Chapter 2

Collections Framework

1. Collections Framework Overview

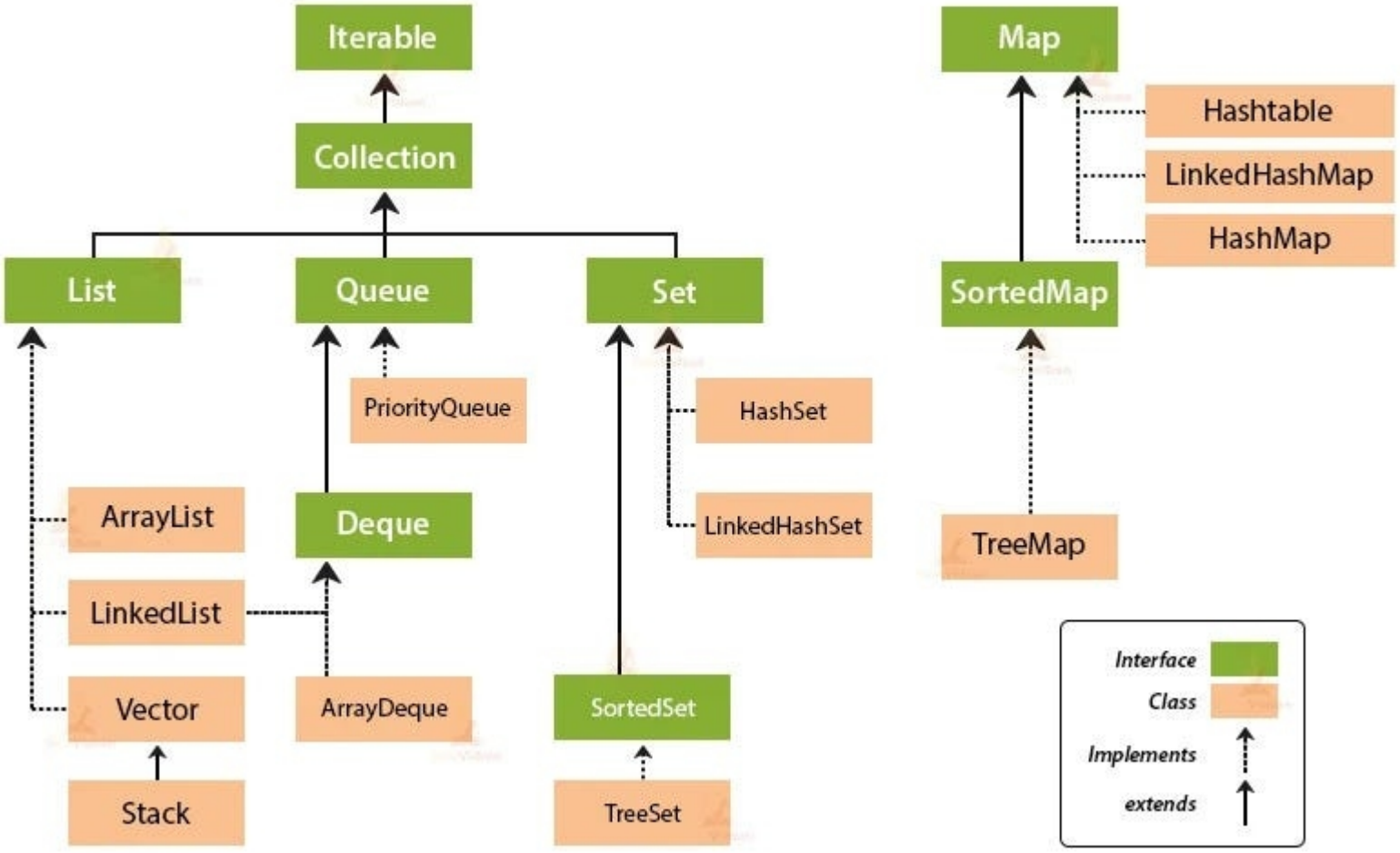
- Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms.
  - Achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation and deletion.
- It standardizes the way in which groups of objects are handled by your programs.
  - Designed to be high-performance
  - Allow different types of collections to work in a similar manner and with a high degree of interoperability.
  - Built standard interfaces and easy to adapt or extend.
  - Algorithms are defined as static methods within the Collections class.

