

1 Input und Output

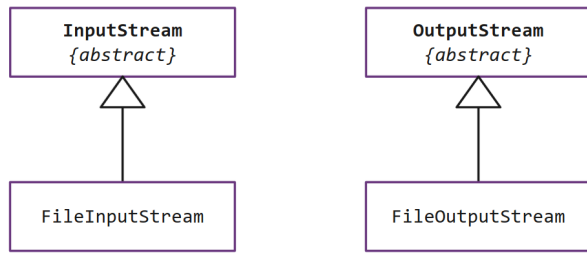


Abbildung 1: Klassenhierarchie von Input und Output

1.1 Input

1.1.1 File-Reader

```
try (var reader = new FileReader("quotes.txt")) {
    int value = reader.read();
    while (value ≥ 0) {
        char c = (char) value;
        // use character
        value = reader.read();
    }
}

new FileReader(f);
// ist äquivalent zu
new InputStreamReader(new FileInputStream(f));
```

1.1.2 Zeilenweises Lesen

```
try (var reader = new BufferedReader(new FileReader("quotes.txt"))) {
    String line = reader.readLine();
    while (line != null) {
        System.out.println(line);
        line = reader.readLine();
    }
}
```

Info: FileReader liest einzelne Zeichen, BufferedReader liest ganze Zeilen.

1.2 Output

1.2.1 File-Writer

```
try (var writer = new FileWriter("test.txt", true)) {
    writer.write("Hello!");
    writer.write("\n");
}
```

1.3 Zusammenfassung

- Byte-Stream: Byteweises Lesen von Dateien
 - ↳ FileInputStream, FileOutputStream
- Character-Stream: Zeichenweises Lesen von Dateien (UTF-8)
 - ↳ FileReader, FileWriter

2 Serialisierung

Das Serializable-Interface implementieren (Marker-Interface). Ohne Marker-Interface wird eine NotSerializableException geworfen. Jedes Feld, das serialisiert werden soll, muss ebenfalls Serializable implementieren (Transitive Serialisierung).

```
class Person implements Serializable {
    private static final long serialVersionUID = 1L;
    private String firstName;
    private String lastName;
```

```
// ...
}
```

Das kann dann vom ObjectOutputStream verwendet werden, um Data Binär zu serialisieren:

```
try (var stream = new ObjectOutputStream(new
FileOutputStream("serial.bin"))) {
    stream.writeObject(person);
}
```

Um ein Objekt aus einem Bytestrom zu deserialisieren, wird der ObjectInputStream verwendet:

```
try (var stream = new ObjectInputStream(
new FileInputStream("serial.bin"))) {
    Person p = (Person) stream.readObject();
    // ...
}
```

2.1 Serialisierung mit Jackson

```
Employee e = new Employee(1, "Frieder Loch");
String jsonString = mapper.writeValueAsString(e);
var writer = new PrintWriter(FILE_PATH);
writer.println(jsonString);
writer.close();
```

Output:

```
{"id":1,"name":"Frieder Loch"}
```

2.1.1 Beeinflussung der Serialisierung

```
public class WeatherData {
    @JsonProperty("temp_celsius")
    private double tempCelsius;
}
```

```
@JsonPropertyOrder({"name", "id"})
public class Employee{
    public int id;
    public String name;
}
```

```
@JsonIgnore, @JsonInclude(Include.NON_NULL)    (nur nicht-null-Werte),
@JsonFormat(pattern = "dd-MM-yyyy")
```

```
@JsonRootName(value="user")
public class Customer {
    public int id;
    public String name;
}
// ...
var mapper = new ObjectMapper().enable(
    SerializationFeature.WRAP_ROOT_VALUE
);

Output:
{
  "user": {
    "id": 1,
    "name": "Frieder Loch"
  }
}
```

2.1.2 JsonGenerator

```
var generator = new JsonFactory().createGenerator(
    new FileOutputStream("employee.json"), JsonEncoding.UTF8);
generator.writeStartObject();
generator.writeFieldName("identity");
generator.writeStartObject();
generator.writeStringField("name", company.name);
generator.writeEndObject();
```

2.1.3 Deserialisierung

```
String json = "{\"name\":\"Max\\\", \"alter\":30}";
ObjectMapper mapper = new ObjectMapper();
Benutzer benutzer = mapper.readValue(json, Benutzer.class); // throws
JsonMappingException
```

