# Übungsblatt 2

## (25. Oktober 2010)

**Aufgabe 1** Auf diesem Blatt soll eine rekursive Baumstruktur bearbeitet werden. Gegeben sei die generische Baumklasse `B`, wie sie in der Vorlesung entwickelt wurde:

B.java

```
1  import java.io.IOException;
2  import java.io.Writer;
3  import java.util.*;
4
5  public class B<A> {
6    A element;
7    List<B<A>> kinder;
8    public B(A element, List<B<A>> kinder) {
9      super();
10     this.element = element;
11     this.kinder = kinder;
12   }
13
14   int count(){
15     int result=1;
16     for (B<A> kid:kinder){
17       result=result+kid.count();
18     }
19     return result;
20   }
21   void writeAsXML(Writer schreiber) throws IOException{
22     schreiber.write("<knoten>");
23     schreiber.write("<element>");
24     schreiber.write(""+element);
25     schreiber.write("</element>");
26     schreiber.write("<kinder>");
27     for (B<A> kid:kinder){kid.writeAsXML(schreiber);}
28     schreiber.write("</kinder>");
29     schreiber.write("</knoten>");
30   }
31
32   public static void main(String[] args) {
33     //Übung
34   }
```

Schreiben Sie folgende Methoden für die Klasse:

**a)** `public boolean contains(A a);`
Genau dann wahr, wenn es ein Knotenelement gleich dem Parameter `a` gibt.

**b)** `void postOrder(List<A> result);`
Die Baumelemente sollen in Postorder in die Parameterliste eingefügt werden.

**c)** `void preOrder(List<A> result);`
Die Baumelemente sollen in Präorder in die Parameterliste eingefügt werden.

**d)** `int maxDepth();`
Die maximale Pfadtiefe des Baums soll zurückgegeben werden.

**e)** Gegeben Sei zusätzlich die folgende Schnittstelle:

Function.java
```
1 interface Function<A,R>{
2   R apply(A arg);
3 }
```

Schreiben Sie für die Klasse `B` folgende Methode:
`<R> B<R> map(Function<A,R> f);`

Sie soll einen neuen Baum erzeugen, dessen Struktur der des Ausgangsbaums gleicht, dessen Elemente entstehen, indem jedes Element durch das Argument `f` transformiert wird.

**Aufgabe 2** Auf diesem Übungsblatt sollen Sie sich mit dem DOM-Api zur XML-Verarbeitung vertraut machen. Zum Testen können Sie für alle Aufgabenteile den XML-Quelltext des Javaskripts (`panitz.name/java/skript.xml`) benutzen.

**a)** Schreiben Sie ein Programm, dass ein XML-Dokument aus einer Datei ausliest, und die Anzahl der Knoten in dem Dokument berechnet.

**b)** Schreiben Sie ein Programm, dass ein XML-Dokument aus einer Datei ausliest, und die Menge der unterschiedlichen Tagnamen in diesem Dokument berechnet.

**c)** Schreiben Sie ein Programm, dass ein XML-Dokument aus einer Datei ausliest, und alle Textknoten dieses Dokuments in eine Datei schreibt.

**d)** Schreiben Sie ein Programm, dass für XML-Dokument testet, ob ein bestimmter Tagname in diesem Dokument benutzt wird.

**e)** Schreiben Sie ein Javaprogramm, das zwei Kommandozeilenparameter hat. Der erste Parameter bezeichnet eine XML-Datei, der zweite einen Klassennamen. In der XML-Datei gibt es Elemente mit dem Tagnamen `<code>`. Dieses Element hat das Attribut `class`. Ihr Programm soll eine Datei erzeugen, in der komplette Text steht, der innerhalb von Code-Tags zu finden ist, deren Attribut `class` den Wert des als zweiten Kommandozeilenparameter übergebenen Klassennamen hat.

**Aufgabe 3** Wiederholen Sie die letzte Aufgaben, jetzt aber unter Verwendung des SAX-API.

**Aufgabe 4** Wiederholen Sie die letzte Aufgaben, jetzt aber unter Verwendung des Stax-API.