📓 Zusammenfassung: Bibliotheksklassen, Schnittstellen & Dokumentation

◇ 1. Bibliotheksklassen verstehen und nutzen

Java bringt eine grosse Standardbibliothek mit `Collections`, `IO`, `Math`, usw.

Für professionelle Programmierung ist es zentral, diese Bibliotheksklassen lesen, verstehen und richtig einsetzen zu können.

Ebenso wichtig ist es, eigene Klassen so zu dokumentieren, dass andere sie wie `Bibliotheksklassen` verwenden können.

📓 Wichtige Konzepte aus diesem Kapitel

◇ 2. Schnittstelle vs. Implementierung

- Schnittstelle: Was eine Klasse kann und wie sie benutzt wird `öffentliche Methoden`, `Sichtbarkeit`.
- Implementierung: Der vollständige Quelltext der Klasse.

Beispiel:

```
// Schnittstelle
public class Konto {
    public void einzahlen(double betrag) { ... }
    public double getSaldo() { ... }
}

    // Implementierung verborgen dank private
    private double saldo;
```

- Das Geheimnisprinzip `Information Hiding` sagt:

  → möglichst viel `private`, möglichst wenig `public`.

3. Javadoc & Dokumentation

Mit Kommentaren wie `/** ... */` kannst du automatisch `HTML-Dokumentation` erzeugen.

Andere Entwickler sollen die `Klasse` benutzen können, ohne den Code zu lesen. Beispiel:

```
/**
* Ein einfaches Konto mit Grundfunktionen.
  */
  public class Konto {
  /** Erhoeht den Kontostand um den angegebenen Betrag. */
    public void einzahlen(double betrag) { ... }
  }
```

4. Wichtige Klassen aus der Java-Bibliothek

Du hast kennengelernt:

- `Map` → Schlüssel-Wert-Struktur
- `Set` → Sammlung ohne doppelte Elemente
- `List` → geordnete Sammlung, Elemente dürfen mehrfach vorkommen

String als Beispiel für unveraenderliche Objekte (immutable)

Beispiele:

```
Map<String, Integer> punkte = new HashMap<>();
punkte.put("Anna", 10);
punkte.put("Ben", 20);

Set<String> namen = new HashSet<>();
namen.add("Anna");
namen.add("Anna");   // wird NICHT doppelt gespeichert
```

5. Autoboxing & Wrapper-Klassen

Primitive Typen z. B. `int` werden automatisch in **Objekte** umgewandelt `Integer`, wenn nötig.

Beispiel:

```
ArrayList<Integer> zahlen = new ArrayList<>();
zahlen.add(5);     // Autoboxing von int → Integer
```

6. Statische Elemente: Klassenvariablen & Klassenmethoden

`static` bedeutet: gehört der `Klasse`, nicht dem `Objekt`.

Von einer Klassenvariablen existiert nur 1 Kopie, egal wie viele Objekte erzeugt wurden.

Beispiel:

```
class Konto {
    static int anzahlKonten = 0;  // Klassenvariable

    Konto() {
        anzahlKonten++;          // erhoeht sich fuer alle Konten gemeinsam
    }
}
```

7. Polymorphe Variablen

Eine Variable vom Typ einer `Oberklasse` kann ein Objekt einer `Unterklasse` halten.

Beispiel:

```
List<String> liste = new ArrayList<>();    // polymorphe Variable
```

8. Methodenreferenzen

Nützlich beim Arbeiten mit Streams oder Lambdas (später relevant).

Beispiel:

```
namen.forEach(System.out::println);
```

Zusammenfassendes Beispiel

Hier ein kleines Beispiel, das viele Konzepte kombiniert:

```
import java.util.*;

/**
* Verwaltung von Punktestaenden.
  */
  public class Punktesystem {
    private Map<String, Integer> punkte = new HashMap<>();

    /** Fuegt einen Namen hinzu, wenn er noch nicht existiert. */
    public void addName(String name) {
        punkte.putIfAbsent(name, 0);
    }

  /** Erhoeht die Punkte eines Spielers. */
    public void addPunkte(String name, int wert) {
        punkte.put(name, punkte.get(name) + wert);
    }

  /** Gibt alle Punktestaende aus. */
    public void ausgabe() {
        punkte.forEach((n, p) ->
        System.out.println(n + ": " + p)); // Methodenreferenz waere auch moeglich
    }
  }
```

Dieses Beispiel zeigt:

- `Map`
- Dokumentation via `Javadoc`
- Geheimnisprinzip mit `private`
- Methodenreferenzen
- Schnittstelle vs. Implementierung

🧠 Merksätze

Gute Dokumentation ist so wichtig wie guter Code.

Schnittstelle zeigt, wie man eine `Klasse` benutzt — **Implementierung** zeigt, wie sie funktioniert.

Das **Geheimnisprinzip** macht Code sicherer und modularer.

`static` bedeutet: genau 1 Wert pro `Klasse`, nicht pro `Objekt`.

`Maps`, `Sets` und `Lists` sind zentrale Werkzeuge in Java.

Autoboxing verwandelt **primitive Typen** automatisch in `Objekte`.