ENGINEER SERIES

IN JAVA LANGUAGE

8
CH

INTRODUCTION TO COLLECTIONS

ENG : Qusay Khudair

Creativity and Accuracy in Work

AUG PLUS .Edu

Chapter 16

Introduction to Collections

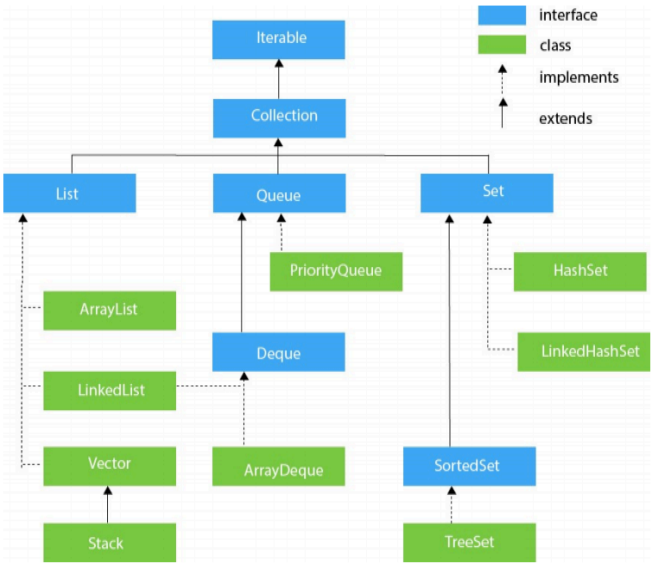
ENG : Qusay khudair

Introduction to Collections

- **Definition: A collection in Java is an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.**
- **Importance: Collections framework provides a well-structured set of interfaces and classes to handle groups of objects, making data manipulation more efficient.**
- **Core Concepts:**
    - **Interfaces: Define the types of collections (e.g., `List`, `Set`, `Queue`, `Map`).**
    - **Implementations: Concrete classes that implement these interfaces (e.g., `ArrayList`, `HashSet`, `PriorityQueue`, `HashMap`).**
    - **Algorithms: Methods that perform useful computations, like searching and sorting, on objects that implement collection interfaces.**

An Overview of the Collections Framework

- **What is the Collections Framework?**
  - **The Collections Framework is a unified architecture for representing and manipulating collections. It includes:**
    - **Interfaces: Abstract data types that represent collections.**
    - **Implementations: Concrete classes that implement the interfaces.**
    - **Algorithms: Static methods that perform operations on collections, like sorting and searching.**
- **Key Interfaces:**
  - `Collection`**: The root interface from which other collection interfaces like** `List`**,** `Set`**, and** `Queue` **are derived.**
  - `Map`**: A separate interface for handling key-value pairs, not derived from** `Collection`**.**



- The `Collection` interface is at the top of the hierarchy
- All `Collection` class implement this interface
- So all have a common set of methods

Types of Collection

- **Collection Interfaces:**
  - **List**: **Ordered collection** that can contain duplicate elements. Example: `ArrayList`, `LinkedList`.
  - **Set**: **Unordered collection** that cannot contain duplicate elements. Example: `HashSet`, `TreeSet`.
  - **Queue**: Collection designed for holding elements prior to processing. Example: `PriorityQueue`, `LinkedList`.

- **Map: A collection of key-value pairs. Example: `HashMap`, `TreeMap`.**

Map<Integer,String>

key	value
1	A
2	B
3	C

- **Key Characteristics:**
  - **List: Allows positional access, duplicate elements.**
  - **Set: No duplicates, unordered (except `TreeSet` which is ordered).**
  - **Queue: Typically FIFO (First-In-First-Out) order, can be priority-based.**
  - **Map: Maps unique keys to values, no duplicate keys.**

The Collection Interface

- **Overview: The `Collection` interface is the root of the collection hierarchy. It represents a group of objects known as its elements.**
- **Key Methods:**
  - `boolean add(E e)`**: Adds an element to the collection. Returns** `true` **if the element was successfully added to the collection.In some collections, like** `Set`**, it may return** `false` **if the collection does not allow duplicates and the element is already present.**

    Ex :

    ```
    List<String> list = new ArrayList<>();

    boolean added = list.add("Java");
    System.out.println(added); // Output: true
    ```

