Fakultät Elektrotechnik/Informatik

Klausur Einführung in die Informatik II für Elektrotechniker 22. Februar 2002

Name: .....

Matr.-Nr. .....

Bearbeitungszeit: 120 Minuten

Bewertung
(bitte offenlassen :–)

Aufgabe	Punkte	Erreichte Punkte
1	4	
2	8	
3	5	
4	7	
5	7	
6	13	
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 und anderen elektronischen Hilfsmitteln nicht gestattet ist.

Viel Erfolg!

• AUFGABE 1 (4 Punkte) Theorie.

1. (1 Punkt) *Quicksort* und *Selektionsort* haben beide Aufwand $O(n^2)$. Warum wird man trotzdem eher *Quicksort* verwenden?

2. (2 Punkte) Erläutern Sie die Begriffe *Interface* und *abstrakte Klasse*. Wo liegen deren Gemeinsamkeiten und Unterschiede?

3. (1 Punkt) Was verstehen Sie unter *dynamischem Binden*?

• **AUFGABE 2 (8 Punkte) JAVA.**

1. (2 Punkte) Was sind *static*-Methoden? Welche Vor- und Nachteile haben sie gegenüber normalen Methoden?

2. (1 Punkt) Wozu dient das Schlüsselwort *protected* in Java?

3. (2 Punkte) Wozu dient das Schlüsselwort *final* in Java?

4. (3 Punkte) Wo sind die Fehler im folgenden *JAVA*-Code? Beschreiben Sie die Fehler und geben Sie die Zeilenzahlen des Auftretens an. (Folgefehler werden wie üblich ignoriert.)

```
 1 class Foo extends Strange {
 2     public static int value;
 3
 4     Foo(int value) {
 5         this.value = value;
 6     }
 7
 8     Foo(Bar other) {
 9         this.value = other.getValue();
10     }
11 }
12
13 interface Bar{
14     public boolean greaterThan(Foo other);
15     public int getValue();
16 }
17
18 class Strange extends Foo {
19     private int val;
20     int counter = 0;
21
22     void veryStrange(String counter) {
23         Foo f;
24         Bar b = new Bar();
25         if (b.greaterThan(new Foo(-1))) {
26             counter = counter + 1;
27         } else {
28             f.value = b.getValue();
29         }
30     }
31 }
```

• AUFGABE 3 (5 Punkte) Numerik.

Schreiben Sie eine *JAVA*-Methode `double sqrtexp()`, welche die Wurzel aus der Eulerschen Zahl e, d.h. $\sqrt{e}$ berechnet.

Dabei sollen Sie e mit Hilfe der Exponentialreihe

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

approximativ berechnen.

Für die Wurzel verwenden Sie bitte *nicht* die Funktion `Math.sqrt`, sondern approximieren die Wurzel $\sqrt{a}$ aus einer Zahl a mittels Newton-Verfahrens durch die Folge $(x_n)_{n \in \mathcal{N}}$:

$$x_{n+1} = \frac{1}{2} x_n + \frac{a}{2x_n}$$

mit dem Startwert $x_0 = a$.

Beenden Sie jeweils den Approximationsprozeß, wenn eine Genauigkeit von 11 Stellen erreicht ist.

• AUFGABE 4 (7 Punkte) Steuern auf Waren.

In dieser Aufgabe sollen Sie Waren und deren Besteuerung modellieren. Jede Ware besitzt einen Nettopreis, auf den die verschiedenen Steuern aufgeschlagen werden. Bei Standardware wird dieser Nettopreis mit dem Mehrwertsteuersatz multipliziert, um den vom Kunden zu zahlenden Endpreis zu bestimmen. Bei Tabakwaren wird auf den Nettopreis zunächst ein bestimmter Betrag für die Tabaksteuer addiert, bevor auch hier der Mehrwertsteuersatz aufgeschlagen wird.

1. (3 Punkte) Erstellen Sie eine abstrakte Klasse `Ware`, welche eine Methode `getPreis()` zur Bestimmung des Verkaufspreises. d.h. einschließlich der diversen Steuern, besitzt.

