Aufgabe 4.1 (Theorie)
Betrachten Sie die unten angegebenen Code-Snippets aus der Klasse EVL<T> (aus der Vorlesung).

Außerdem ist der Code einer statischen Methode erzeugeListe () angegeben, die zB in einer Main-Klasse definiert sein könnte.

Die Klassen Person, Stud, Sportler, Boxer seien wie in Übung 1 definiert.

Zeichnen Sie ein Speicherbild, aus dem erkennbar ist, wie sich durch Ausführung der Methode erzeugeListe () die Liste evl aufbaut.
Also: Zeichnen Sie nicht nur das Endergebnis, sondern machen Sie (entweder durch mehrere Bilder oder durch farbige Markierungen oder durch Nummerierungen) deutlich, welche Speicherzellen wann angelegt und mit welchem Inhalt sie belegt werden, bzw wann sich welche Referenzen verändern.

```
public class EVL<T> {
        private ListenElem first;
        private int size;

        class ListenElem {
                T value;
                ListenElem next;

                ListenElem (T v) {
                        value = v;
                        next = null;
                }
        }
        // ...
        public EVL() {
                this.first = null;
                this.size = 0;
        }
        // ...
        public void insert(T v) {
                ListenElem neu = new ListenElem(v);
                neu.next = first;
                first = neu;
                size++;
        }
        // ...
}

//////////////////////////////////

public static void erzeugeListe() {
        EVL<Person> evl = new EVL<>();
        Stud s  =  new Stud("Susi", 2002);
        Boxer b = new Boxer("Ben", 1990, 178, 85.6);

        evl.insert(s);
        evl.insert(b);
}
```

Aufgabe 4.2
Definieren Sie die Klasse EVLL, die eine „Einfach verkettete Liste mit last-Zeiger“ implementieren soll, so wie diese Variante von EVL in der Vorlesung vorgestellt wurde:

- Ersetzen Sie die Methode T get() durch T getFirst() und ergänzen Sie T getLast().
  Die Methoden geben das erste bzw das letzte Element der Liste zurück. Falls die Liste leer ist, erzeugen sie eine NoSuchElementException.
- Ergänzen Sie **void** append(T v).
  Die Methode fügt ein neues Element mit Eintrag v *hinten* an die Liste an.
- Ersetzen Sie die Methode **void** remove() durch **void** removeFirst().
  Die Methoden entfernt das erste Element der Liste. Falls die Liste (vorher) leer ist, hat die Methode keine Wirkung.
- Ergänzen Sie **void** delete(T v).
  Die Methode entfernt das (erste vorkommende) Element mit Eintrag v, das in der Liste gefunden wird. Falls es kein solches Element gibt, hat die Methode keine Wirkung.
- Testen Sie ausführlich! Sie finden im Downlaod-Bereich des LEA-Kurses JUnit-Tests, die Sie nutzen können.
  Ergänzen Sie weitere Tests!

Aufgabe 4.3
In den vorherigen Übungen haben wir bereits das generische Interface Menge und Implementierungen mittels eines Arrays fester Größe bzw mittels eines dynamischen Arrays kennengelernt:

```
public interface Menge<T> {

        int size();
        boolean isEmpty();
        T get();
        void insert(T e);
        void delete(T e);
        boolean contains(T e);
}
```

a) Implementieren Sie das Interface nun in einer Klasse MengeEVL mittels einer EVL.

b) Prüfen Sie durch JUnit-Tests, ob in MengeEVL das Verhalten von insert korrekt implementiert wurde: Das Hinzufügen von bereits vorhandenen Elementen darf keine Wirkung haben!

c) Testen Sie die Methode merge der Klasse MengeUtil aus Aufgabe 2.3 und das Stud-Modul-Szenario aus Aufgabe 3.4 nun auch mit der neuen Mengenimplementierung. Wenn Sie dort sauber „gegen das Interface“ implementiert haben, brauchen Sie die Klassen Stud bzw Modul nur jeweils an genau einer (!) Stelle und die Methode merge bzw die TestMain zum Stud-Modul-Szenario gar nicht zu ändern!

Aufgabe 4.4 (*)
Definieren Sie eine Datenstruktur DVL<T> („doppelt verkettete Liste“).
Die Listenelemente einer DVL verfügen nicht nur über eine Referenz next auf das *nächste* Listenelement, sondern auch über eine Referenz prev auf das „Vorgänger-Element“.
Für das erste Listenelement ist diese Referenz natürlich gleich **null**.

Die Datenstruktur soll folgende Operationen implementieren:

```
int size()
boolean isEmpty()
boolean contains(T v)
T get(int pos)
T getFirst()
T getLast()
void set(int pos, T v) // aendern eines existierenden Elements
void insert(T v)       // einfuegen vorne
void append(T v)       // einfuegen hinten
void insert(int pos, T v) //  eingefuegtes Element hat dann Positions-Nummer pos
void remove(int pos)  // entfernen des pos-ten Elements
void removeFirst()
void removeLast()
void delete(T v)
```

- Bei Bedarf können weitere private Hilfsmethoden implementiert werden.
- Positionsnummern beginnen bei 0 und enden bei $(size - 1)$.
- Bevor Sie anfangen zu codieren, machen Sie sich durch Speicherbilder (wie in A 4.1) bewußt, *was* Sie eigentlich programmieren wollen!
- Testen Sie ausführlich!