  - `boolean remove(Object o)`**: Removes a single instance of the specified element from the collection.Returns** `true` **if the element was successfully removed and Returns** `false` **if the element was not found in the collection.**

Ex :

```
List<String> list = new ArrayList<>();

list.add("Java");

 boolean removed = list.remove("Java");
System.out.println(removed); // Output: true
```

- `int size()`: **Returns the number of elements in the collection.**

Ex :

```
List<String> list = new ArrayList<>();

list.add("Java");

list.add("Python");

int size = list.size();

System.out.println(size); // Output: 2
```

- `boolean contains(Object o)`: **Returns** `true` **if the collection contains the specified element.Returns** `false` **if the element is not found.**
- **boolean isEmpty()**

- **Ex :**

```
List<String> list = new ArrayList<>();
 list.add("Java");
 boolean contains = list.contains("Java");
 System.out.println(contains); // Output: true
```

- `boolean isEmpty()`: checks whether a collection (such as a `List`, `Set`, or `Queue`) is empty. Returns `true` if the collection contains no elements and Returns `false` if the collection contains one or more elements.

```
List<String> list = new ArrayList<>();
System.out.println(list.isEmpty()); // Output: true
 list.add("Java");
 System.out.println(list.isEmpty()); // Output: false
```

- `Object[] toArray()`: converts the elements of a collection into an array of Object type.Returns an array containing all the elements in the collection.

```
List<String> list = new ArrayList<>();

list.add("Java");

list.add("Python");

Object[] array = list.toArray();
System.out.println(Arrays.toString(array)); //
Output: [Java, Python]
```

- `Iterator<T> iterator()` : **returns an iterator over the elements in the collection, which can be used to traverse the collection. Returns an Iterator object that can be used to iterate over the elements of the collection.**

```
List<String> list = new ArrayList<>();
list.add("Java");

 list.add("Python");

Iterator<String> iterator = list.iterator();

 while (iterator.hasNext()) {
System.out.println(iterator.next()); }

// Output:

 Java

 Python
```

An Overview of the Collections Framework

- Every class that implements the Collection interface has these methods.

```
import java.util.*;

public class CollectionTest {

public static void main(String[] args) {
Collection<String> col = new ArrayList<String>();
col = new TreeSet<String>();
col.add("Wesam");
col.add("Rami");
col.add("Ali");
col.add("Rami");

String s = col.toString();
System.out.println(col); // will print [Ali, Rami, Wesam]
col.remove("Khaled"); // false
boolean b = col.remove("Wesam"); // true

b = col.contains("Ali"); // true

for (String name : col) {
System.out.println(name);
   }
 }

}
```

Using Collections

import java.util.*

or

import java.util.Collection;

- **Note : There is a sister class, java.util.Collections; that provides a number of algorithms for use with collections: sort, binarySearch, copy, shuffle, reverse, max, min.**

```
import java.util.*;  // importing Arrays, List, and Collections

public class TestCollections {
    public static void main(String args[]) {
        String[] array = {"Rami", "Ali", "Wesam", "Tamer"};
        List<String> myList = Arrays.asList(array);
        Collections.sort(myList);  // sort(List<? Extends Comparable>
        System.out.println("Sorted: " + myList);
        int where = Collections.binarySearch(myList, "Tamer");
        System.out.println("Tamer is at " + where);
        Collections.shuffle(myList);
        System.out.println("Shuffled: " + myList);
    }
}
```

```
Output:
Sorted: [Ali, Rami, Tamer, Wesam]
Tamer is at 2
Shuffled: [Wesam, Ali, Rami, Tamer]
```

The List Interface

- **Overview: A `List` is an ordered collection (sometimes called a sequence). Lists can contain duplicate elements.**
  - **`ArrayList`: Resizable array implementation of the `List` interface.**
  - **`LinkedList`: Doubly-linked list implementation of the `List` interface.**
- **Key Methods:**
  - **`get(int index)`: Returns the element at the specified position in the list.**
  - **`set(int index, E element)`: Replaces the element at the specified position in the list with the specified element.**
  - **`add(int index, E element)`: Inserts the specified element at the specified position in the list.**
  - **`remove(int index)`: Removes the element at the specified position in the list.**

- `int indexOf(Object o):` returns the index of the first occurrence of the specified element in a list.Returns the index (0-based) of the first occurrence of the specified element in the list. and Returns `-1` if the element is not found.

```
List<String> list = new ArrayList<>();
list.add("Java");
list.add("Python");
list.add("C++");
int index = list.indexOf("Python");
System.out.println(index); // Output: 1
```

- `int lastIndexOf(Object o) :` returns the index of the last occurrence of the specified element in a list.Returns the index (0-based) of the last occurrence of the specified element in the list.Returns `-1` if the element is not found.

