Assoziationen

- Unidirektional
  z. B. Verwaltungsklasse – Objekte
- Bidirektional
  z. B. Spieler in einem Raum
- Multiplizität
  - 1:n unidirektional: Array/Collection
  - 1:1 bidirektional: Beispiele (extern, intern)

Beispiel 1:1 bidirektional (extern)

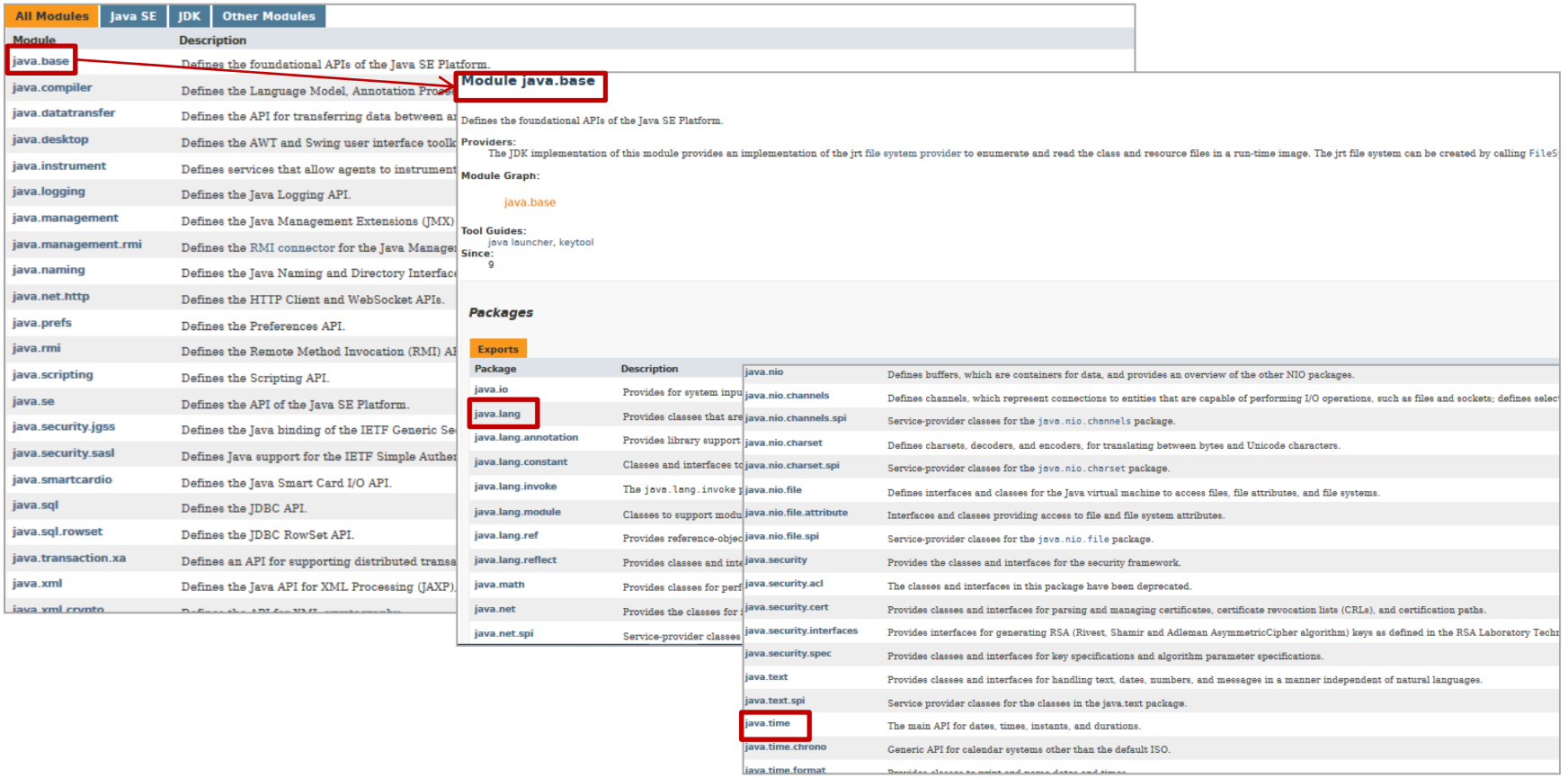
```
public class Raum {
    Spieler sp;
    public void setSp(Spieler sp) {
          this.sp = sp;
    }
}
public class Spieler {
    Raum r;
     public Spieler(Raum r) {
        this.r = r;
     }
}
public class Spielverwaltung {
    public static void main(String[] args) {
          Raum r = new Raum();
          Spieler sp = new Spieler(r); // Spieler erhält Referenz auf Raum
          r.setSp(sp); // Raum erhält Referenz auf Spieler
    }
}
```

