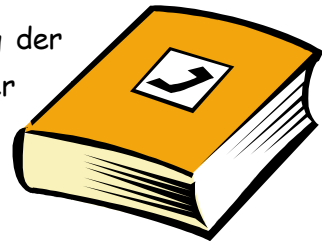
Abstrakte Datenstrukturen

Algorithmen sind ein zentrales Thema der Informatik. Ihre Erforschung und Untersuchung nimmt dort einen bedeutenden Platz ein. Algorithmen operieren nur dann effektiv mit Daten, wenn diese geeignet strukturiert sind.



Schon das Beispiel Telefonbuch zeigt, wie wichtig die Ordnung der Daten nach einem Schema ist. Die Suche nach einer Telefonnummer bei gegebenem Namen gelingt schnell, während die Suche nach einem Namen bei bekannter Telefonnummer ein mühseliges Unterfangen darstellt.

Datenstrukturen und Algorithmen sind also eng miteinander verbunden, und die Wahl der richtigen Datenstruktur entscheidet über effiziente Laufzeiten; beide erfüllen alleine nie ihren Zweck. Leider ist die Wahl der „richtigen" Datenstruktur nicht so einfach, wie es sich anhört, und eine Reihe von schwierigen Problemen in der Informatik sind wohl noch nicht gelöst, da eine passende Datenorganisation bis jetzt nicht gefunden wurde.

Die einfachste und zugleich wichtigste abstrakte Datenstruktur – den **Array** - haben Sie bereits im letzten Schuljahr kennengelernt. Der Vorteil: Sie haben einen direkten Zugriff auf jedes Element und dies ziemlich schnell. Der Nachteil liegt aber darin, dass Sie schon zu Beginn die Größe des Arrays festlegen müssen. Dies kann man mittlerweile zwar durch Kopiermethoden umgehen, doch beim näheren Hinsehen ergeben sich auch weitere Schwierigkeiten bei Operationen auf dem Array. Beispielsweise beim Einfügen oder Löschen eines Elements im Array muss man alle bestehenden Zellen überschreiben bzw. die Inhalte verschieben.

Daher schauen wir uns zum Vergleich einige abstrakte Datentypen an, die sich durch die Struktur vom Array unterscheiden und dadurch andere Vor- und Nachteile aufzeigen.

Lineare Listen

Eine Liste ist eine verkettete Folge von Elementen eines gegebenen Datentyps. Die lineare Liste ist die wichtigste *dynamische Datenstruktur*. Im Unterschied zu Arrays können Listen beliebig wachsen und schrumpfen (neue Elemente werden an die Liste angehängt, nicht mehr benötigte Elemente können entfernt werden.

Verkettete Listen kann man sich wie eine Perlenschnur vorstellen.

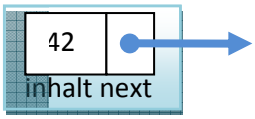
Die Zelle – unsere Perle

```
public class Zelle {

  int inhalt; //Inhalt der Zelle
  Zelle next; //Verweis auf nächste Zelle

  /** Konstruktor
      @param inhalt Inhalt der Zelle
      @param next Verweis auf nächste Zelle
  */
  Zelle(int inhalt, Zelle next){
    this.inhalt = inhalt;
    this.next = next;
  }
}
```

Die Liste besteht aus Zellen. Jede Zelle hat einen Inhalt und einen Verweis auf die nächste Zelle



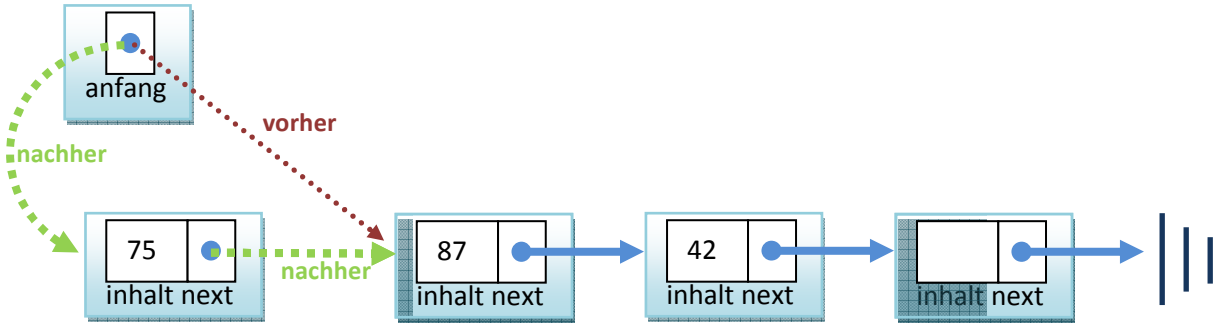
Die Liste – Perlen aufgefädelt

```
public class Liste {
  Zelle anfang;

  public void insert(int n){
    Zelle z = new Zelle(n,anfang);
    anfang = z;
  }
```

