1 Die Klasse ArrayList

1.1 Allgemeine Beschreibung eines ArrayList

Ein Objekt der Klasse ArrayList kann man als ein "dynamisches" Array (Feld) auffassen. Dynamische deswegen, weil das Feld durch zufügen eines Elements größer bzw. durch Entfernen eines Elements kleiner wird. Im Gegensatz zu einem "normalen" Feld kann sich bei einem "dynamisches" Array die Länge des Feldes während des Programmlaufs ändern.
Die ArrayList befindet sich in einem package und muss deswegen importiert werden.

1.2 Verschiedene Möglichkeiten der Verwendung

Es gibt zwei Möglichkeiten, eine ArrayList zu verwenden:

1) In die ArrayList kommen nur Elemente mit dem gleichen Datentyp.

```
ArrayList <Hund> hundeListe = new ArrayList <Hund>();
```

Wichtig: Elementare Datentypen (wie z.B. int) sind als Typparameter **nicht** erlaubt. Z.B:
```
ArrayList <int> zahlenListe = new ArrayList <int>();
```

Der Datentyp in den spitzen Klammern zeigt dem Compiler an, dass nur Elemente dieses Datentyps in die ArrayList aufgenommen werden dürfen.

2) In die ArrayList dürfen auch Elemente mit verschiedenen Datentypen.

```
ArrayList gemischteListe = new ArrayList();
```

Da ein Datentyp in den Klammern fehlt, dürfen Elemente mit verschiedenen Datentypen in die ArrayList aufgenommen werden.

1.3 Einige Methoden der Klasse ArrayList

1) Anfügen eines Elements (an das Ende von ArrayList)
```
public boolean add(Object e)
```

2) Entfernen eines Elements an der Stelle (Index) i
```
public Object remove (int index)
```

3) Testet ob das Element in ArrayList vorkommt (und liefert dann true zurück)
```
public boolean contains(Object e)
```

4) Testet ob ArrayList keine Elemte enthält (und liefert dann true zurück)
```
public boolean isEmpty()
```

5) Liefert den Index des Objekts zurück (falls das Objekt in ArrayList vorkommt, oder falls nicht -1)
```
public int indexOf (Object e)
```

6) Liefert die Anzahl der sich aktuell in ArrayList befindlichen Elemente zurück.
```
public int size()
```

7) Liefert das Objekt zurück, das sich an dieser Stelle (Index) von ArrayList befindet.
```
public Object get(int index)
```

8) Siehe dazu unbedingt: **Java - Dokumentation** !!

1.3.1 Iteratoren (Beispiel siehe unten)

Zusätzlich zum Index (z.B. dem Index i in einer for-Schleife) gibt es noch so genannte Iteratoren, mit denen man eine Schleife durchwandern kann.
Im Gegensatz zum Index eines Felds kann man mit einem Iterator ein Element einer Liste nicht noch einmal auslesen und nicht auf ein beliebiges Element vorspringen beziehungsweise hin und her springen. Man kann sich nur in eine Richtung um jeweils ein Element vor - bzw. zurückbewegen.
Ein Iterator hat eine Methode, die das nächste Element erfragt (next() bzw. previous()) und eine Methode, die ermittelt, ob es überhaupt ein nächstes Element gibt hasNext() bzw. hasPrevious().
So wandert der Iterator einen Datengeber (in der Regel eine Datenstruktur) Element für Element ab. Die Namen der Operationen unterscheiden sich in den Schnittstellen ein wenig und sind beim Iterator kürzer.

1.4 Ein Beispiel

```
import java.util.*;

public class MainArrayList1 {
  public static void main(String[] args){
    Object obj;
    int i;

    // gemischte Liste
    ArrayList gemischteListe;
    gemischteListe = new ArrayList();
    gemischteListe.add(new Integer(139));
    gemischteListe.add("BKI1");

    for(i=0; i<gemischteListe.size();i++){
      obj = gemischteListe.get(i);
      System.out.println(obj);
    }

    // nicht gemischte Liste
    ArrayList <Hund> hundeListe;
    hundeListe = new ArrayList <Hund>();
    Hund h;
    Hund h1 = new Hund("Bello", 12);
    Hund h2 = new Hund("Waldi", 13);
    hundeListe.add(h1);
    hundeListe.add(h2);

    // 1) Durchlaufen einer Liste ohne Iteratoren
    for(i=0; i<hundeListe.size();i++){
      h = hundeListe.get(i);
      System.out.println(h.getName());
    }

    // 2) Durchlaufen einer Liste mit Iteratoren
    // a) Durchlaufen der Liste von hinten
    // Listenzeiger wird um 1 über das Listenende
    // hinaus gesetzt
    /*
    ListIterator<Hund> it =
                   hundeListe.listIterator(hundeListe.size());
    while(it.hasPrevious()){
      System.out.println(it.previous().getName());
    }
    */
    // b) Durchlaufen der Liste von vorne
    // Listenzeiger wird an den Listenanfang gesetzt
    //(=Position 0)
    ListIterator<Hund> it = hundeListe.listIterator(0);
    while(it.hasNext()){
      System.out.println(it.next().getName());
    }
  }
}
```

```
class Hund{
  private String name;
  private int gewicht;

  public Hund(String pName, int pGewicht){
    name = pName;
    gewicht = pGewicht;
  }

  public String getName(){
    return(name);
  }

  public int getGeicht(){
    return(gewicht);
  }
}
```