Lernskript: Das Java Collections Framework (JCF)

Das Java Collections Framework (JCF) ist eine Sammlung von wiederverwendbaren Datenstrukturen (Klassen) und deren Verträgen (Interfaces), die in Java eingebaut ist, um Gruppen von Objekten zu verwalten.

Die 3 Kern-Interfaces

Die Hierarchie basiert auf Interfaces, die das "Was" definieren, und Klassen, die das "Wie" umsetzen.

1. `List` (Die geordnete Liste)

- **Konzept:** Eine geordnete Sammlung. Die Reihenfolge der Elemente bleibt erhalten. Duplikate sind erlaubt.
- **Wichtige Implementierungen:**
  - `ArrayList`: Intern basiert auf einem Array. Extrem schneller Lesezugriff per Index (`get(i)`). Langsam beim Einfügen/Löschen in der Mitte. **Ideal für lese-intensive Operationen.**
  - `LinkedList`: Intern basiert auf einer doppelt verketteten Liste. Extrem schnelles Einfügen/Löschen am Anfang und in der Mitte. Langsamer Lesezugriff per Index. **Ideal für schreib-intensive Operationen in der Mitte der Liste.**

2. `Set` (Die einzigartige Menge)

- **Konzept:** Eine Sammlung, die **keine Duplikate** enthält. Die meisten Implementierungen garantieren keine bestimmte Reihenfolge.
- **Wichtige Implementierungen:**
  - `HashSet`: Schnellste Implementierung für Hinzufügen, Suchen und Löschen. Garantiert **keine** Reihenfolge.
  - `LinkedHashSet`: Merkt sich die **Einfügereihenfolge**.
  - `TreeSet`: Hält die Elemente **automatisch sortiert**.

3. `Map` (Die Schlüssel-Wert-Zuordnung)

- **Konzept:** Eine Sammlung, die **Schlüssel-Wert-Paare** speichert. Jeder Schlüssel (`Key`) muss einzigartig sein.
- **Wichtige Implementierungen:**
  - `HashMap`: Schnellste Implementierung für das Speichern (`put`) und Abrufen (`get`). Garantiert **keine** Reihenfolge der Schlüssel.
  - `LinkedHashMap`: Merkt sich die **Einfügereihenfolge** der Schlüssel.
  - `TreeMap`: Hält die Paare **sortiert nach den Schlüsseln**.

"Classic vs. Modern"-Prinzipien

Programmieren gegen Interfaces (Klassisch & Modern)

Die Best Practice ist, Variablen immer mit dem Interface-Typ zu deklarieren. Das macht den Code flexibel, da die konkrete Implementierung leicht ausgetauscht werden kann.

```
List<String> userList = new ArrayList<>(); // Gut!
Set<Integer> uniqueIds = new HashSet<>();   // Gut!
Map<String, String> translations = new HashMap<>(); // Gut!
```

Unveränderliche Collections (Modern, seit Java 9)

Für feste, unveränderliche Sammlungen gibt es kurze und sichere Factory-Methoden.
Generated java

```
// Erstellt eine Liste, die nicht mehr verändert werden kann
List<String> farben = List.of("Rot", "Grün", "Blau");

// Erstellt ein Set, das nicht mehr verändert werden kann
Set<Integer> primzahlen = Set.of(2, 3, 5, 7);

// Erstellt eine Map, die nicht mehr verändert werden kann
Map<String, String> fehlercodes = Map.of("404", "Nicht gefunden", "500",
"Serverfehler");
```

Wichtige Konzepte im Hintergrund

    Generics (<...>): Sorgen für Typsicherheit zur Kompilierzeit und verhindern Casts zur Laufzeit.

    equals() & hashCode(): Müssen für eigene Objekte korrekt überschrieben werden, damit sie in HashSet oder als Schlüssel in HashMap korrekt funktionieren.