   Leiten Sie von dieser Klasse zwei Klassen `Standardware` und `Tabakware` mit den oben beschriebenen Eigenschaften ab. Ihre Klassen sollen jeweils die Methode `getPreis` implementieren sowie geeignete Konstruktoren zur Initialisierung der Attribute bereitstellen. Dabei können Sie für die Preise `double`-Werte verwenden, die Mehrwertsteuer soll als ganze Zahl in Prozent angegeben werden.

2. (2 Punkte) Schreiben Sie nun eine Methode

   ```
   void steuererhoehung(Ware[] lager, double erhoehung)
   ```

   welche die Tabaksteuer bei allen Tabakwaren im Lager um den Betrag `erhoehung` vergrößert.

3. (2 Punkte) Schreiben Sie ferner eine Methode

   ```
   double gesamtSteuer(Ware[] lager)
   ```

   zur Berechnung der anfallenden Steuern bei Verkauf aller Waren im Lager. Dabei ist die Steuer jeweils die Differenz von Verkaufspreis und Nettopreis.

• AUFGABE 5 (7 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()`

Zur Darstellung von Grafikelementen auf dem Bildschirm muß vorher das Layout berechnet werden, um zu ermitteln, auf welcher Bildschirmposition welches Element gezeichnet werden soll. Dazu dient die Klasse `LayoutData`:

```
class LayoutData {
      int breite;       // Breite des Elements
      int start;        // Anfangsposition
      int ende;         // Endposition
}
```

Die einzelnen zu layoutenden Grafikelemente seien in einem Binärbaum abgespeichert. Die Daten in den Blättern repräsentieren die tatsächlich darstellbaren Elemente, die Daten in den inneren Knoten entsprechen einem Container für die im Unterbaum enthaltenen Elemente.

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

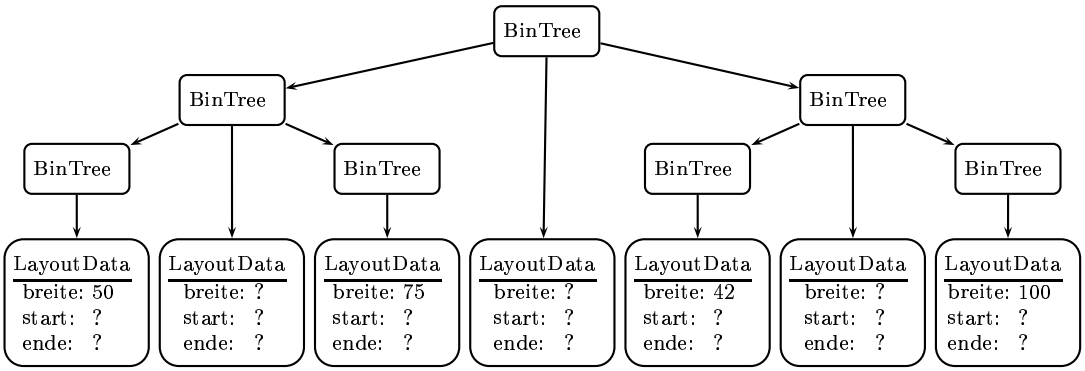
1. (3 Punkte) Eine Methode `int minBreite()` zur Bestimmung der Mindestbreite, wenn alle in den Blättern enthaltenen Layoutelemente ohne Zwischenraum nebeneinander dargestellt werden.

   Dabei stehen Ihnen nur die in den Blättern gespeicherten Breiteninformationen zur Verfügung (`start` und `ende` sind unbelegt.)

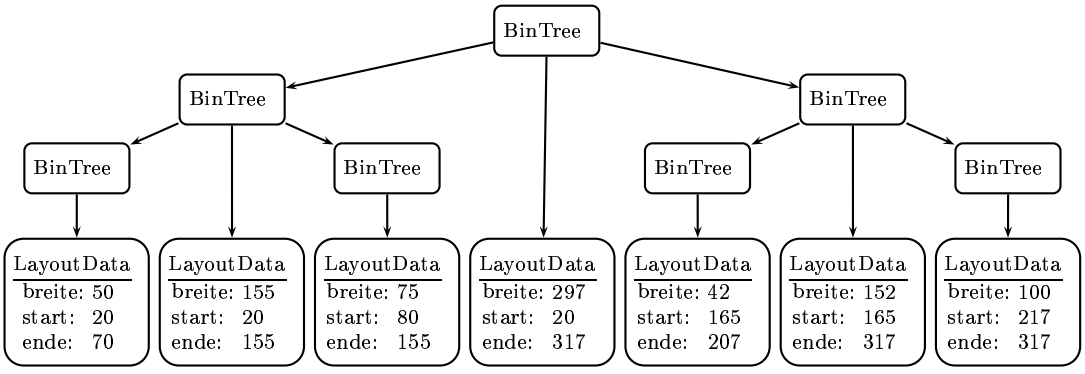
2. (4 Punkte) Schreiben Sie ferner eine Methode `calcLayout(int startPos, int abstand)`, welche die Layoutdaten aller Knoten im Baum berechnet. Dabei gibt `startPos` die Anfangsposition des linkesten Grafikelementes an, `abstand` bezeichnet den Abstand zwischen zwei darstellbaren Grafikelementen.

