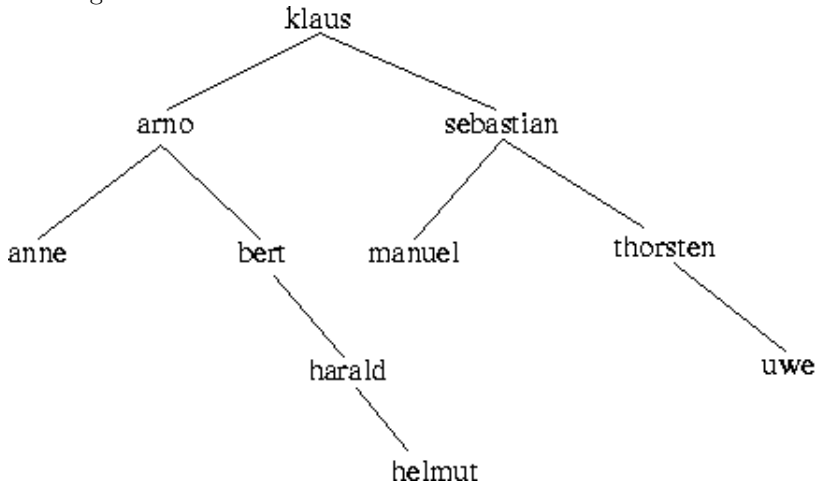
Hauptklausur: PRGII MD

Aufgabe 1 (18 Punkte)
Betrachten Sie folgenden binären Suchbaum.



a) Geben Sie die Knoten des Baumes in *postorder* aus.

Lösung
anne, helmut, harald, bert, arno, manuel, uwe, thorsten, sebastian, klaus.

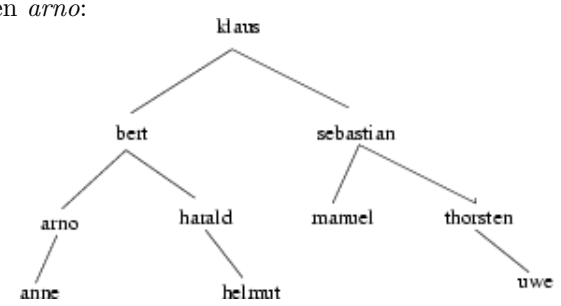
b) Fügen Sie in den Suchbaum den Knoten `"jupp"` als Blatt ein, so daß die Suchbaumeigenschaft nicht verletzt wird.

Lösung
jupp ist rechts unter *helmut* anzuhängen.

c) Ist der Baum balanciert? Wenn nicht balancieren Sie den Baum.

Lösung
Nein. Linkes Kind von *arno* hat Tiefe 1, rechtes Kind Tiefe 3.

Balanciere am Knoten *arno*:



Aufgabe 2 (15 Punkte)
Betrachten Sie das folgende Javamethode, die der Baumklasse `Tree` hinzugefügt wurden. Rechnen Sie die Methoden auf dem Papier für den Baum aus Aufgabe 1 durch und geben ihr Ergebnis an.

```
1  List foo1(){
2    List result = new ArrayList();
3    foo1(1,result);
4    return result;
5  }
6
7  void foo1(int i,List result){
8    if (i%2==0) result.add(mark());
9    for (Iterator it=theChildren().iterator();it.hasNext();){
10     ((Tree)it.next()).foo1(i+1,result);
11   }
12 }
```

Lösung
`foo1` mit:`result = []` und `i=1`
`i%2=1` **if** trifft nicht zu.
for Schleife für die Kinder *klaus:*

- rekursiver Aufruf für *arno:* mit `result = []` und `i=2`
  `i%2=0` **if** trifft zu. Füge *arno* zum Ergebnis hinzu: `result=[arno]`
  for Schleife für die Kinder *arno*:
  - rekursiver Aufruf für *anne* mit `result = [arno]` und `i=3`
    `i%2=1` **if** trifft nicht zu.
  - rekursiver Aufruf für *bert* mit `result = [arno]` und `i=3`
    `i%2=1` **if** trifft nicht zu.
    for Schleife für die Kinder *bert*:
    - rekursiver Aufruf für *harald* mit `result = [arno]` und `i=4`
      `i%2=0` **if** trifft zu. Füge *harald* zum Ergebnis hinzu: `result=[arno,harald]`
      for Schleife für die Kinder *harald*:
      - rekursiver Aufruf für *helmut* mit `result = [arno,harald]` und `i=5`
        `i%2=1`**if** trifft nicht zu.