Beispiel 1:1 bidirektional (intern)

```
public class Raum {
    Spieler sp;
    public Raum() {
          sp = new Spieler(this);
    }
}
public class Spieler {
    Raum r;
    public Spieler(Raum r) {
        this.r = r;
     }
}
public class Spielverwaltung {
    public static void main(String[] args) {
          Raum r = new Raum();
    }
}
```

5.1 Java API

docs.oracle.com/en/java/javase/18/docs/api/index.html



Wichtige Packages

java.base:

- java.lang: Z. B. Exceptions, Wrapper-Klassen für primitive Datentypen, Comparable, Enum, Math, Object, Runnable, String, System, Thread
- java.net: Z. B. HttpURLConnection, InetAddress, ServerSocket, Socket, URL
- java.time: Z. B. Duration, LocalDateTime, DayOfWeek, Month, Year
- java.util: Z. B. ArrayList, Collection, Queue/Deque, EventListener, EventObject, Formatter, HashMap/Set, Iterator, List/Map/Set, Random, Scanner, Stack, TreeMap/Set
- java.security/javax.crypto: Kryptographie

java.sql: JDBC (Datenbank)

java.xml: Verarbeitung von XML

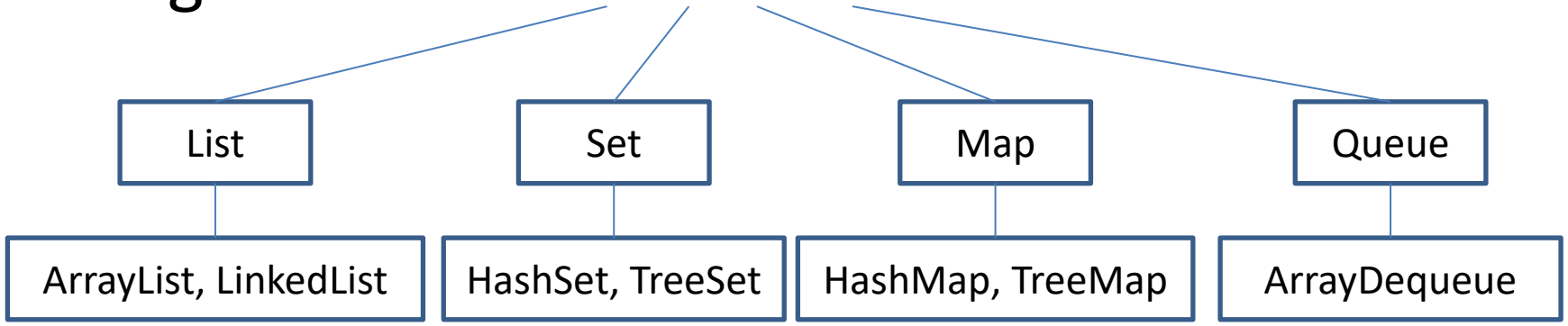
Separate API: www.oracle.com/technical-resources/articles/java/json.html

Collection-Klassen

Probleme von Feldern (arrays)?

- Länge muss im Voraus festliegen
- Entfernen von Elementen
- Keine felderübergreifenden Methoden

Lösung: Collections



(J e w e i l s n u r e i n i g e B e i s p i e l e)

ArrayList

Nur (Wrapper-)Klassen möglich, keine primitiven Datentypen

Einführungsbeispiel:

```
ArrayList<Integer> quadrate = new
  ArrayList<>(); // Diamantoperator von Generics
for (int i = 0; i < 20; i++) {
  quadrate.add(i*i);
}
for (int i = 0; i < quadrate.size(); i++) {
  System.out.println("Quadratzahl von i: " +
  quadrate.get(i));
}
System.out.println("Methode von ArrayList: " +
  quadrate.toString());
```

https://docs.oracle.com/javase/10/docs/api/java/util/ArrayList.html

ArrayList

Speicherplatz effizient nutzen:
capacity, trimToSize, ensureCapacity

```
ArrayList<Integer> zufallszahlen =
  new ArrayList<>(20);
Random rd = new Random();
for (int i = 0; i < 20; i++) {
  zufallszahlen.add(rd.nextInt(6)+1);
}
System.out.println("Zufahlszahlen " +
  zufallszahlen);
quadratzahlen.trimToSize();
```

ArrayList

Mit eigener Klasse

```
public class Veranstaltung {
    private String name;
    private LocalDateTime beginn;
    private LocalDateTime ende;
… }
```

Verwendung

```
ArrayList<Veranstaltung> veranstaltungskalender =
   new ArrayList<>();
```

