



# COLLECTIONS

KOLLEKTIONEN - SAMMLUNGEN





# COLLECTIONS

- Das **Collections Framework** ist eine Menge an häufig benötigten Datenstrukturen und dazu passenden Such und Sortieralgorithmen
- Die Gemeinsamkeiten der Collections befinden sich im Interface Collection
- Bis auf Datenstrukturen zur Realisierung von Mengen wird dieses Interface (oder daraus abgeleitete Interfaces) von allen Klassen des Collections Frameworks implementiert



# COLLECTIONS

- **Hauptaufgaben**
  - Daten effizient zu speichern
  - Effizienten Zugriff auf die Daten zu ermöglichen
- Beides sind konkurrierende Ziele, daher:
  - Wahl zwischen verschiedene Implementierungen, entweder:
    - sparsame Speicherung oder
    - schneller Zugriff begünstigen

# COLLECTIONS

Die wichtigsten Methoden der Schnittstelle Collection:

Platzhalter für den Typ der gespeicherten Objekte.  
Wird erst zur Laufzeit bzw. bei der Deklaration definiert („Generics“)



```
public interface Collection<E> extends Iterable<E> {  
    boolean add(E e);          fügt Objekt hinzu (Typ erst zur Laufzeit bekannt, s. o.)  
    boolean remove(Object o);   entfernt ein Objekt  
    int size();                 liefert Anzahl der Objekte zurück (nicht die Kapazität!)  
    boolean isEmpty();          prüft, ob irgendwelche Objekte enthalten sind  
    boolean contains(Object o); prüft, ob ein bestimmtes Objekt enthalten ist  
    void clear();               entfernt alle Objekte der Collection  
    Iterator<E> iterator();     liefert den Iterator zurück (s. nächste Lektion)  
    Object[] toArray();         liefert die Collection als einfachen Array zurück  
}
```

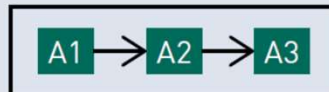
# COLLECTIONS

Die wichtigsten Collections:

## Programmieraufgabe

Artikel im Warenkorb  
verwalten

## Skizze



## Besondere Eigenschaften

Elemente in sequenziell geordneter Reihenfolge

## Interfaces/Klassen im Collections-Framework

Interface `java.util.List`

Beispiele für Implementierungen:

- `java.util.ArrayList` (als Array realisiert)
- `java.util.LinkedList` (als Verkettung von Referenzdatentypen realisiert)

# COLLECTIONS

Die wichtigsten Collections:

## Programmieraufgabe

Verwaltung des Shop-Sortiments

## Skizze



## Besondere Eigenschaften

Keine doppelten Elemente; Reihenfolge egal

## Interfaces/Klassen im Collections-Framework


Interface `java.util.Set`

Beispiele für Implementierungen:

- `java.util.TreeSet` (als Baum realisiert)
- `java.util.HashSet` (als Hash-Tabelle)

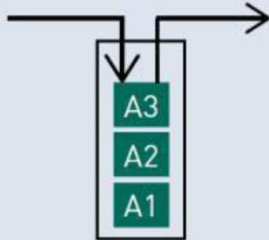
# COLLECTIONS

Die wichtigsten Collections:

Programmieraufgabe	Besondere Eigenschaften
Kundenverwaltung	Schneller Zugriff anhand der Kundennummer
Skizze	Interfaces/Klassen im Collections-Framework
	<p>Interface <code>java.util.Map</code> Beispiele für Implementierungen:</p> <ul style="list-style-type: none"><li>• <code>java.util.TreeMap</code> (als Baum realisiert)</li><li>• <code>java.util.HashMap</code> (als Hash-Tabelle)</li><li>• <code>java.util.LinkedHashMap</code> (Kombination aus Hash-Tabelle und verketteter Liste)</li></ul>

# COLLECTIONS


Die wichtigsten Collections:

Programmieraufgabe	Besondere Eigenschaften
„Undo“-Funktion im Bestellprozess	Die Bestellschritte des Benutzers sollen rückgängig gemacht werden können („Last in, First out“)
Skizze	Interfaces/Klassen im Collections-Framework
	<p>Interface <code>java.util.Deque</code> Beispielimplementierung:</p> <ul style="list-style-type: none"><li>• <code>java.util.ArrayDeque</code> (als Array realisiert)</li></ul> <p>Interface <code>java.util.List</code></p> <ul style="list-style-type: none"><li>• <code>java.util.Stack</code></li></ul>



# COLLECTIONS

Die wichtigsten Collections:

Programmieraufgabe	Besondere Eigenschaften
Warteschlange für Bestellungen	Bearbeitung in der Reihenfolge des Eingangs („First in, First out“)
Skizze	Interfaces/Klassen im Collections-Framework
	Interface <code>java.util.Queue</code> Beispielimplementierung: <ul style="list-style-type: none"><li>• <code>java.util.LinkedList</code> (s. o.)</li></ul>