- rekursiver Aufruf für *sebastian* mit `result = [arno,harald]` und `i=2`
  `i%2=0` **if** trifft zu. Füge *sebastian* zum Ergebnis hinzu:
  `result=[arno,harald,sebastian]`
  for Schleife für die Kinder *sebastian*:
  - rekursiver Aufruf für *manuel* mit `result = [arno,harald,sebastian]` und `i=3`
    `i%2=1` **if** trifft nicht zu.

- rekursiver Aufruf für *thorsten* mit `result = [arno,harald,sebastian]` und `i=3` `i%2=1` **if** trifft nicht zu.
  for Schleife für die Kinder *thorsten*:
  - rekursiver Aufruf für *uwe* mit `result = [arno,harald,sebastian]` und `i=4` `i%2=0` **if** trifft zu. Füge *uwe* zum Ergebnis hinzu:
    `result=[arno,harald,sebastian,uwe]`

Aufgabe 3 (24 Punkte)
Gegeben sei die folgende abstrakte Klasse für Bäume, gemäß unserer Spezifikation aus der Vorlesung:

AbTree.java

```
1  import java.util.*;
2
3  abstract class AbTree{
4    abstract public AbTree
5      tree(Object mark, List/*AbTree*/ xs);
6    abstract public Object mark();
7    abstract public List/*AbTree*/ theChildren();
8  }
```

Schreiben Sie für die Klasse folgende Methoden, die ihr hinzugefügt werden können:

a) `List longNames()`: ergibt eine Liste aller Knotenmarkierungen, deren Stringdarstellung mehr als 10 Buchstaben hat. Lösen Sie diese Aufgabe mit einer akkumulativen Hilfsmethode `List longNames(List result)`.

Lösung

```
1  List longNames(){return longNames(new ArrayList());}
2
3  List longNames(List result){
4    if (mark().toString().length()>10) result.add(mark());
5
6    for (Iterator it=theChildren().iterator(); it.hasNext();){
7       Tree n = (Tree)it.next();
8       n.longNames(result);
9    }
10   return result;
11 }
```

b) `int howManyLeaves(Object o)`: zählt, wieviel Blätter der Baum enthält. (Achtung: die Methode `leaves` aus der Übung steht Ihnen nicht zur Verfügung.)

Lösung
Der Parameter wird in der Methode nicht berücksichtigt.

```
int howManyLeaves(Object o){
  if (theChildren()==0) return 1;
  int result= 0;
  for (Iterator it=theChildren().iterator(); it.hasNext();){
     Tree n = (Tree)it.next();
     result= result + n.howManyLeaves(o);
  }
  return result;
}
```

Aufgabe 4 (23 Punkte)
Gegeben seien folgende Gui-Komponenten:

Counter.java

```
import javax.swing.*;
import java.awt.event.*;

public class Counter extends JFrame{
  IntBox v = new IntBox();

  JButton knopf = new JButton("+1");
  JLabel  label = new JLabel(v.value+"  ");
  JPanel  panel = new JPanel();

  public Counter(){
    super("Ein einfacher Zähler");

    knopf.addActionListener(new Plus1Listener(v,label));

    panel.add(knopf);
    panel.add(label);
    getContentPane().add(panel);
    pack();
    setVisible(true);
  }

  public static void main(String [] _){new Counter();}
}
```

IntBox.java

```
public class IntBox{int value = 0;}
```

Plus1Listener.java

```
import javax.swing.*;
import java.awt.event.*;

```

```
4  class Plus1Listener implements ActionListener {
5    JLabel label;
6    IntBox v;
7
8    public Plus1Listener(IntBox vp,JLabel l){
9      label=l;v=vp;
10   }
11
12   public void actionPerformed(ActionEvent _){
13     v.value=v.value+1;
14     label.setText(v.value+"");
15     label.repaint();
16   }
17 }
```

a) Erweitern Sie die Komponente `Counter` um einen weiteren Knopf, mit dem der Zähler wieder auf `0` zurückgesetzt werden kann.

