

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

All Modules	Java SE	JDK	Other Modules
Module	Description		
java.base	Defines the foundational APIs of the Java SE Platform.		
java.compiler	Defines the Language Model, Annotation Processor, and Compiler APIs.		
java.datatransfer	Defines the API for transferring data between applications.		
java.desktop	Defines the AWT and Swing user interface toolkits.		
java.instrument	Defines services that allow agents to instrument classes.		
java.logging	Defines the Java Logging API.		
java.management	Defines the Java Management Extensions (JMX) API.		
java.management.rmi	Defines the RMI connector for the Java Management Extensions.		
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.		
java.net.http	Defines the HTTP Client and WebSocket APIs.		
java.prefs	Defines the Preferences API.		
java.rmi	Defines the Remote Method Invocation (RMI) API.		
java.scripting	Defines the Scripting API.		
java.se	Defines the API of the Java SE Platform.		
java.security.jgss	Defines the Java binding of the IETF Generic Security Service (GSS) API.		
java.security.sasl	Defines Java support for the IETF Simple Authentication and Security Layer (SASL) API.		
java.smartcardio	Defines the Java Smart Card I/O API.		
java.sql	Defines the JDBC API.		
java.sql.rowset	Defines the JDBC RowSet API.		
java.transaction.xa	Defines an API for supporting distributed transactions.		
java.xml	Defines the Java API for XML Processing (JAXP).		
java.xml.crypto	Defines the Java API for XML Cryptography (JAX-WS).		

Module java.base

Defines the foundational APIs of the Java SE Platform.

Providers:
The JDK implementation of this module provides an implementation of the jrt file system provider to enumerate and read the class and resource files in a run-time image. The jrt file system can be created by calling `FileSystems.newJrtFileSystemProvider()`.

Module Graph:
`java.base`

Tool Guides:
`java launcher`, `keytool`

Since: 9

Packages

Exports

Package	Description
java.io	Provides for system input and output.
java.lang	Provides classes that are fundamental to the Java language.
java.lang.annotation	Provides library support for annotations.
java.lang.constant	Classes and interfaces to support constant pools.
java.lang.invoke	The <code>java.lang.invoke</code> package provides classes and interfaces for invoking methods dynamically.
java.lang.module	Classes to support module systems.
java.lang.ref	Provides reference objects.
java.lang.reflect	Provides classes and interfaces for reflection.
java.math	Provides classes for performing mathematical operations.
java.net	Provides the classes for network access.
java.net.spi	Service-provider classes for the <code>java.net</code> package.
java.nio	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
java.nio.channels	Defines channels, which represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors.
java.nio.channels.spi	Service-provider classes for the <code>java.nio.channels</code> package.
java.nio.charset	Defines charsets, decoders, and encoders, for translating between bytes and Unicode characters.
java.nio.charset.spi	Service-provider classes for the <code>java.nio.charset</code> package.
java.nio.file	Defines interfaces and classes for the Java virtual machine to access files, file attributes, and file systems.
java.nio.file.attribute	Interfaces and classes providing access to file and file system attributes.
java.nio.file.spi	Service-provider classes for the <code>java.nio.file</code> package.
java.security	Provides the classes and interfaces for the security framework.
java.security.aci	The classes and interfaces in this package have been deprecated.
java.security.cert	Provides classes and interfaces for parsing and managing certificates, certificate revocation lists (CRLs), and certification paths.
java.security.interfaces	Provides interfaces for generating RSA (Rivest, Shamir and Adleman AsymmetricCipher algorithm) keys as defined in the RSA Laboratory Technical Report 12-16.
java.security.spec	Provides classes and interfaces for key specifications and algorithm parameter specifications.
java.text	Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.
java.text.spi	Service provider classes for the classes in the <code>java.text</code> package.
java.time	The main API for dates, times, instants, and durations.
java.time.chrono	Generic API for calendar systems other than the default ISO.
java.time.format	Provides classes to parse and generate dates and times.

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

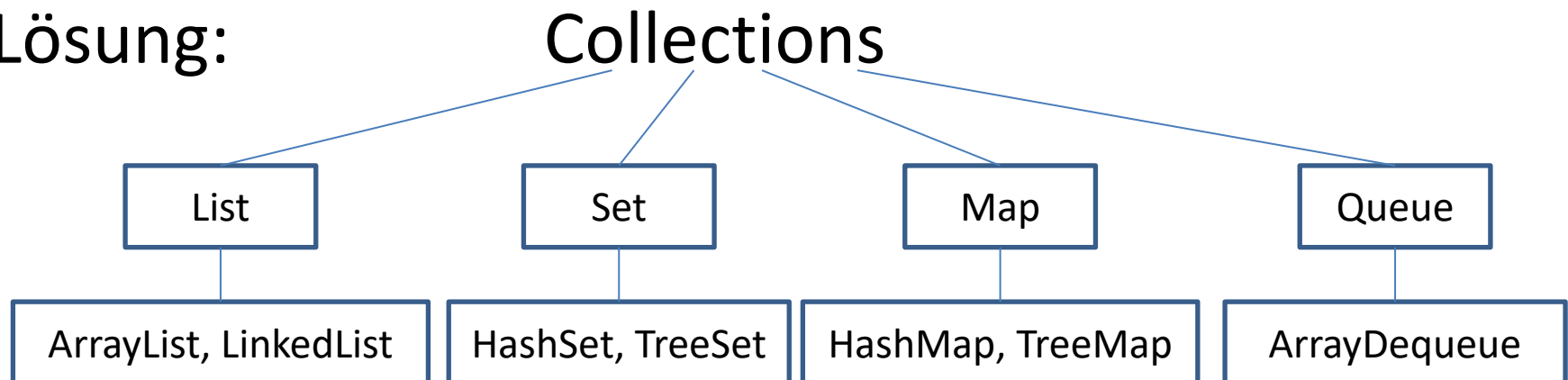
Separate API: www.oracle.com/technical-resources/articles/java/json.html

Collection-Klassen

Probleme von Feldern (arrays)?

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

Lösung:



(Jeweils nur einige Beispiele)

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);  
quadratzen.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());
}
```