Seminar 3

Einführung: Java Collections Framework (JCF)

Was ist das Java Collections Framework?

= eine Sammlung von Schnittstellen und Klassen, die Standarddatenstrukturen und -algorithmen bereitstellt. Es erleichtert die Arbeit mit Gruppen von Objekten (nicht wie Arrays, die feste Größe haben).

```
List<String> names = new ArrayList<>();
names.add("Anna");
names.add("Lukas");
```

→ Kein Limit der Größe, viele praktische Methoden, typsicher mit Generics.

Wichtige Interfaces und Implementierungen

Interface	**Typische Implementierungen**	**Eigenschaften**
List	ArrayList, LinkedList	Geordnete Sammlung, erlaubt Duplikate, Indexzugriff
Set	HashSet, TreeSet	Keine Duplikate, keine bestimmte Reihenfolge (außer TreeSet)
Map	HashMap, TreeMap	Schlüssel-Wert-Paare, key → value
Queue/Deque	PriorityQueue, LinkedList	FIFO/LIFO Verhalten, Warteschlangenprinzip

Warum Collections statt Arrays?

Arrays	**Collections**
Feste Größe	Dynamisch
Kein Typparameter	Typgesichert (Generics)
Wenig Komfortmethoden	Viele Utility-Methoden
Schwer erweiterbar	Leicht kombinierbar

=> Das JCF bietet **Flexibilität, Typsicherheit und Wiederverwendbarkeit**.

Beispiel, warum Arrays sind nicht type safe:

```
Object[] objects = new String[3]; // allowed because String[] is an Object[]
objects[0] = 42; // compiles, but...
```

=> compiles, but throws `java.lang.ArrayStoreException`

- Compile-time: Java sees Object[] and allows storing any object.
- Runtime: The array remembers it's really a String[], so storing a non-string violates that.

!! So arrays are type-checked at runtime, not fully at compile time — that's what makes them not type safe.

Iterator und Iterable

Warum brauchen wir sie?

Wenn man eigene Klassen „durchlaufen" möchte, z. B. mit einer for-each-Schleife:

```
for (Element e : myCollection) { ... }
```

→ Dann muss die Klasse das Interface **Iterable** implementieren.

Iterable liefert einen **Iterator**, der die Logik des Durchlaufens (Iteration) übernimmt.

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

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

→ Das bedeutet:

- Iterable liefert den Startpunkt für die Iteration
- Iterator weiß, wie man die Elemente durchläuft

Iterator Pattern

Grundidee

Das Iterator-Pattern ist ein Verhaltensmuster (behavioral design pattern), das beschreibt, **wie man Elemente einer Sammlung durchläuft, ohne deren interne Struktur offenzulegen.**

Motivation

Der Benutzer soll **nicht** wissen müssen, wie die Daten gespeichert sind (Array, Liste, Baum etc.), nur **wie man sie durchläuft** (hasNext(), next()).

Vorteile

- Kapselung (Encapsulation): interne Struktur bleibt privat

- Einheitliche Schnittstelle: für alle Collections gleich (iterator())
- Mehrere Iterationen gleichzeitig möglich
- Flexibilität: Man kann eigene Iterationslogiken implementieren (z. B. rückwärts, überspringend)

Good to know.

1. Wie funktioniert for-each in Java?

Sie verwendet den Iterator, ruft hasNext() und next() intern auf. Sie ruft intern den Iterator auf:

```
for (T e : collection) { ... }
```

wird zu

```
for (Iterator<T> it = collection.iterator(); it.hasNext();) {
    T e = it.next();
}
```

2. Was ist der Unterschied zwischen Iterator und ListIterator?

Iterator:

- Nur vorwärts iterierbar
- Gilt für jede Collection
- Keine Indexposition

ListIterator:

- Vorwärts und rückwärts
- nur für Listen
- kennt indexposition

3. Warum ist das Iterator-Pattern ein gutes Beispiel für Abstraktion?

Weil der Benutzer nicht wissen muss, wie Daten gespeichert sind, sondern nur wie er sie lesen kann.

Verbindung zu Generics

Alle Interfaces (Iterable, Iterator, Collection) verwenden Generics, damit der Iterator immer weiß, welchen Typ er liefert.

```
Iterator<Integer> it = numbers.iterator(); // gibt Integer zurück
```

→ Keine Casts nötig, keine Laufzeitfehler.

Verbindung zu Enums

Enums sind intern wie eine kleine Collection:

```
for (Day d : Day.values()) { ... }
```

→ Hier arbeitet Java ebenfalls mit dem Iterable-Mechanismus

Übung 1

- Erstelle eine Klasse MusicPlaylist, die Iterable implementiert.
- Songs werden in einer ArrayList gespeichert.

```
class Song {
    String title;
    Song(String title) { this.title = title; }
}

class MusicPlaylist implements Iterable<Song> {
    private List<Song> songs = new ArrayList<>();

    public void add(Song s) { songs.add(s); }

    @Override
    public Iterator<Song> iterator() {
        return songs.iterator();
    }
}
```

```
MusicPlaylist playlist = new MusicPlaylist();
playlist.add(new Song("Imagine"));
playlist.add(new Song("Hey Jude"));

for (Song s : playlist) {
    System.out.println(s.title);
}
```

Übung 2

Jede Klasse, die Iterable entsprechend implementiert, kann in for-each-Schleifen verwendet werden. Um eine iterierbare Datenstruktur zu implementieren, muss man:

1. Iterable implementieren
2. eine Iterator Klasse erstellen, die Iterator und die entsprechenden Methoden implementieren

```
class CustomDataStructure implements Iterable<> {

    // code for data structure
    public Iterator<> iterator() {
        return new CustomIterator<>(this);
    }
}
class CustomIterator<> implements Iterator<> {

    // constructor
    CustomIterator<>(CustomDataStructure obj) {
        // initialize cursor
    }

    // Checks if the next element exists
    public boolean hasNext() {
    }

    // moves the cursor/iterator to next element
    public T next() {
    }

    // Used to remove an element. Implement only if needed
    public void remove() {
        // Default throws UnsupportedOperationException.
    }
}
```

Übung 3

Implementieren Sie **eine Klasse Spielkarte** mit zwei Attribute: *farbe* und *wert*.
Farbe stellt bei Spielkarten die Sorte dar. Also Pik, Kreuz, Herz, Karo sind Farben.
Implementieren Sie eine **Klasse Deck**, damit der folgende Codefragment valid ist:

```
for (Spielkarte c : deck)
```

Übung 4

Implementieren Sie dieselbe Funktionalität von Punkt 1 aber mit Enum.

Übung 5

Implementieren Sie eine **Klasse TV** mit einer *Liste von Kanäle* als Attribut.
Sie soll auch zwei Methoden *channel_up* und *channel_down* bereitstellen, die die TV-Kanäle entsprechend wechseln.
Implementieren Sie eine **Klasse Remote** mit Methoden für Kanalsteuerung. Eine Fernbedienung kann nur mit einem TV funktionieren.

Übung 6

Zeichnen Sie das UML-Diagramm für Aufgabe 5. Die Einsendung des Diagramms an Mihaela über Teams könnte positiv berücksichtigt werden.