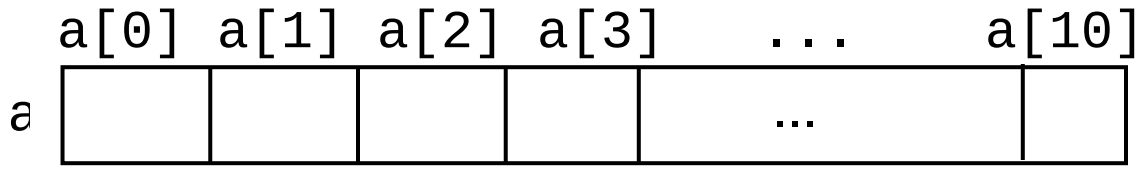
Java Einführung
Collections

Inhalt dieser Einheit

Behälterklassen, die in der Java API bereitgestellt werden

- Wiederholung Array
- Collections (Vector, List, Set)
- Map

Wiederholung Array



- Ein Array hat einen **Namen** (hier: a)
- Die einzelnen Elemente können durch den **Index** (`[0],[1],...`) angesprochen werden
- Die einzelnen Elemente verhalten sich wie namenlose Variablen
- Die Größe wird einmal festgelegt und ist dann fix.

java.util.Arrays

Die Klasse `java.util.Arrays` stellt Funktionalitäten für das Arbeiten mit Arrays zur Verfügung. Ein Auszug:

- `boolean Arrays.`**`equals`**`(Object[] a, Object[] a2)`
  Liefert `true`, wenn die Arrays a und a2 identisch sind. Die Objekte im Array müssen mittels `equals()` vergleichbar sein!

- `void Arrays.`**`fill`**`(Object[] a, Object val)`
  Weist allen Elementen des Array a das Objekt `val` zu.

- `void Arrays.`**`sort`**`(Object[] a)`
  Sortiert das Array `a` in aufsteigender Reihenfolge. Alle Elementen müssen das `Comparable` Interface implementiert haben!

- `int Arrays.`**`binarySearch`**`(Object[] a, Object key)`
  Sucht im sortierten (durch `sort(...)`) Array a nach dem Element `key` unter Verwendung der binären Suchmethode und liefert den Index des gefundenen Objektes bzw. einen negativen Wert bei Misserfolg.

Beispiel: Arrays

```
import java.util.Arrays;

class ArrayDemo {
  public static void main (String [] args) {

  int [] myArray = {9, 23, 1, -28, 8};

  Arrays.sort(myArray);

  for (int i = 0; i < myArray.length;i++){
   System.out.println(myArray[i]);
  }
}
```

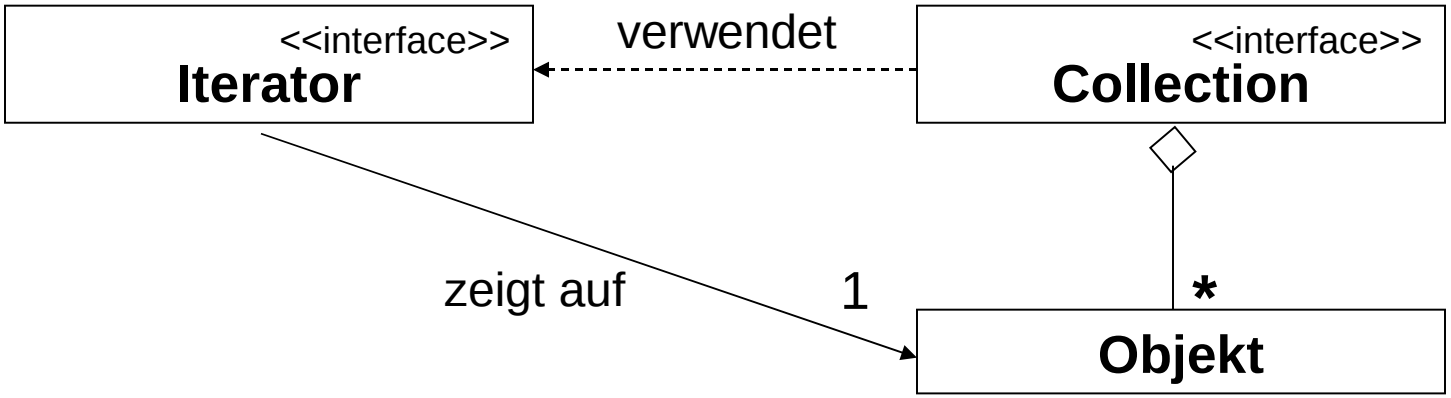
Collections

- Collections bieten ein Konzept, welches Objekte zu Gruppen ***zusammen fasst*** (z.B. eine Klasse (Gruppe von Studenten), ein Postfach (Gruppe von Emails) oder ein Telefonverzeichnis (Gruppe von Name-Telefonnummer- Paaren).

- Mit einem Collection-Objekt können Instanzen ***beliebiger Klassen*** verwaltet werden.

