

**Achtung:** Nicht alle Aufgaben auf diesem Blatt können während des Workshops bearbeitet werden. Es ist sinnvoll, von beiden Teilgebieten jeweils den ersten Teil im Workshop und den Rest zuhause zur Wiederholung zu bearbeiten. Musterlösungen werden später auf der Internetseite der Progwerkstatt veröffentlicht.

Programmieraufgaben sind mit einem Sternchen versehen. Die Teilaufgaben sind außerdem nach Schwierigkeitsgrad sortiert.

## Aufgabe 1 - Listen

Für die folgende Aufgabe benötigst du die Datei *Liste.zip* von der Progwerkstatt-Homepage, sie enthält alle benötigten Klassen und Aufgabendateien.

### \*a)

Öffne die Klassen *Liste* und *ListenElement* und versuche nachzuvollziehen, was diese Klassen tun. Öffne außerdem die Datei *ListenTestProgramm.java* und führe sie aus.

In der Klasse *Liste* befindet sich eine Methode *gebeListeAus()*, die den Inhalt der Liste ausgeben soll, aber noch nicht funktioniert. Vervollständige nun diese Methode. Hierbei kannst du dich iterativ durch die Liste bewegen.

### \*b)

Auch eine Methode *listenLaenge()*, die die Anzahl der in der Liste enthaltenen Elemente aufzählen soll, befindet sich zwar schon im Quelltext, gibt jedoch bisher immer 0 zurück. Diese Methode sollst du nun vollständig implementieren, allerdings sollst du hierbei rekursiv arbeiten.

**Hinweis:** Hierfür benötigst du eine weitere Hilfsmethode.

### c)

Java bietet bereits verschiedene vorgefertigte Datenstrukturen, unter anderem auch Listen. *ArrayList* und *LinkedList* sind zwei solcher vorgefertigter Listen. Informiere dich im Internet über diese zwei Listenklassen und untersuche dabei insbesondere Unterschiede beider Implementierungen und ihre jeweiligen Einsatzzwecke.

**Tipp:** Die Java-API Dokumentation ist eine ausgezeichnete Quelle für solche Informationen! Du findest sie auf der Java-Homepage [java.sun.com](http://java.sun.com)

### \*d)

Bisher war unsere Listenklasse eine allgemeine Liste, die Listenelemente mit Objekten aufnahm. Angenommen, wir beschränken uns nun auf Integer-Werte als Inhalt der Listenelemente. Die Klasse *Liste* soll nun um eine Funktion erweitert

werden, die die Vielfachen einer übergebenen Zahl aus der Liste entfernt. Die Methodensignatur hierfür soll wie folgt aussehen:

```
public void entferneVielfache(int zahl)
```

Teste deine neue Funktion, indem du in der Main-Methode der Datei *Listen-TestProgramm.java* den Befehl `liste.entferneVielfache(3)`; einfügst und dir die neue Liste ausgeben lässt.

**Tipp:** Diese Funktion lässt sich rekursiv sehr schön implementieren

**Hinweis:** Beachte insbesondere eventuelle Sonderfälle und mögliche Probleme.

**Hinweis zu den Integer-Werten:** Die Klasse `Listenelement` nimmt Objekte vom Typ `Object` auf. Die Integer-Werte der Testmethode wurden über sogenannte Wrapper-Klassen eingetragen, die ein Objekt darstellen, das sich um den primitiven Datentyp „wrappt“.

An den eigentlichen `int`-Wert des `Listenelement` „element“ kommt man mit dem Aufruf:

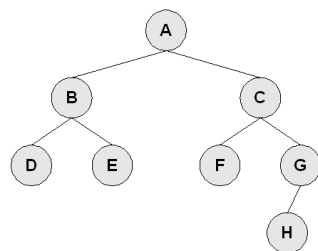
```
Listenelement element = new Listenelement(new Integer(3));  
int wert = ((Integer) element.inhalt).intValue();
```

## Aufgabe 2 - Bäume

Für die folgende Aufgabe benötigst du die Datei *Baum.zip* von der Progwerkstatt-Homepage, sie enthält alle benötigten Klassen und Aufgabendateien.

a)

Gegeben ist folgender Binärbaum. Notiere die Baumknoten jeweils in Pre-, Post- und In-Order-Reihenfolge.



\*b)

Nun soll auch die Klasse `Baum` um eine Ausgabe erweitert werden. Es sollen die Inhalte der Knoten in In-Order-Reihenfolge ausgegeben werden.

**Hinweis:** Da das Durchlaufen rekursiv geschieht, musst du wahrscheinlich neue Methoden anlegen.

**\*c)**

Auch eine Prüfung, ob der Binärbaum vollständig ist, fehlt bisher. Füge sie nun hinzu.

**Hinweis:** Da das Durchlaufen rekursiv geschieht, musst du wahrscheinlich neue Methoden anlegen.

**Tipp:** Wenn du geschickt vorgehst, musst du keine Knoten oder Pfadtiefen zählen.

**\*d)**

Viele Dateibrowser wie der Windows-Explorer stellen Verzeichnisstrukturen in Baumform dar. Implementiere eine ähnliche Ausgabe des Baums für unsere Binärbaumklasse. Als zusätzliche Teilaufgabe kannst du dir Gedanken machen, wie du bei den Blattknoten die Tiefe ausgeben lassen könntest. Setze anschließend deine Überlegungen um.

**Hinweis:** Da das Durchlaufen rekursiv geschieht, musst du wahrscheinlich neue Methoden anlegen.

**\*e)**

Binärbäume eignen sich auch sehr gut dafür, Rechterme zu repräsentieren. Hierbei stellen alle inneren Knoten Rechenoperatoren dar, und alle Blattknoten Zahlenwerte. Zeichne folgende Terme in Baumform auf, um dies zu verdeutlichen.

- $(3 + 5) * 2$
- $(7 + 2) - (6 * (3 - 1))$
- $((5 + 2) * (3 - 2)) * ((1+1) + (2 - (2+2)))$

Anschließend sollst du nun eine entsprechende Baumstruktur selbst implementieren. Hierfür brauchst du verschiedene Knoten-Klassen für die Rechenoperatoren und Zahlenwerte. Außerdem soll ein Term ausgegeben oder berechnet werden können.

**Hinweis:** Wie bei fast allen Funktionen, die auf Bäumen operieren, ist auch hier die Anwendung von Rekursion sehr hilfreich.