Deserializier:

```
public class CompanyJsonDeserializer extends JsonSerializer {
    @Override
    public Company deserialize(JsonParser jP, DeserializationContext dc)
        throws IOException {
        var tree = jP.readValueAs(JsonNode.class);
        var identity = tree.get("identity");
        var url = new URL(tree.get("website").asText());
        var nameString = identity.get("name").asText();
        var uuid = UUID.fromString(identity.get("id").asText());
        return new Company(nameString, url, uuid);
    }
}
```

@JacksonInject:

```
public class Book {
    public String name;
    @JacksonInject
    public LocalDateTime lastUpdate;
}

InjectableValues inject = new InjectableValues.Std()
    .addValue(LocalDateTime.class, LocalDateTime.now());
Book[] books = new ObjectMapper().reader(inject)
    .forType(new TypeReference<Book[]>()).readValue(jsonString);
```

3 Generics

3.1 Iterator

```
for (Iterator<String> it = list.iterator(); it.hasNext(); ) {
    String s = it.next();
    System.out.println(s);
}
```

3.1.1 Iterable und Iterator

```
interface Iterable<T> {
    Iterator<T> iterator();
}

interface Iterator<T> {
    boolean hasNext();
    T next();
}
```

Klassen, die Iterable implementieren, können in einer enhanced for-Schleife verwendet werden:

3.2 Generische Methoden

```
public static <T> Stack<T> multiPush(T value, int times) {
    var result = new Stack<T>();
    for(var i = 0; i < times; i++) {
        result.push(value);
    }
    return result;
}

ausführbarer
ausführbarer
```

Typ wird am Kontext erkannt:

```
Stack<String> stack1 = multiPush("Hallo", 3);
Stack<Double> stack2 = multiPush(3.141, 3);
```

Generics mit Type-Bounds verwenden immer extends, kein implements.

Vorsicht:

```
private static <T extends Comparable<T>> T majority(T x, T y, T z) {
    // ...
}
// ...
Number n = majority(1, 2.4232, 3); // Compilerfehler
Main.<Number>majority(1, 2.4232, 3); // Eigentlich OK, aber Number hat
keine Comparable-Implementierung
```

Erstellung eines Type T geht nicht:

```
T t = new T(); // Compilerfehler
T[] array = (T[]) new Object[10]; // Funktioniert
```

Die JVM hat keine Typinformationen zur Laufzeit → Non-Reifiable Types, Type-Erasing.

- So laufen:
- Alte, nicht generische Programme auf neuen JVMs
 - Neue, generische Programme auf alten JVMs
 - Alter, nicht generischer Code kompiliert mit neuen Compilern

3.3 Unterschied Comparable

```
<T extends Comparable<T>> T max(T x, T y) {
    return x.compareTo("lmaooo") > 0 ? x : y; // Compilerfehler
}

<T extends Comparable> T max(T x, T y) {
    return x.compareTo("lmaooo") > 0 ? x : y; // OK
}
```

3.4 Wildcards

```
public static void printAnimals(List<? extends Animal> animals) {
    for (Animal animal : animals) {
        System.out.println(animal.getName());
    }
}

public static void main(String[] args) {
    List<Animal> animallist = new ArrayList<>();
    printAnimals(animallist);
    List<Cat> catList = new ArrayList<>();
    printAnimals(catList);
}
```

3.5 Variance

	Typ	Kompatible Typ-Argumente	Lesen	Schreiben
Invarianz	C<T>	T	✓	✓
Kovarianz	C<? extends T>	T und Subtypen	✓	✗
Kontravarianz	C<? super T>	T und Basistypen	✗	✓
Bivarianz	C<?>	Alle	✗	✗

3.6 Generics vs ArrayList

```
ArrayList<String> stringsArray = new ArrayList<>();
ArrayList<Object> objectsArray = stringsArray; // Compilerfehler
```

```
String[] stringsArray = new String[10];
Object[] objectsArray = stringsArray; // OK
objectsArray[0] = Integer.valueOf(2); // Exception
```

Kompiliert nicht mit Subtypen:

```
Object[] objectsArray = new Object[10];
String[] stringsArray = objectsArray; // Compilerfehler
```

3.6.1 Kovarianz

```
Stack<? extends Graphic> stack = new Stack<Rectangle>();
stack.push(new Graphic()); // nicht erlaubt
stack.push(new Rectangle()); // auch nicht erlaubt
```

→ Kovariante generische Typen sind **readonly**.

