizieren die alten Zahlen erhalten bleiben und eine neue Zahl als Ergebnis erzeugt wird.

3. (3 Punkte) Schreiben Sie ferner eine Klasse `Complex`, die ebenfalls von `Number` abgeleitet ist und deren Methoden implementiert.

   `Complex` soll komplexe Zahlen in Polarkoordinaten repräsentieren und enthält deshalb als Attribute den Absolutbetrag und den Winkel als `double`-Werte, welche ebenfalls per Konstruktor gesetzt werden sollen.

4. (2 Punkte) Erweitern Sie die Klasse `Number` um eine Operation `Number average(Number[] N)`, welche die Wurzel $\sqrt{N_0 * \ldots * N_{n-1}}$ aus dem Produkt aller Zahlen im Feld `N` bildet und das Ergebnis dieser Operation zurückliefert.

   Beispiele:

N	`avarage(N)`
[Real(4)]	Real(2)
[Real(2), Real(2)]	Real(2)
[Real(1), Real(-1)]	Complex(1, π)
[Complex(4, 1)]	Complex(2, 0.5)
[Complex(4, 1), Complex(1, 0)]	Complex(2, 0.5)
[Real(0.25), Complex(4, 1)]	Complex(1, 0.5)
[Real(1), Complex(4, 1), Real(0.25), Complex(2, 1), Complex(2, -1)]	Complex(2, 0.5)

5. (1 Punkt) Hätte man `Number` statt als abstrakte Klasse auch als Interface deklarieren können und welche Auswirkungen hätte dies gegebenfalls gehabt?