ArrayList

Weitere Methoden

```
System.out.println("Ist die Zahl 25 enthalten? " +
   quadrate.contains(25));
boolean mehrfach = (quadrate.indexOf(25)>-1) &&
   (quadrate.indexOf(25)!=quadrate.lastIndexOf(25));
System.out.println("Ist 25 mehrfach enthalten? " +
   mehrfach);
System.out.println("Ist die Liste leer? " +
   quadrate.isEmpty());
int index25 = quadrate.indexOf(25);
System.out.println("An welcher Stelle ist 25 enthalten?
   " + index25);
quadrate.remove(index25); // hier alternativ
   ...remove(25);
```

Collections

Methoden auf Collections, z. B.

```
for (int i = 0; i < 6; i++) {
   System.out.println("Häufigkeit von i: " +
   Collections.frequency(zufallszahlen, i));
}
System.out.println("Größte Zahl: " +
   Collections.max(zufallszahlen));
System.out.println("Kleinste Zahl: " +
   Collections.min(zufallszahlen));
System.out.println("Zufahlszahlen " + zufallszahlen);
Collections.sort(zufallszahlen);
System.out.println("Zufahlszahlen sortiert " + zufallszahlen);
Collections.reverse(zufallszahlen);
System.out.println("Zufahlszahlen umgedreht " + zufallszahlen);
```

https://docs.oracle.com/javase/10/docs/api/java/util/Collections.html

Aufgabenblatt 8

Siehe pdf-Datei
Aufgabe 1-3

Für Sortierbarkeit

```
… implements Comparable<PersonBasis>
@Override
public int compareTo(PersonBasis p) {
   if (this.nname.equals(p.nname))
    {
     if (this.vname.equals(p.vname)) {
      if (this.alter==p.alter) {return 0;}
      else { return this.alter-p.alter;   }
     } else {return this.vname.compareTo(p.vname);}
   } else {return this.nname.compareTo(p.nname);}
}
```

HashSet: Beispiel

```
Random rd = new Random();
ArrayList<Integer> zufallszahlenListe = new ArrayList<>(20);
for (int i = 0; i < 20; i++) {
   zufallszahlenListe.add(rd.nextInt(20));}
System.out.println("Zufahlszahlenliste " + zufallszahlenListe);
HashSet<Integer> zufallszahlenMenge = new HashSet<>(20);
for (int i = 0; i < zufallszahlenListe.size(); i++) {
   zufallszahlenMenge.add(zufallszahlenListe.get(i));}
System.out.println("Zufahlszahlenmenge " + zufallszahlenMenge +
   " Größe " + zufallszahlenMenge.size());
```

Weitere Methode: `zufallszahlenMenge.remove(object);`

Ausgabe

Zufahlszahlenliste [1, 13, 10, 11, 17, 14, 14, 4, 1, 14, 2, 10, 9, 9, 0, 11, 0, 13, 12, 0]

Zufahlszahlenmenge [0, 1, 17, 2, 4, 9, 10, 11, 12, 13, 14]

HashSet

- Keine doppelten Elemente
- Problem bei nachträglicher Änderung
- Keine Sequenz → kein Index
- Iterator für Durchlaufen:

```
for (Iterator iterator = zufallszahlenMenge.iterator();
  iterator.hasNext();)
{
  Integer integer = (Integer) iterator.next();
}
for (Integer integer : zufallszahlenMenge) // foreach
{
  System.out.println(integer); // keine Modifikation möglich
}
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashSet.html

TreeSet

- Balancierte Binärbäume
- Sortierung möglich
- Erforderlich:

  Ordnung oder Comparator-/Comparable-Implementierung
- Iteratoren: descendingIterator(), iterator()

Beispiel:

```
TreeSet<Integer> zzSortierteMenge = new TreeSet<>();
for (Integer integer : zufallszahlenListe)
{
   zzSortierteMenge.add(integer); // iterator
}
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/TreeSet.html

Aufgabenblatt 8

Siehe pdf-Datei
Aufgabe 4

HashMap

Prinzip: Schlüssel-/Wert-Paare, FIFO

```
HashMap<String, String> postleitzahlen = new
  HashMap<>();
postleitzahlen.put("70567", "Stuttgart-
  Möhringen");
postleitzahlen.put("70794", "Bernhausen");
postleitzahlen.put("88048", "Friedrichshafen");
System.out.println("Postleitzahlen " +
  postleitzahlen + " Größe " +
  postleitzahlen.size());
System.out.println(postleitzahlen.get("88048"));
postleitzahlen.remove("70794");
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html

HashMap: Iteratoren

Mengen

- keySet mit descendingIterator, iterator
- entrySet mit descendingIterator, iterator

Beispiel

```
// Ausgabe Schlüsselmenge
for (Iterator iterator =
   postleitzahlen.keySet().iterator(); iterator.hasNext();)
   {
System.out.println(iterator.next());
}
// Ausgabe Zuordnungsmenge
for (Iterator iterator =
   postleitzahlen.entrySet().iterator();
   iterator.hasNext();) {
System.out.println(iterator.next());
}
```