3.6.2 Kontravarianz

```
public static void addToCollection(List<? super Integer> list, Integer i) {
    list.add(i);
}
```

```
List<Object> objects = new ArrayList<>();
addToCollection(objects, 1); // OK
```

Lesen aus Collection mit Kontravarianz ist nicht möglich:

```
Stack<? super Graphic> stack = new Stack<Object>();
stack.add(new Object()); // Nicht OK, Object ist kein Graphic
stack.add(new Circle()); // OK
Graphic g = stack.pop(); // Compilerfehler
```

3.6.3 PECS

> Producer Extends, Consumer Super

```
<T> void move(Stack<? extends T> from, Stack<? super T> to) {
    while (!from.isEmpty()) {
        to.push(from.pop());
    }
}
```

3.6.4 Bivarianz

Schreiben nicht möglich, Lesen mit Einschränkungen:

```
static void appendNewObject(List<?> list) {
    list.add(new Object()); // Compilerfehler
}

public static void printList(List<?> list) {
    for (Object elem: list) {
        System.out.print(elem + " "); // OK
    }
    System.out.println();
}
```

4 Annotations und Reflection

Beispiele für Annotations:

- @Override
- @Deprecated
- @SuppressWarnings(value = "unchecked")
- @FunctionalInterface

4.1 Implementation von Annotations

```
@Target(ElementType.METHOD) // oder TYPE, FIELD, PARAMETER, CONSTRUCTOR
@Retention(RetentionPolicy.RUNTIME) // oder SOURCE, CLASS
public @interface Profile {}
```

4.2 Reflection

```
Class c = "foo".getClass();
Class c = Boolean.class;
```

Wichtige Methoden von Class:

- public Method[] getDeclaredMethods() throws SecurityException
- public Constructor<?>[] getDeclaredConstructors() throws SecurityException
- public Field[] getDeclaredFields() throws SecurityException

4.2.1 Methoden

- public String getName()
- public Object invoke(Object obj, Object... args)

4.2.2 Auswahl annotierter Methoden

```
for (var m : methods) {
    if(m.isAnnotationPresent(Profile.class)) {
        PerformanceAnalyzer.profileMethod(testFunctions, m, new Object[]
{array});
    }
}
```

4.2.3 Aufruf und Profiling der Methoden

```
public class PerformanceAnalyzer {
    public static void profileMethod(Object object, Method method, Object[]
args) {
```

```
long startTime = System.nanoTime();
try {
    method.invoke(object, args);
} catch (IllegalAccessException | InvocationTargetException e) {
    e.printStackTrace();
}
long endTime = System.nanoTime();
long elapsedTime = endTime - startTime;
System.out.println(method.getName() + " took " + elapsedTime + "
nanoseconds to execute.");
}
```

5 Arrays und Listen

5.1 Sortieren

5.1.1 Platz finden und Platz schaffen

Beispiel (Highscore-Liste):

- Iteration vom Ende zu Beginn
- Neuer Score grösser als Score an position - 1?
 - Ja: Kopiere position - 1 an position
 - Nein: Iteration abbrechen
- Eintrag an position speichern

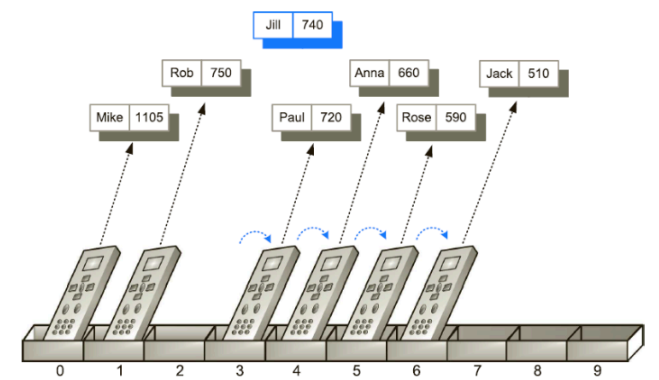
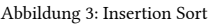


Abbildung 2: Leaderboard

```
public void add(GameEntry entry) {
    int newScore = entry.getScore();
    if(isHighscore(newScore)) {
        if(numEntries < board.length) {
            numEntries++;
        }
        int j = numEntries - 1;
        for(; j > 0 && board[j - 1].getScore() < newScore; j--) {
            board[j] = board[j - 1]
        }
        board[j] = entry;
    }
}
```

