Java Collections Framework

The Java Collections Framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. It provides a unified architecture to store and manipulate a group of objects.

1. Functional Programming in Java

Functional programming is a paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. Java introduced functional programming concepts in Java 8 with features like lambdas, streams, and functional interfaces.

Key Functional Programming Concepts in Java:

Lambda Expressions: Allow us to write anonymous functions in a concise way.

Example:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

numbers.forEach(n -> System.out.println(n));
```

Functional Interfaces: An interface with a single abstract method (SAM). Common examples include Runnable, Callable, Comparator, and Java 8's Function, Predicate, Consumer, and Supplier.

Example of Functional Interface (Predicate):

```
Predicate<Integer> isEven = n -> n % 2 == 0;

System.out.println(isEven.test(4));  // Output: true
```

Streams: Provide a functional approach to processing sequences of elements, offering methods like map(), filter(), reduce(), and more.

Example of Stream:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

numbers.stream().filter(n -> n % 2 == 0).forEach(System.out::println);
```

2. Collections in Java

The Collections Framework is a unified architecture for representing and manipulating collections, enabling operations like searching, sorting, insertion, and deletion. The core of the collections framework is based on a few main interfaces.

Key Interfaces of the Collections Framework:

Collection: The root interface of the framework.

List: Ordered collections that allow duplicate elements.

Set: Unordered collections that do not allow duplicates.

Queue: Ordered collections that process elements in a specific order.

Map: A collection of key-value pairs where keys are unique.

3. Hierarchy of Collections Framework

The Collections Framework Hierarchy consists of several interfaces and classes. Below is an overview of the main elements in the hierarchy:

```
Collection
   |-- List
   |   |-- ArrayList
   |   |-- LinkedList
   |   |-- Vector
   |   `-- Stack
   |
   |-- Set
   |   |-- HashSet
   |   |-- LinkedHashSet
   |   `-- TreeSet
   |
   |-- Queue
```

```
        |-- PriorityQueue
        `-- Deque
            |-- ArrayDeque
            `-- LinkedList (implements Deque)


Map
   |-- HashMap
   |-- LinkedHashMap
   |-- TreeMap
   `-- WeakHashMap
```

Key Classes in the Java Collections Framework:

a. List Interface

The List interface extends Collection and represents an ordered sequence of elements. It allows duplicate elements.

ArrayList: A resizable array implementation of the List interface. It provides fast random access to elements.

Example:

```java
List<String> list = new ArrayList<>();
list.add("Apple");
list.add("Banana");
System.out.println(list);
```

LinkedList: A doubly-linked list implementation of the List and Deque interfaces. It's efficient for insertions and deletions.

Example:

```java
LinkedList<String> linkedList = new LinkedList<>();
linkedList.add("A");
linkedList.addFirst("B");
System.out.println(linkedList);
```

b. Set Interface

The Set interface extends Collection and represents a collection that does not allow duplicate elements.

HashSet: Implements the Set interface, backed by a hash table. It allows null elements and does not guarantee the order of elements.

Example:

```java
Set<String> set = new HashSet<>();
set.add("Apple");
set.add("Banana");
System.out.println(set);
```

TreeSet: Implements the Set interface, backed by a balanced tree. It stores elements in ascending order.

Example:

```java
Set<Integer> treeSet = new TreeSet<>();
treeSet.add(3);
treeSet.add(1);
treeSet.add(2);
System.out.println(treeSet);  // Output: [1, 2, 3]
```

c. Map Interface

The Map interface does not extend Collection but represents a collection of key-value pairs. Each key is unique, but values can be duplicated.

HashMap: Implements the Map interface, backed by a hash table. It allows null for both keys and values.

Example:

```
Map<String, Integer> map = new HashMap<>();
map.put("Apple", 1);
map.put("Banana", 2);
System.out.println(map);
```

TreeMap: Implements the Map interface, backed by a balanced tree. It stores entries sorted by keys.

Example:

```
Map<String, Integer> treeMap = new TreeMap<>();
treeMap.put("Apple", 1);
treeMap.put("Banana", 2);
System.out.println(treeMap);
```

d. Queue Interface

The Queue interface extends Collection and represents a collection designed for holding elements before processing. Typically, elements are processed in FIFO (First-In-First-Out) order.

PriorityQueue: Implements the Queue interface and sorts elements based on their priority.

Example:

```
Queue<Integer> queue = new PriorityQueue<>();
queue.add(10);
queue.add(20);
queue.add(15);
System.out.println(queue.poll());  // Output: 10 (smallest element)
```

4. Important Methods in Collection Framework

add(E e): Adds an element to the collection.

remove(Object o): Removes an element from the collection.

clear(): Removes all elements from the collection.

contains(Object o): Checks if the collection contains the specified element.

size(): Returns the number of elements in the collection.

isEmpty(): Checks if the collection is empty.

iterator(): Returns an iterator to traverse the collection.

5. Functional Programming with Collections

Functional programming can be applied to collections using streams and lambda expressions. Java 8 introduced the Stream API, which allows us to process collections using a functional approach.

Example of Functional Operations with Streams:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

// Filter and print even numbers
numbers.stream().filter(n -> n % 2 == 0).forEach(System.out::println);

// Sum all numbers
```

```
int sum = numbers.stream().reduce(0, (a, b) -> a + b);
System.out.println("Sum: " + sum);
```

6*. Summary of Key Collection Interfaces*

Interface	Description	Implementation Classes
List	Ordered collection, allows duplicates.	ArrayList, LinkedList, Vector, Stack
Set	Unordered collection, no duplicates.	HashSet, LinkedHashSet, TreeSet
Queue	Collection designed for holding elements before processing (FIFO).	PriorityQueue, Deque, ArrayDeque, LinkedList
Map	Collection of key-value pairs, unique keys.	HashMap, TreeMap, LinkedHashMap, WeakHashMap

Conclusion

The Java Collections Framework provides a comprehensive architecture for storing and manipulating data in the form of objects. It allows developers to work with different data structures like List, Set, Map, and Queue, along with providing thread-safe versions like ConcurrentHashMap. Java's inclusion of functional programming features, like lambda expressions and streams, further enhances how we work with collections in a concise and efficient manner.