Java Collection API

The Java Collection API is a set of interfaces, implementations, and algorithms provided by Java to handle and manipulate collections of objects. It provides a wide range of data structures, such as lists, sets, queues, maps, and more, along with various utility classes for sorting, searching, and manipulating collections.

Iterable Interface:
Iterable interface is part of the Java Collections Framework and provides a standard way to iterate over a collection of elements. It is defined in the java.lang package. Let's explore the Iterable interface in Java in depth.

The Iterable interface in Java is defined as follows:

```
public interface Iterable<T> {

  Iterator<T> iterator();

}
```

As seen in the definition, the Iterable interface is a generic interface that takes a type parameter T representing the type of elements in the collection. It declares a single method, iterator(), which returns an Iterator<T> object.

The Iterator interface is another important interface in Java that is returned by the iterator() method of the Iterable interface. It provides methods for iterating over elements in a collection. The Iterator interface is defined as follows:

```
public interface Iterator<E> {

  boolean hasNext();

  E next();

  void remove();

}
```

The Iterator interface has three methods:

1. hasNext(): This method returns true if there are more elements to iterate over; otherwise, it returns false.
2. next(): This method returns the next element in the iteration. It also advances the iterator to the next element.
3. remove(): This method removes the last element returned by next() from the underlying collection. This method is optional, and not all implementations of Iterator support it.

By implementing the Iterable interface, a class allows its instances to be used in enhanced for loops and other constructs that rely on the iterable protocol. To make a class iterable, it needs to provide an implementation of the iterator() method.

Here's an example to illustrate the usage of the Iterable interface:

```
import java.util.Iterator;

public class MyIterable implements Iterable<String> {

   private String[] elements = {"firstElement", "secondElement", "thirdElement"};

   @Override

   public Iterator<String> iterator() {

      return new MyIterator();

   }

   private class MyIterator implements Iterator<String> {

      private int index = 0;

      @Override

      public boolean hasNext() {

         return index < elements.length;

      }

      @Override

      public String next() {

         if (!hasNext()) {

            throw new NoSuchElementException();

         }

         return elements[index++];

      }

   }

   public static void main(String[] args) {

      MyIterable iterable = new MyIterable();

      for (String element : iterable) {

         System.out.println(element);
```

```
        }
    }
}
```

In this example, the MyIterable class implements the Iterable<String> interface and provides an implementation of the iterator() method. The iterator() method returns an instance of the MyIterator class, which is an implementation of the Iterator<String> interface.

In the main method, an instance of MyIterable is created, and it can be used in an enhanced for loop. The loop iterates over the elements in the MyIterable object, printing each element.

By implementing the Iterable interface and providing an iterator, custom classes can be used with the enhanced for loop and other Java constructs that work with iterables. It provides a standardized way to iterate over elements in a collection.

Java Collection

All java collection is Iterable as it extends Iterable interface

1. **Interfaces in collection api:**

    - Collection: It is the root interface which extends Iterable interface of the collection hierarchy. It defines the basic operations that can be performed on a collection, such as adding, removing, and querying elements.

    - List: It extends the Collection interface and represents an ordered collection of elements that allows duplicate values. Common implementations include ArrayList and LinkedList.

    - Set: It extends the Collection interface and represents a collection of unique elements. It does not allow duplicate values. Common implementations include HashSet and TreeSet.

    - Queue: It extends the Collection interface and represents a collection designed for holding elements before processing. It typically follows the FIFO (First-In-First-Out) principle. Common implementations include LinkedList and PriorityQueue.

Linked list implements both List and Queue interface so it can be considered as both List and Queue implementation

- Map: It represents a mapping between keys and values, where each key is unique. It allows efficient lookup and retrieval of values based on their associated keys. Common implementations include HashMap and TreeMap.

Basic Hierarchy of Collection api:

```
//Iterable returns Iterator
//    |
//Collection
//    - List - List of objects which maintains order
//           - ArrayList
//           - LinkedList
//    - Set - Unique set of object, which will not contain duplicate
//            - HashSet
//            - TreeSet
//    - Queue - List of object with first in first out implementation
//            - ArrayDequeue
//            - PriorityQueue
```

