Collections

Frida Cano Falcón

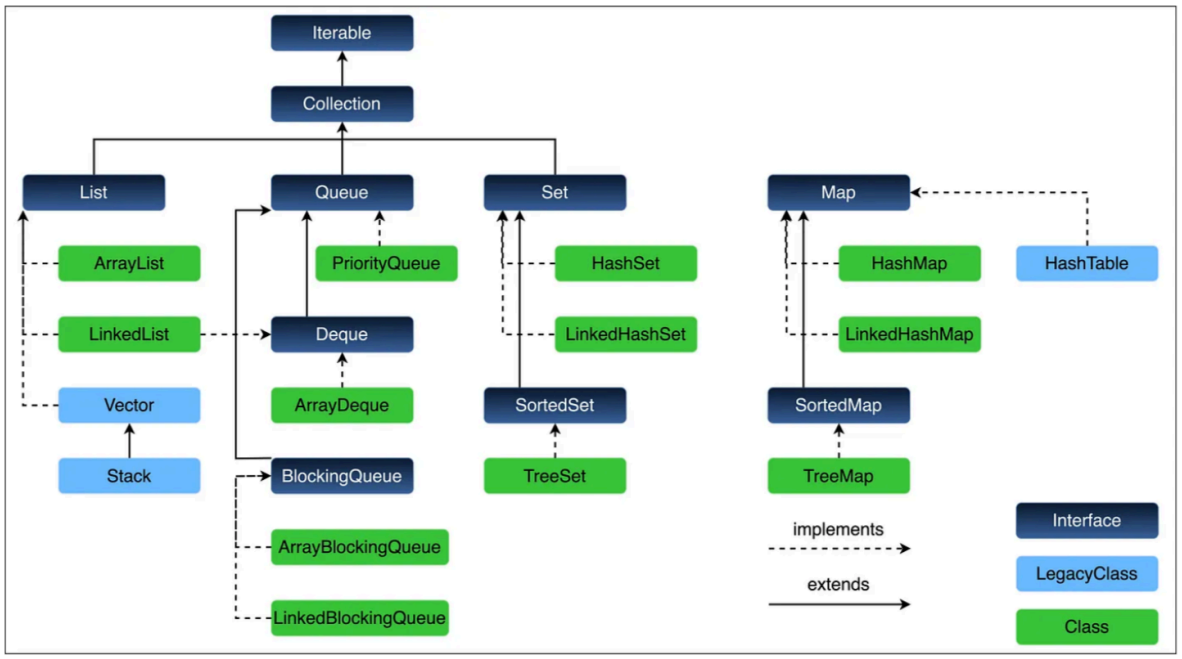
Java Backend Academy MTY
August 2024

Week 2

August 23, 2024

Introduction

In Java, collections are a framework that provides architecture to store and manipulate a group of objects. The Java Collections Framework (JCF) is a unified architecture for representing and manipulating collections. It includes interfaces, implementations (classes), and algorithms.



Collection Framework — Class Hierarchy

Collection Interfaces

The core collection interfaces in Java with examples are:

- **Collection**: The root of the collection hierarchy. A collection represents a group of objects known as its elements.
- **List**: An ordered collection that can contain duplicate elements. It provides precise control over where each element is inserted.
    - **ArrayList**: Resizable-array implementation of the *List* interface.

src > com > curso > tarea > Principal.java > Principal > main(String[])

```java
package com.curso.tarea;
import java.util.ArrayList;
import java.util.List;

public class Principal {
    Run | Debug
    public static void main(String[] args) {
        // Create a list of integers
        List<Integer> numbers = new ArrayList<>();

        // Add elements to the list
        numbers.add(e:10);
        numbers.add(e:20);
        numbers.add(e:30);

        // Display the list
        System.out.println("Numbers: " + numbers);
    }
}
```

- **LinkedList**: Doubly-linked list implementation of the *List* interface.

src > com > curso > tarea > LinkedListExample.java > LinkedListExample > main(String[])

```java
package com.curso.tarea;
import java.util.LinkedList;
import java.util.List;

public class LinkedListExample {
    Run | Debug
    public static void main(String[] args) {
        // Create a LinkedList of strings
        List<String> animals = new LinkedList<>();

        // Add elements to the LinkedList
        animals.add(e:"Dog");
        animals.add(e:"Cat");
        animals.add(e:"Horse");
        animals.add(e:"Elephant");

        // Display the LinkedList
        System.out.println("Animals: " + animals);

        // Add an element at a specific position
        animals.add(index:2, element:"Lion");

        // Display the LinkedList after adding an element at index 2
        System.out.println("Animals after adding Lion: " + animals);
    }
}
```

- **Set**: A collection that cannot contain duplicate elements.
  - **HashSet**: Implements the *Set* interface, backed by a hash table.