Usage Example:

```
List<String> list = new ArrayList<>();
list.add("Java");
list.add("Python");
list.add("C++");
list.add("Python");
```

```
int lastIndex = list.lastIndexOf("Python");
System.out.println(lastIndex); // Output: 3
```

- `ListIterator<E> listIterator()`: returns a `ListIterator` object for the list, which can be used to traverse the list in both forward and backward directions. Returns a `ListIterator` over the elements in the list.

  Usage Example:

```
List<String> list = new ArrayList<>();
list.add("Java");
list.add("Python");
list.add("C++");
ListIterator<String> iterator =
list.listIterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
// Output:
// Java
// Python
// C++
```

Full Example :

```
import java.util.*;


public class ListTest {
public static void main(String args[]) {
List<String> myList = new ArrayList<String>();
myList.add("Rami");
myList.add("Ali");
myList.add("Wesam");
myList.add("Tammer");
System.out.println(myList);
System.out.println(myList.get(0));
myList.set(1, "Khaled");
System.out.println(myList);
myList.set(2, "Ahmed");
System.out.println(myList);

for (int i = 0; i < myList.size(); i++) {
String value = myList.get(i);
System.out.println(value);
}

}
}
```

```
Output:
[Rami, Ali, Wesam, Tammer]
Rami
[Rami, Khaled, Wesam, Tammer]
[Rami, Khaled, Ahmed, Tammer]
Rami
Khaled
Ahmed
Tammer
```

The LinkedList Class of the Java Collections

- **LinkedList has the methods of the Collection interface and List interface .**

- **Common Methods:**
  - `addFirst(E e)`: **Inserts the specified element at the beginning of this list.**
  - `addLast(E e)`: **Appends the specified element to the end of this list.**
  - `removeFirst()`: **Removes and returns the first element from this list.**
  - `removeLast()`: **Removes and returns the last element from this list.**

Example Usage:

```
import java.util.*;

public class LinkedListTest {

public static void main(String args[]) {
LinkedList<String> list = new LinkedList<String>();
list.addFirst("Ali");
list.addLast("Rami");
list.addFirst("Tamer");
System.out.println(list); // will print [Tamer, Ali, Rami]
String name1 = list.getFirst();
String name2 = list.getLast();
name1 = list.removeFirst();
name2 = list.removeLast();
System.out.println(list); // will print [Ali]

}
}
```

```
Output:
[Tamer, Ali, Rami]
[Ali]
```

Collection Bulk Operations

- **Definition: Bulk operations are methods that operate on entire collections rather than individual elements.**

- **Key Bulk Operations:**
  - `boolean addAll(Collection<? extends E> c)`:

    Add all the elements of c2 in c1 [c1=c1 ∪ c2].

  - `boolean removeAll(Collection<?> c)`:

    Remove all the elements of c2 from c1[c1=c1 - c2].

  - `boolean retainAll(Collection<?> c)`:

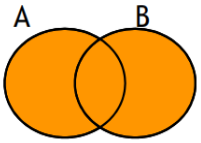
    Remove all the elements of c1 except the elements found in c2 [c1=c1 ∩ c2] .

  - `boolean containsAll(Collection<?> c)`:

    Return true if all the elements of c2 found in c1 .