- Seit J2SE 5.0 gibt es ***Generics***, davor galt:
  - Spezielle Typinformation der einzelnen Objekte ***geht dabei verloren***, da sie als ***`Object`*** gespeichert werden.
  - Um die Instanzen verwenden zu können, müssen auf den entsprechenden Datentyp ***umgewandelt*** werden (Casting).

Struktur von Collections



Interface von Collections

- `boolean` **`add`**`(E o)`
  Fügt das Objekt o hinzu und liefert `true` bzw. `false`, je nachdem, ob das Objekt erfolgreich hinzugefügt werden konnte.
- `void` **`clear`**`()`
  Löscht alle Elemente im Container.
- `boolean` **`contains`**`(E o)`
  Liefert `true`, wenn das Objekt o im Container enthalten ist.
- `boolean` **`remove`**`(E o)`
  Liefert `true`, wenn das Objekt o im Container gelöscht werden konnte.
- `int` **`size`**`()`
  Liefert die Anzahl der Elemente.

- `Iterator` **`iterator`**`()`
  Liefert ein Iterator Objekt zum Zugriff auf die Elemente des Containers.

E … Generic: Klasse für die die Collection erzeugt wurde.

Interface von Iterator

- `boolean` **`hasNext`**`()`
  Liefert `true`, wenn es weitere Elemente im Container gibt - bei `false` liefert der Aufruf von `next` eine `Exception`.
- `Object` **`next`**`()`
  Liefert das nächste Element.
- `void` **`remove`**`()`
  Löscht das aktuelle Element der Collection.

Arten von Collections

- **Set** (HashSet, TreeSet,...)
  - kann Elemente nicht doppelt enthalten
  - schnelle Suche

- **List** (ArrayList, LinkedList, Vector, Stack,...)
  - kann Elemente doppelt enthalten
  - Elemente haben eine Reihenfolge
  - variable Länge, schnelles Einfügen und Löschen

Verwendung von List

```
import java.util.*;
class PointList {
  public static void main (String [] args) {
    List<Point> myPoints = new LinkedList<Point>();
    myPoints.add(new Point(1.0,1.0));
    System.out.println("Die Liste enthält "
         + myPoints.size()+" Punkt(e)!\n");

    Iterator<Point> it = myPoints.iterator();
    while(it.hasNext()) {
      Point b = it.next();
   System.out.println("Point("+b.x+","+b.y+") ");
    }
}}
class Point { double x,y;
       Point(double xi, double yi) { x=xi; y=yi; }}
```

Verwendung von Vector

```
import java.util.*;
class PointVector {
  public static void main (String [] args) {
    Vector myPoints = new Vector<Point>();
    myPoints.add(new Point(1.0,1.0));
    System.out.println("Die Liste enthält "
         + myPoints.size()+" Punkt(e)!\n");

    for( int i=0; i<myPoints.size(); i++) {
       Point b = myPoints.elementAt(i);
   System.out.println("Point("+b.x+","+b.y+") ");
    }
}}
class Point { double x,y;
       Point(double xi, double yi) { x=xi; y=yi; }}
```

Sonderfall: Map

- **Map** (Hashtable, TreeMap,...)
  - schnelles Auffinden von Elementenen über einen key
  - jedes Element muss einen eindeutigen key haben

- Map hat ein anderes Interface als Collections:

```
Object put(K key, Object V value);
Object get(K key);
Object remove(K key);
int size();
boolean isEmpty();
void putAll(Map t);
void clear();
public Set keySet(); /* dadurch kann durch die Elemente wie
   durch ein Set iteriert werden! */
```

K, V … Generics

Verwendung von Maps

```
import java.util.*;

class PointMap {
  public static void main (String [] args) {

    Map<String, Point> myPoints =
                new Hashtable<String, Point>();

    myPoints.put("BUBU", new Point(1.0,1.0));

     // 1. einen Iterator auf das keySet erzeugen
    Iterator<String> it = myPoints.keySet().iterator();

    // 2. Keys durchgehen und Elemente aus der Map holen
    while(it.hasNext()) {
      String aKey = it.next();
      Point b = myPoints.get(aKey);
      System.out.println("Key: "+aKey +" Value: "+b);
    }
} }
```

Verwendung von Maps II

```
import java.util.*;

class PointMap {
  public static void main (String [] args) {

    Map<String, Point> myPoints =
              new Hashtable<String, Point>();

    myPoints.put("BUBU", new Point(1.0,1.0));

    for(String aKey : myPoints.keySet())) {  // for-each loop
      Point b = myPoints.get(aKey);
      System.out.println("Key: "+aKey +" Value: "+b);
    }
} }
```

Was Sie nach dieser Einheit wissen sollten...

- Die Verwendung der Hilfsmethoden von `java.util.Arrays`.
- Die Interfaces von Collection und Interator.
- Die Unterschiede von Lists, Sets und Maps und wie diese eingesetzt werden.