Lösung

Klasse für den Resetknopf:

```
1  import javax.swing.*;
2  import java.awt.event.*;
3
4  class ResetListener implements ActionListener {
5    JLabel label;
6    IntBox v;
7
8    public Reset(IntBox vp,JLabel l){
9      label=l;v=vp;
10   }
11
12   public void actionPerformed(ActionEvent _){
13     v.value=0;
14     label.setText(v.value+"");
15     label.repaint();
16   }
17 }
```

Füge in `Counter` folgendes Feld ein:

```
1    JButton resetKnopf = new JButton("reset");
```

Setze dessen Listener:

```
1  resetKnopf.addActionListener(new ResetListener(v,label));
```

Und füge den Knopf der Gui-Komponente zu:

```
1  panel.add(resetKnopf);
```

b) Erweitern Sie die Komponente `Counter` so, daß beim Schließen des Fensters die Ausgabe `Auf Wiedersehen` auf der Kommandozeile getätigt und das Programm beendet wirtd.

Lösung

Füge im Konstruktor einen Fensterlistener zu:

```
1   addWindowListener(
2     //Ein Objekt einer anonymen FensterListener Klasse
3     new WindowAdapter(){
4       //in dem auf das Schließen des Fensters mit einem
5       //Programmabbruch reagiert wird
6       public void windowClosing(WindowEvent e) {
7         System.out.println("AufWiedersehen");
8         System.exit(0);
9       }
10  });
```

Aufgabe 5 (20 Punkte)
Gegeben sei die folgende Grammatik (wobei Zahl für ein Token, das eine beliebige Zahl darstellt, steht):

start ::= *aufgabenliste gesamt*
aufgabenliste ::= *aufgabenliste aufgabe*|*aufgabe*
aufgabe ::= Aufgabe Zahl: Zahl Punkte
gesamt ::= insgesamt Zahl Punkte

a) Leiten Sie folgenden Satz mit dieser Grammatik ab:

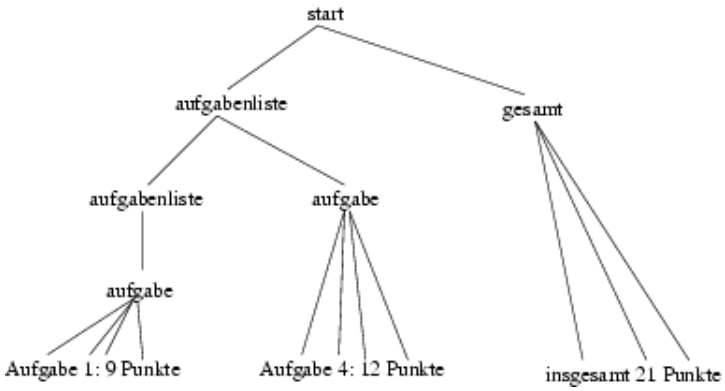
Aufgabe 1: 9 Punkte Aufgabe 4: 12 Punkte insgesamt 21 Punkte

Lösung
start
→*aufgabenliste gesamt*
→*aufgabenliste aufgabe gesamt*
→*aufgabe aufgabe gesamt*

→Aufgabe 1: 9 Punkte *aufgabe gesamt*
→Aufgabe 1: 9 Punkte Aufgabe 4: 12 Punkte *gesamt*
→Aufgabe 1: 9 Punkte Aufgabe 4: 12 Punkte insgesamt 21 Punkte

b) Zeichnen Sie den Ableitungsbaum für Ihre obige Ableitung.

Lösung



c) Transformieren Sie die Grammatik in eine Grammatik, die die gleiche Sprache erzeugt, aber dabei nicht linksrekursiv ist.

Lösung

Eleminierung der Linksrekursion in Regel *aufgabenliste*:

start ::= *aufgabenliste gesamt*
aufgabenliste ::= *aufgabe weitereAufgaben*
weitereAufgaben ::= *aufgabe weitereAufgaben*| ϵ
aufgabe ::= Aufgabe Zahl: Zahl Punkte
gesamt ::= insgesamt Zahl Punkte

Benotung

Aufgabe:	1	2	3	4	5		Summe		Übungen	Gesamt
Punkte:	18	15	24	23	20		100		10	
erreicht:										