2. **Implementations:**

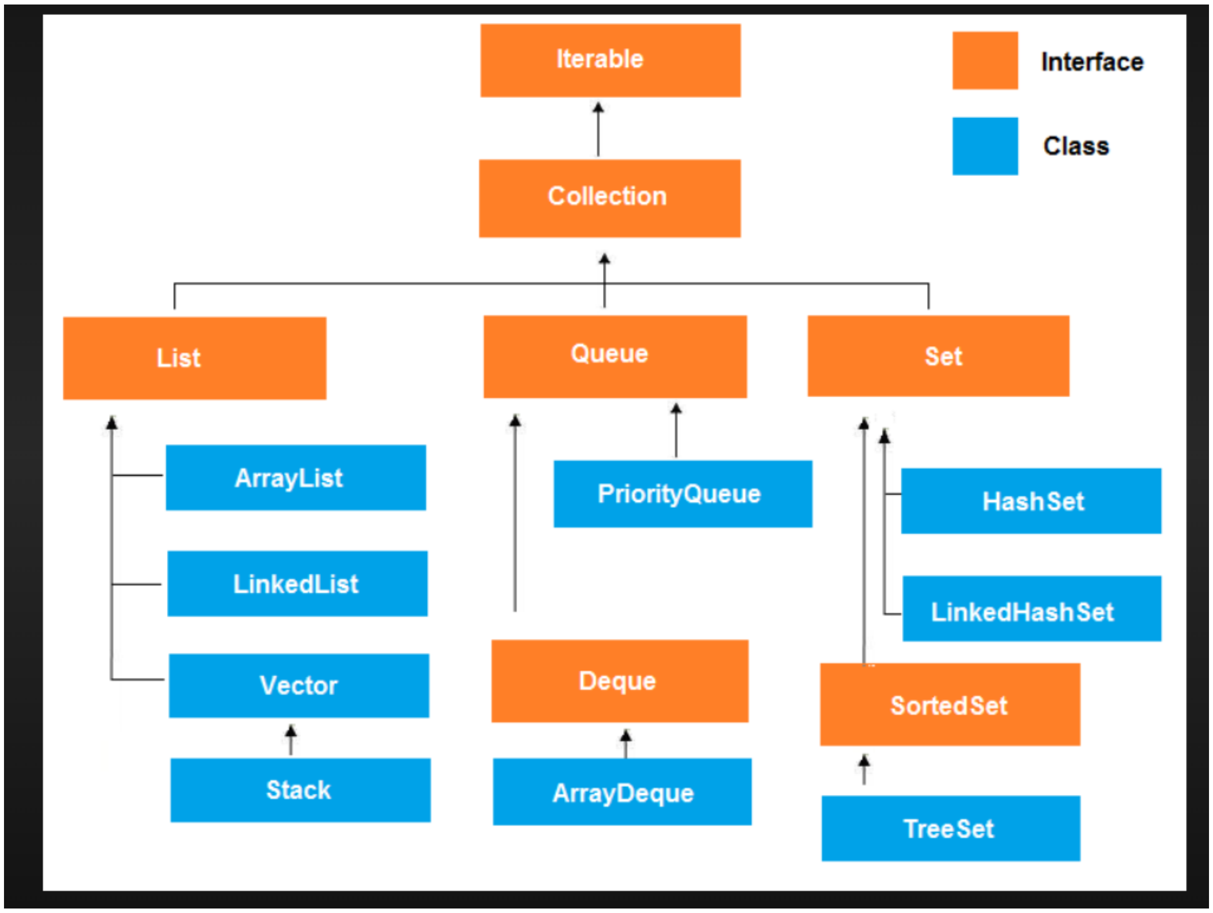
- ArrayList: It implements the List interface using a dynamically resizing array. It provides fast random access and is efficient for retrieving elements by index.
- LinkedList: It implements the List interface using a doubly-linked list. It provides fast insertion and removal operations but slower random access.
- HashSet: It implements the Set interface using a hash table. It provides constant-time performance for basic operations but does not guarantee a specific order of elements.
- TreeSet: It implements the Set interface using a self-balancing binary search tree. It maintains elements in sorted order and provides efficient operations for retrieval and traversal.
- HashMap: It implements the Map interface using a hash table. It allows efficient lookup and retrieval of values based on keys.
- TreeMap: It implements the Map interface using a self-balancing binary search tree. It maintains key-value pairs in sorted order based on the keys.

3. **Utility Classes:**

- Collections: It provides various static methods for manipulating and searching collections, such as sorting, shuffling, reversing, and finding the minimum or maximum element.
- Arrays: It offers static methods for working with arrays, including sorting, searching, and performing operations such as filling or copying arrays.

These are just a few highlights of the Java Collection API. It provides many more interfaces, implementations, and utility classes that can be utilized based on specific requirements. By using the collection framework, Java developers can efficiently manage and manipulate groups of objects in their applications.

Hierarchy diagram:


Iterable
Collection
List
Queue
Set
ArrayList
LinkedList
Vector
Stack
PriorityQueue
Deque
ArrayDeque
HashSet
LinkedHashSet
SortedSet
TreeSet
Interface
Class