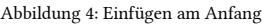
5.1.2 Insertion Sort



5.2 Linked List

5.2.1 Einfügen am Anfang

1. Neuen Knoten mit altem Kopf verketteten
2. head auf neuen Knoten setzen



5.2.2 Einfügen am Ende

1. Neuen Knoten auf null zeigen lassen

2. Früheren Endknoten mit neuem Knoten verketteten
3. tail auf neuen Knoten setzen

5.2.3 Laufzeit Einfügen/Lesen

	Lesen	Einfügen
Array	$O(1)$	$O(n)$
Liste	$O(n)$	$O(n)$

5.3 Doubly Linked List

5.3.1 Einfügen eines Knotens am Anfang

```
public void addFirst(T element) {
    DoublyLinkedListNode<T> newNode = new DoublyLinkedListNode<>(element, null,
null);
    DoublyLinkedListNode<T> f = header.getNext();
    header.setNext(newNode);
    newNode.setNext(f);
    size++;
}
```

5.3.2 Entfernen eines Knotens am Ende

```
public T removeLast() {
    DoublyLinkedListNode<T> oldPrevNode
    = trailer.getPrev();
    DoublyLinkedListNode<T> prevPrevNode
    = oldPrevNode.getPrev();
    trailer.setPrev(prevPrevNode);
    prevPrevNode.setNext(trailer);
    oldPrevNode.setPrev(null);
    oldPrevNode.setNext(null);
    size--;
    return oldPrevNode.getElement();
}
```

6 Algorithmenparadigmen

Definition: Endliches, deterministisches und allgemeines Verfahren unter Verwendung ausführbarer, elementarer Schritte.

6.1 Set-Covering Problem

Beispiel: Alle Staaten mit möglichst wenigen Radiosendern abdecken.

RADIO STATION	AVAILABLE IN
KONE	ID, NV, UT
KTWO	WA, ID, MT
KTHREE	OR, NV, CA
KFOUR	NV, UT
KFIVE	CA, AZ

...etc...

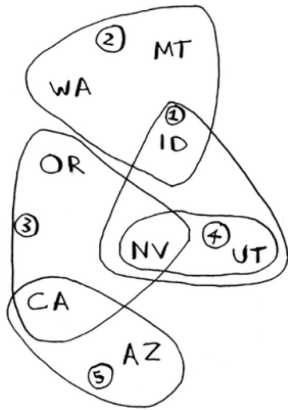


Abbildung 5: Set-Covering Problem

Optimaler Algorithmus:

- Teilmengen der Stationen aufzählen

- Minimale Anzahl Stationen wählen
- Problem: 2^n mögliche Kombinationen

Greedy Algorithmus:

- Immer Sender wählen, der die meisten neuen Staaten hinzufügt

```
public static void calculateSolution(HashSet<String> statesNeeded,
HashMap<String, HashSet<String>> stations) {
    var finalStations = new HashSet<String>();
    while (!statesNeeded.isEmpty()) {
        String bestStation = "";
        var statesCovered = new HashSet<String>();
        for (String station : stations.keySet()) {
            var covered = new HashSet<String>(statesNeeded);
            covered.retainAll(stations.get(station));
            if (covered.size() > statesCovered.size()) {
                bestStation = station;
                statesCovered = covered;
            }
        }
        statesNeeded.removeAll(statesCovered);
        finalStations.add(bestStation);
    }
    System.out.println(finalStations);
}
```

6.2 Binary Search

```
public static <T extends Comparable<T>> boolean searchBinary(List<T> data, T target, int low, int high) {
    if (low > high) {
        return false;
    } else {
        int pivot = low + ((high - low) / 2);
        if (target.equals(data.get(pivot))) {
            return true;
        } else if (target.compareTo(data.get(pivot)) < 0) {
            return searchBinary(data, target, low, pivot - 1);
        } else {
            return searchBinary(data, target, pivot + 1, high);
        }
    }
}
```

6.3 Backtracking

- Ziel erreicht:
 - Lösungspfad aktualisieren
 - **True** zurückgeben
- Wenn (x, y) bereits Teil des Lösungspfades:
 - **False** zurückgeben
- (x, y) als Teil des Lösungspfades markieren
- Vorwärts in X-Richtung suchen: \rightarrow
- Keine Lösung: In Y-Richtung abwärts suchen: \downarrow
- Keine Lösung: Zurück in X-Richtung suchen: \leftarrow
- Keine Lösung: Aufwärts in Y-Richtung suchen: \uparrow
- Immer noch keine Lösung: (x, y) aus Lösungspfad entfernen und **Backtracking**
- **False** zurückgeben

6.4 Dynamische Programmierung

```
public static long fibonacci(int n) {
    long[] f = new long[n + 2];
    f[0] = 0;
    f[1] = 1;

    for(int i = 2; i ≤ n; i++) {
        f[i] = f[i - 1] + f[i - 2];
    }

    return f[n];
}
```