Programmierung: Java 2

Jürgen Hermes – IDH – SoSe 2025

Wiederholung: Iterator und Iterable

- Iterator: Interface das eine Iteration als Objekt darstellt
  - Ersetzt die klassische Schleife mit Zählvariable durch eine objektorientierte Lösung
  - Kann sehr einfach in einer while-Schleife genutzt werden.
  - Zentrale Methoden: boolean hasNext() und T next()
- Iterable: Interface, das signalisiert: Diese Klasse ist iterierbar!
  - Kann sehr einfach über eine for-each-Schleife genutzt werden – for(T element : collection)
  - Zentrale Methode: Iterator<T> iterator()

Das Konzept der Generics in Java

- Generics (quasi “Typschablonen”) ermöglichen es (seit Java 5), Typen als Platzhalter zu verwenden.
- Der konkrete Typ wird beim Verwenden angegeben und schon zur Kompilierzeit geprüft.
- `Bsp: Iterator<T>` ist ein Interface mit einem Typparameter `T` in spitzen Klammern.
- Beim Verwenden wird T durch eine konkrete Klasse ersetzt (z.B. String, Integer usw.).
- Der Iterator liefert dann Objekte genau dieses Typs.
- Typensicherheit: Fehler werden schon zur Kompilierzeit erkannt
- Keine (unsichere) Typumwandlungen (Casts) mehr nötig
- Wiederverwendbarkeit: Einmal geschriebener Code funktioniert für viele Typen.

Das Java Collections Framework (Einführung)

- Collections: Objekte, die eine Gruppe von Objekten repräsentieren.
- Frameworks: Sammlungen von Interfaces, Klassen und Algorithmen
- Das JCF (seit Java 2) wurde entwickelt für das strukturierte, effiziente und sichere Speichern, Verwalten und Bearbeiten von Gruppen von Objekten.
  - Vereinheitlichte Schnittstellen für verschiedene Datensammlungen
  - Wiederverwendbare Algorithmen (z.B. Sortieren, Suchen)
  - Flexibilität, Leistungsfähigkeit, Typensicherheit durch Generics
- Bestandteile des JCF:
  - Interfaces (`Collection`, `List`, `Set`, `Map` …)
  - Implementierungen (`ArrayList`, `LinkedList`, `HashSet`, `TreeSet`, `HashMap`, `TreeMap` …)
  - Hilfsklassen mit Algorithmen (`Collections.sort(...)`, `Collections.shuffle(...)` …)

Über Listen

- Grundidee: Endliche Anzahl von geordneten Objekten.
- Typ der Ordnung initial meist Reihenfolge der Einspeisung in die Liste.
- Zugriff über indexierte Elemente (wie bei Arrays, Startindex 0)
- Zentrale Methoden des JCF-Interfaces java.util.List (teilweise geerbt):
  - Einfügen oder Löschen: add, addAll, set, remove, removeAll, retainAll, clear ...
  - Auffinden oder geliefert bekommen: contains, indexOf, get, subList, lastIndexOf ...
  - Weiteres: iterator, size, sort, clear, isEmpty, toArray
- Implementationen: ArrayList vs. LinkedList

ArrayList vs. LinkedList

	ArrayList	**LinkedList**
Speicherstruktur	dynamisches Array, in einem Block gespeichert	(doppelt) verkettete Knoten, verteilt über Speicher
Zugriff	direkter Indexzugriff (schnell)	lineares Durchlaufen (langsam)
Manipulationen	Kopien aller Daten v. a. bei Einfügen / Löschen in der Mitte (langsam)	Lediglich Änderungen einzelner Zeiger nötig (schnell)
Speicherbedarf	nur enthaltene Elemente (kompakter)	zusätzliche Zeiger pro Element nötig
Anwendungsfall	häufige Lesezugriffe, seltene nachträgliche Änderungen	seltene Lesezugriffe, häufige nachträgliche Änderungen