src > com > curso > tarea > HashMapExample.java > HashMapExample > main(String[])

```java
package com.curso.tarea;
import java.util.HashMap;
import java.util.Map;

public class HashMapExample {
    Run | Debug
    public static void main(String[] args) {
        // Create a map of integers to strings
        Map<Integer, String> map = new HashMap<>();

        // Add key-value pairs to the map
        map.put(key:1, value:"One");
        map.put(key:2, value:"Two");
        map.put(key:3, value:"Three");

        // Display the map
        System.out.println("Map: " + map);

        // Get a value by its key
        String value = map.get(key:2);
        System.out.println("Value for key 2: " + value);

        // Remove a key-value pair from the map
        map.remove(key:3);

        // Display the map after removal
        System.out.println("Map after removal: " + map);

        // Iterate over the map
        for (Map.Entry<Integer, String> entry : map.entrySet()) {
```

- **TreeSet**: Implements the *Set* interface, backed by a tree (e.g., Red-Black tree).

src > com > curso > tarea > TreeSetExample.java > {} com.curso.tarea

```java
package com.curso.tarea;
import java.util.Set;
import java.util.TreeSet;

public class TreeSetExample {
    Run | Debug
    public static void main(String[] args) {
        // Create a TreeSet of strings
        Set<String> fruits = new TreeSet<>();

        // Add elements to the TreeSet
        fruits.add(e:"Apple");
        fruits.add(e:"Banana");
        fruits.add(e:"Orange");

        // Display the TreeSet (will be in natural order)
        System.out.println("Fruits: " + fruits);

        // Remove an element from the TreeSet
        fruits.remove(o:"Banana");

        // Display the TreeSet after removal
        System.out.println("Fruits after removal: " + fruits);

        // Iterate over the TreeSet
        for (String fruit : fruits) {
            System.out.println("Fruit: " + fruit);
        }
    }
}
```

- **Queue**: A collection used to hold multiple elements prior to processing.
    - **PriorityQueue**: Implements a priority queue.

src > com > curso > tarea > PriorityQueueExample.java > PriorityQueueExample >

```java
package com.curso.tarea;
import java.util.PriorityQueue;
import java.util.Queue;

public class PriorityQueueExample {
    Run | Debug
    public static void main(String[] args) {
        // Create a priority queue of integers
        Queue<Integer> queue = new PriorityQueue<>();

        // Add elements to the queue
        queue.add(e:30);
        queue.add(e:10);
        queue.add(e:20);

        // Display the queue (Note: order may not reflect prio
        System.out.println("Queue: " + queue);

        // Poll the queue (retrieve and remove the head)
        int head = queue.poll();
        System.out.println("Head of queue: " + head);

        // Display the queue after polling
        System.out.println("Queue after polling: " + queue);

        // Iterate over the queue
        for (int number : queue) {
            System.out.println("Queue element: " + number);
        }
    }
}
```

- **Map**: An object that maps keys to values. A *Map* cannot contain duplicate keys; each key can map to at most one value.
    - **HashMap**: Implements the *Map* interface, backed by a hash table.

src > com > curso > tarea > HashSetExample.java > HashSetExample

```java
package com.curso.tarea;
import java.util.HashMap;
import java.util.Map;

public class HashSetExample {
    Run | Debug
    public static void main(String[] args) {
        // Create a HashMap of integers to strings
        Map<Integer, String> students = new HashMap<>();

        // Add key-value pairs to the HashMap
        students.put(key:1, value:"Alice");
        students.put(key:2, value:"Bob");
        students.put(key:3, value:"Charlie");

        // Display the HashMap
        System.out.println("Students: " + students);

        // Access a value by its key
        String student = students.get(key:2);
        System.out.println("Student with ID 2: " + student);

        // Remove a key-value pair by key
        students.remove(key:3);

        // Display the HashMap after removal
        System.out.println("Students after removing ID 3: " + students);

        // Iterate over the HashMap
        for (Map.Entry<Integer, String> entry : students.entrySet()) {
```

- **TreeMap**: A *Map* implementation that keeps its entries sorted according to the natural ordering of its keys.

src > com > curso > tarea > TreeMapExample.java > TreeMapExample > main(String[])

```java
import java.util.Map;
import java.util.TreeMap;

public class TreeMapExample {
    Run | Debug
    public static void main(String[] args) {
        // Create a TreeMap of integers to strings
        Map<Integer, String> books = new TreeMap<>();

        // Add key-value pairs to the TreeMap
        books.put(key:102, value:"The Catcher in the Rye");
        books.put(key:103, value:"1984");

        // Display the TreeMap (will be sorted by keys)
        System.out.println("Books: " + books);

        // Access a value by its key
        String book = books.get(key:101);
        System.out.println("Book with ID 101: " + book);

        // Remove a key-value pair by key
        books.remove(key:102);

        // Display the TreeMap after removal (still sorted by keys)
        System.out.println("Books after removing ID 102: " + books);

        // Iterate over the TreeMap
        for (Map.Entry<Integer, String> entry : books.entrySet()) {
            System.out.println("ID: " + entry.getKey() + ", Title: " + entry.getValue());
```