Lernskript: Einfache Dateiein- und -ausgabe (File I/O)

Die Interaktion mit dem Dateisystem ist eine fundamentale Aufgabe in der Softwareentwicklung. Java bietet hierfür verschiedene Ansätze, bei denen sich der moderne Weg deutlich vom klassischen unterscheidet.

---

1. Daten aus einer Datei lesen

Klassisch (vor Java 7, mit `try-catch-finally`)

Der traditionelle Weg war umständlich und fehleranfällig. Ressourcen wie `Reader` oder `Streams` mussten **manuell** in einem `finally`-Block geschlossen werden, was oft vergessen wurde oder zu komplexen, verschachtelten `try-catch`-Blöcken führte.

```
// Klassisch - Nicht mehr verwenden!
BufferedReader reader = null;
try {
    reader = new BufferedReader(new FileReader("datei.txt"));
    String zeile;
    while ((zeile = reader.readLine()) != null) {
        System.out.println(zeile);
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (reader != null) {
        try {
            reader.close(); // Manuelles, fehleranfälliges Schließen
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Modern (seit Java 7, mit try-with-resources)

Dies ist der empfohlene Standard, wenn Dateien zeilenweise oder in Blöcken verarbeitet werden müssen. Jede Ressource, die in den runden Klammern des try deklariert wird, wird automatisch und garantiert am Ende des Blocks geschlossen.
Generated java

```
// Modern - Sicher und lesbar
try (BufferedReader reader = new BufferedReader(new FileReader("datei.txt"))) {
    String zeile;
    while ((zeile = reader.readLine()) != null) {
        System.out.println(zeile);
    }
```

```
} catch (IOException e) {
    e.printStackTrace();
}
```

Super-Modern (seit Java 11, mit Files.readString)

Für den häufigen Fall, dass der gesamte Inhalt einer (kleineren) Datei als einzelner String benötigt wird, bietet die Files-API eine extrem bequeme Einzeiler-Methode.
Generated java

```
// Super-Modern für das Lesen der kompletten Datei
try {
    String inhalt = Files.readString(Path.of("datei.txt"));
    System.out.println(inhalt);
} catch (IOException e) {
    e.printStackTrace();
}
```

2. Daten in eine Datei schreiben
Modern (seit Java 7, mit try-with-resources)

Genau wie beim Lesen ist try-with-resources der sichere Standardweg, der das manuelle Schließen von Writern überflüssig macht.
Generated java

```
// Modern - Sicher und lesbar
try (BufferedWriter writer = new BufferedWriter(new FileWriter("ausgabe.txt"))) {
    writer.write("Hallo Welt!");
    writer.newLine(); // Plattformunabhängiger Zeilenumbruch
    writer.write("Das ist die zweite Zeile.");
} catch (IOException e) {
    e.printStackTrace();
}
```

Super-Modern (seit Java 11, mit Files.writeString)

Um einen String schnell und einfach in eine Datei zu schreiben (und sie dabei zu überschreiben oder neu zu erstellen), ist dies der kürzeste Weg.
Generated java

```
// Super-Modern für das Schreiben eines Strings
```

```
try {
    String inhalt = "Dies ist der gesamte Inhalt, der geschrieben wird.";
    Files.writeString(Path.of("ausgabe.txt"), inhalt);
} catch (IOException e) {
    e.printStackTrace();
}
```