Java Collections: Ein Überblick mit Lösungen

Die **Java Collections** bieten flexible Datenstrukturen zum Speichern und Verwalten von Objekten. In Java gibt es verschiedene Collection-Typen, die jeweils unterschiedliche Eigenschaften und Einsatzmöglichkeiten bieten. Hier sind die wichtigsten Collections-Typen:

1. **Listen** (`List`)
2. **Mengen** (`Set`)
3. **Verzeichnisse** (`Map`)

Hinweis: Zum Verwenden dieser Klassen sollten Sie `import java.util.*;` hinzufügen.

---

1. Listen

Listen sind geordnete Datenstrukturen, die aufeinanderfolgende Elemente speichern. Sie erlauben den Zugriff über einen Index und das Einfügen an beliebigen Stellen. Die wichtigsten Implementierungen sind `ArrayList`, `LinkedList`, `Vector`, und `Stack`.

1.1 `ArrayList`

Die `ArrayList` speichert Elemente in einem dynamischen Array. Sie eignet sich gut für überwiegend lesenden Zugriff.

```
import java.util.ArrayList;

public class ListBeispiel {
    public static void main(String[] args) {
        ArrayList<String> autoListe = new ArrayList<>();
        autoListe.add("BMW");
        autoListe.add("Audi");
        autoListe.add("Ford");

        System.out.println("Inhalt der ArrayList: " + autoListe); // Inhalt der
ArrayList: [BMW, Audi, Ford]
        System.out.println("Erstes Element: " + autoListe.get(0)); // Erstes
Element: BMW
        System.out.println("Größe der Liste: " + autoListe.size()); // Größe der
Liste: 3
    }
}
```

Methoden von `ArrayList`

Methode	Beschreibung
`add(element)`	Fügt ein Element hinzu.

Methode	Beschreibung
`get(index)`	Gibt das Element am angegebenen Index zurück.
`size()`	Gibt die Anzahl der Elemente in der Liste zurück.
`remove(index)`	Entfernt das Element am angegebenen Index.
`contains(element)`	Prüft, ob ein Element in der Liste vorhanden ist.

2. Mengen (`Set`)

Mengen sind Datenstrukturen, die keine Duplikate zulassen. Die Reihenfolge der Elemente ist nicht festgelegt. Die gängigsten Implementierungen sind `HashSet` und `TreeSet`.

2.1 `HashSet`

Ein `HashSet` ist eine ungeordnete Menge, in der jedes Element nur einmal vorkommt.

```
import java.util.HashSet;

public class SetBeispiel {
    public static void main(String[] args) {
        HashSet<String> autoSet = new HashSet<>();
        autoSet.add("BMW");
        autoSet.add("Audi");
        autoSet.add("Ford");
        autoSet.add("BMW"); // Duplikat, wird nicht hinzugefügt.

        System.out.println("Inhalt des HashSet: " + autoSet); // Inhalt des
HashSet: [BMW, Audi, Ford]
        System.out.println("Ist Audi im Set? " + autoSet.contains("Audi")); // Ist
Audi im Set? true
    }
}
```

Methoden von `HashSet`

Methode	Beschreibung
`add(element)`	Fügt ein Element hinzu, wenn es nicht bereits existiert.
`size()`	Gibt die Anzahl der Elemente in der Menge zurück.
`contains(element)`	Prüft, ob ein Element in der Menge vorhanden ist.
`remove(element)`	Entfernt das Element aus der Menge.

3. Verzeichnisse (`Map`)

Verzeichnisse (Maps) speichern Daten als Schlüssel-Wert-Paare, wobei jeder Schlüssel eindeutig ist. Die gängigsten Implementierungen sind `HashMap` und `TreeMap`.

3.1 `HashMap`

Eine `HashMap` ordnet jedem Schlüssel einen Wert zu. Dies ist besonders nützlich für schnelle Zuordnungstabellen.

```
import java.util.HashMap;

public class MapBeispiel {
    public static void main(String[] args) {
        HashMap<String, Integer> zahlenMap = new HashMap<>();
        zahlenMap.put("Eins", 1);
        zahlenMap.put("Zwei", 2);
        zahlenMap.put("Drei", 3);

        System.out.println("Inhalt der HashMap: " + zahlenMap); // Inhalt der
HashMap: {Eins=1, Zwei=2, Drei=3}
        System.out.println("Wert für 'Drei': " + zahlenMap.get("Drei")); // Wert
für 'Drei': 3
        System.out.println("Enthält Schlüssel 'Eins'? " +
zahlenMap.containsKey("Eins")); // Enthält Schlüssel 'Eins'? true
    }
}
```

Methoden von `HashMap`

Methode	Beschreibung
`put(key, value)`	Fügt ein Schlüssel-Wert-Paar hinzu oder aktualisiert den Wert.
`get(key)`	Gibt den Wert für den angegebenen Schlüssel zurück.
`size()`	Gibt die Anzahl der Schlüssel-Wert-Paare zurück.
`containsKey(key)`	Prüft, ob ein Schlüssel vorhanden ist.
`containsValue(value)`	Prüft, ob ein Wert vorhanden ist.

Zusammenfassung

Collection-Typ	Beschreibung	Beispiele
List	Geordnete Liste von Elementen mit Zugriff per Index	`ArrayList`, `LinkedList`
Set	Menge von einzigartigen Elementen, keine Duplikate	`HashSet`, `TreeSet`

Collection-Typ	Beschreibung	Beispiele
Map	Schlüssel-Wert-Paare für schnelles Suchen nach einem Schlüssel	`HashMap`, `TreeMap`

Java Collections bieten eine Vielzahl an Methoden zur Verwaltung von Daten. Jede Collection hat ihre spezifischen Eigenschaften und eignet sich für bestimmte Anwendungen.