   Beim Aufruf dieser Methode ist lediglich die Breite der Layoutelemente in den Blättern, die anderen Daten (Anfangs- und Endposition sowie Breite der Containerelemente in den inneren Knoten) jedoch noch `nicht`, diese sollen von Ihnen erst berechnet und in die Baumelemente eingetragen werden.

Für den folgenden Beispielbaum liefert `minBreite()` den Wert $50 + 75 + 42 + 100 = 267$.



Der Aufruf `calcLayout(20, 10)` soll den Baum wie folgt verändern:



• AUFGABE 6 (13 Punkte) Prioritäts-Stack.

In einer Firma sollen Aufträge verwaltet werden die unterschiedlich hohe Bearbeitungsprioritäten haben. Jeder Auftrag besitzt eine eindeutige Auftragsnummer, eine Bezeichnung sowie eine Priorität, die durch eine ganze Zahl ausgedrückt wird. Je höher die Zahl, desto höher die Priorität.

In dieser Aufgabe sollen Sie einen $Prioritäts-Stack$ implementieren. Ein $einfacher Stack$ besteht aus einer Liste von Einträgen, wobei neue Einträge stets nur von vorne eingefügt werden und das Entfernen bzw. Herausholen von Einträgen ebenfalls nur von vorne geschehen darf. Diese Eigenschaften sollen etwas gelockert werden, um den Einträgen je nach der ihnen zugeordneten *Priorität* eine bestimmte Position in der Liste zu geben. Höher priorisierte Aufträge stehen weiter vorne im Stack als niedrigere. Dadurch wird beim Entfernen des vordersten Eintrags zur Bearbeitung stets der Auftrag genommen, der die höchste Priorität hat. Es darf also hinter keinem Auftrag, der die Priorität x hat, ein weiterer Eintrag stehen, der eine Priorität y mit $y > x$ besitzt.

1. (2 Punkte) Erstellen Sie eine Klasse `Auftrag`, welche die Daten eines Auftrags enthalten. Die Attribute sollen von außen nicht zugreifbar sein und im Konstruktor gesetzt werden.

   Die Klasse *Auftrag* soll ferner Methoden `getNr`, `getPrio` und `setPrio` zum Lesen bzw. Setzen von Auftragsnummer und Priorität sowie eine geeignete Methode `public String toString()` zur Ausgabe bereitstellen.

2. (1 Punkt) Schreiben Sie ferner eine Klasse `PrioStackCell`, die eine einzelne Zelle des Prioritätsstack darstellt.

3. (10 Punkte) Verwenden Sie obige Klassen, um eine Klasse `PrioStack` zu implementieren. Diese soll folgende Operationen bereit stellen:

   - eine Methode zur Erzeugung eines leeren Prioritäts-Stacks.
   - eine Methode `void push(Auftrag auftr)`, welche einen Auftrag in den Stack an einer der Priorität entsprechende Stelle einfügt.
   - eine Methode `Auftrag pop()`, welche den im ersten Stack-Eintrag enthaltenen Auftrag herausholt und aus den Stack entfernt.
   - eine Methode `void promote(int nr, int incPrio)` welche die Priorität des Auftrags mit der Nummer `nr` um den Wert `incPrio` erhöht bzw. erniedrigt und ggf. den Eintrag im Stack entsprechend verschiebt.

   Definieren Sie einen eigenen Exceptiontyp `PrioStackException`, die im Fehlerfall jeweils ausgelöst werden soll (mit einem entsprechenden Fehlertext als Parameter).