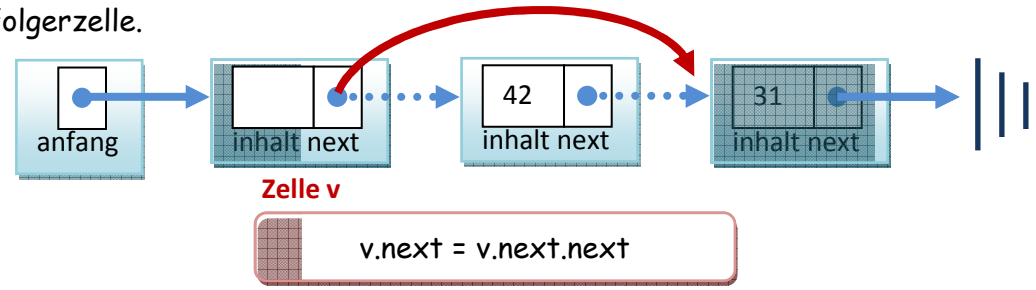
Eine Liste besteht aus einem Link auf die erste Zelle und Methoden, um Elemente …

- einzufügen
- zu entfernen, etc.



Entfernen eines Elements

Um ein Element einer Liste zu entfernen, verbindet man die Vorgängerzelle mit der Nachfolgerzelle.



Das abgeklemmte Element wird irgendwann automatisch von der Java-Speicherbereinigung (garbage collection) gefunden und recycled.

Aufgabe:

Entwickeln Sie eine Lösung, um von einer gegebenen Liste das n-te Element zu entfernen.

Lösung:

```
/** Entfernt das n-te Element der Liste
    @param n Nummer des zu entfernenden Elements
*/
public void removeNth(int n){
  //von der leeren Liste kann nichts entfernt werden
  if(anfang != null){
    //das Element hat keine Vorgängerzelle => gesondert
    if(n==1) anfang = anfang.next;
    else{
      Zelle v = anfang;
      // solange v einen Nachfolger hat und n>1, suche die Zelle
      while(v.next != null && n>1){
        v = v.next;
        n--;
      }
      // falls v einen Nachfolger hat, klemme ab
      if(v.next != null) v.next = v.next.next;
    }
  }
}
```

Übung

Listen und Interfaces

Die beiden Datenstrukturen Liste und Array sollen in dieser Übung vergleichen werden. Erstellen Sie eine Liste mit einem Knoten und eine Klasse mit einem Array und einem Zähler als Attribute. Beide Datenstrukturen sollen vorerst nur int-Werte aufnehmen dürfen.

Jetzt haben Sie zwei unterschiedliche Strukturen, die aber zwecks verschiedener Operationen vergleichbar sein sollen: Also werden die ein Interface implementieren müssen! Dies soll folgendes Aussehen haben:

```
/**
  * Interface für Behälter (Array vs. Liste)
  *
  * @version 1.0 vom 18.09.2011
  * @author H. Frank
  */

public interface Strukturierbar {

  /** Überprüft, ob Behälter leer ist */
  public boolean isEmpty();

  /** Gibt Anzahl der Werte im Behälter an */
  public int length();

  /** Liefert den Inhalt an der i-ten Stelle zurück */
  public int getAtIndex(int i);

  /** Fügt den Wert wert an der Stelle index ein */
  public void insert(int wert, int index);

  /** Fügt wert an der aktuellen Stelle ein */
  public void insert(int wert);

  /** Löscht den Wert an der Stelle index */
  public void delete(int index);

  /** Überprüft, ob wert im Container enthalten ist */
  public boolean contains(int wert);

}
```

Implementieren Sie also die beiden Klassen Array und Liste, die dieses Interface implementieren.