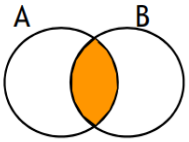
- **Note : addAll, removeAll, retainAll return true if the object receiving the message was modified.**

Look :::



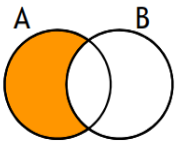
Collection union: $A \cup B$

colA.addAll(colB);



Collection intersection: $A \cap B$

colA.retainAll(colB);



collection difference: $A - B$

colA.removeAll(colB);

Mixing Collection Types

- **Overview: In Java, different collection types can be mixed together to achieve complex data structures and functionality.**

- **Note that most methods, such as**

  boolean containsAll(Collection<?> c);

  are defined for any type of Collection, and take any type of Collection as an argument

Ex :

```
List<String> myList = new ArrayList<String>();
//assume we add items to the myList
List<String> otherList = new ArrayList<String>();
otherList.addAll(myList);
```

The Iterator Interface

- **Overview: The `Iterator` interface provides a way to access elements of a collection sequentially without exposing its underlying structure.**
- **An iterator is an object that will return the elements of a collection,one at a time**
- **Key Methods:**
  - **`boolean hasNext()`: Returns `true` if the iteration has more elements.**

- `E next()`: Returns the next element in the iteration.throws NoSuchElementException- If there are no elements left in the iteration.
- `void remove()`: Removes from the underlying collection the last element returned by this iterator.
- `boolean hasPrevious()`: Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
- `E previous()`: Returns the previous element in the list.
- `void add(E e)`: Inserts the specified element into the list (optional operation).
- `void set(E e)`: Replaces the current element.

- Use Case:
  - Iterators are commonly used to traverse collections like `List`, `Set`, and `Queue`.

Example Code:

```
List<String> list = new ArrayList<>();

list.add("Java");
list.add("Python");
list.add("C++");
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
Output :
Java
Python
C++
```

Using an Iterator

- **Why Use an Iterator?**
  - **Iterators provide a consistent way to iterate over different types of collections.**
  - **They are safe to use even when the collection is modified during iteration (with limitations).**

Example Code:

```
List<String> list = new ArrayList<>();
list.add("Java");
list.add("Python");
list.add("C++");
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
 String lang = iterator.next();
    if (lang.equals("Python")) {
        iterator.remove(); // Removes "Python" from the list
    }
}
System.out.println(list); // Output: [Java, C++]
```

The Set Interface

- **Overview: A `Set` is a collection that does not allow duplicate elements. It models the mathematical set abstraction.**
- **Common Implementations:**
    - **`HashSet`: Backed by a hash table, offers constant time performance for basic operations.**

- `TreeSet`: Implements `NavigableSet` interface, elements are sorted in natural order or by a comparator provided at creation time.
- `LinkedHashSet`: Maintains a linked list of the entries in the set, in the order in which they were inserted.

- **The following methods are especially interesting:**
  - **boolean contains(Object o) // membership test**
  - **boolean containsAll(Collection<?> c) //subset test**
  - **boolean addAll(Collection<? extends E> c) // union**
  - **boolean retainAll(Collection<?> c) // intersection**
  - **boolean removeAll(Collection<?> c) // difference**
  - **addAll, retainAll, and removeAll return true if the receiving set is changed, and false otherwise**

- **Notes :**

- **Set also extends *Collection*, but it prohibits duplicate items (this is what defines a Set).**
- **No new methods are introduced; specifically, none for index-based operations (elements of Sets are not ordered).**

- **Concrete Set implementations contain methods that forbid adding two equal Objects.**
- **More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element**

Example Code 1 :

```
Set<String> set = new HashSet<>();
set.add("Java");
set.add("Python");
set.add("Java"); // Duplicate element, will not be added
System.out.println(set); // Output: [Java, Python]
```

Example Code 2 :

```
import java.util.*;

public class FindDups {
public static void main(String[] args) {
String words[]= {"Ali","Rami","Wesam","Ali","Ahmed","Tamer","Rami"};
Set<String> s = new HashSet<String>();
for (String a : words)
if (!s.add(a))
System.out.println("Duplicate detected: " + a);
System.out.println(s.size() + " distinct words: " + s);
}
}
```

```
Output:
Duplicate detected: Ali
Duplicate detected: Rami
5 distinct words: [Tamer, Ahmed, Wesam, Rami, Ali]
```

- **For More … Follow Me :**

  Eng.Qusay Khudair

- **For Business :**

  Linkedin

  Behance

  Github

- **For My Files and Slides :**

  studocu.com

- **Whatsapp : +972567166452**