1. Collections Framework Overview

- Collections provide a better way of doing several things

- The collections-framework classes and interfaces are members of package java.util.
- Java collection means a single unit of objects.
  - A collection represents a group of objects, known as its elements

Collection Framework Hierarchy in Java



2. Collection Interface

- It is at the top of the collections hierarchy.
- It is generic interface declared as: interface Collection<E>
- Extended by List, Set, and Queue interfaces.
- Determine the fundamental nature of the collection classes
  - The concrete classes provide different implementations of Collection interface.
- The Collection interface contains methods that perform basic operations, such as int size() , boolean isEmpty() , boolean contains(Object element) , boolean add(E element) , boolean remove(Object element) , boolean addAll(Collection<? extends E> c), Iterator<E> iterator(), ...

3. List Interface

- Store ordered collection of objects and can have duplicate elements.
- Declared as **public interface** List<E> **extends** Collection<E>
- In addition to the methods defined by Collection, List defines some of its own, like void add(int index, E e), boolean addAll(int index, Collection<? extends E> c), E get(int index), int indexOf(Object obj).
- Implemented by the classes ArrayList, LinkedList, Vector, and Stack.

ArrayList class

- ✔ Extends AbstractList and implements the List interface.
- ✔ Use dynamic array to store its elements, I.e created with an initial size.
  - When this size is exceeded, the collection is automatically enlarged.
  - When objects are removed, the array can be shrunk.

3. List Interface

- Maintains insertion order and are non-synchronized
- Its constructors here:

```
ArrayList( )
ArrayList(Collection<? extends E> c)
ArrayList(int capacity)
```

LinkedList class

- Uses double linked list internally to store elements
- Maintains insertion order and non-synchronized
- Its constructors are:

```
LinkedList( )
LinkedList(Collection<? extends E> c)
```

4. Iterators

- Iterators provide a way to access elements of collection sequentially without exposing its underline representation.
  - Separate traversal code from the list itself
  - Iterator keeps cursor to the current location.
- An iterator is an object that implements either the Iterator or the ListIterator interface.
  - Iterator enables you to cycle through a collection, obtaining or removing elements.
  - ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.
- Iterator and ListIterator are generic interfaces.

4. Iterators

- Iterator defines methods:-
  - boolean hasNext( )
    - Returns true if there are more elements. Otherwise, returns false. and
  - E next( )
    - Returns the next element. Throws NoSuchElementException if there is not a next element.
- ListIterator defines methods boolean hasPrevious( ) and E previous( )
- To use an iterator, obtain an iterator to the start of the collection by calling the collection's iterator( ) or listIterator( ) method.

4. Iterators

```
1  public static void main(String[] args) {
2          Integer [] arr = {3, 2, 9, 90, 87, 123, 21};
3          ArrayList<Integer> arrl = new ArrayList<>(Arrays.asList(arr));
4          ListIterator<Integer> i = li.listIterator();
5          while(i.hasNext())
6              System.out.print(i.next() + ", ");
7          System.out.println("\nThe list in reverse order");
8          while(i.hasPrevious())
9              System.out.print(i.previous()+ ", ");
10 }
```

5. Spliterators

- Like Iterator and ListIterator, Spliterator is a Java Iterator
- Spliterator is defined by the Spliterator interface.
- similar to the iterators. However, the techniques required to use it differ.
- Furthermore, it offers substantially more functionality than does either Iterator or ListIterator.
- Provide support for parallel iteration of portions of the sequence.
- Offers a streamlined approach that combines the hasNext and next operations into one method.

6. The Queue Interface

- Extends Collection interface and declares the behavior of a queue.
  - Is often a first-in, first-out list.
  - There are types of queues in which the ordering is based upon other criteria.
- Extended by Dequeue interface and implemented by PriorityQueue.

The PriorityQueue Class

- Creates a queue that is prioritized based on the queue's comparator.
- dynamic, growing as necessary.

6. The Queue Interface

- Has the following constructors

  PriorityQueue( )

  PriorityQueue(**int** capacity)

  PriorityQueue(Comparator<? **super** E> comp)

  PriorityQueue(**int** capacity, Comparator<? **super** E> comp)

  PriorityQueue(Collection<? **extends** E> c)

  PriorityQueue(PriorityQueue<? **extends** E> c)

  PriorityQueue(SortedSet<? **extends** E> c)

- The first constructor builds an empty queue. Its starting capacity is 11.
- The second constructor builds a queue with the specified initial capacity.
- The third constructor specifies a comparator.

6. The Queue Interface

- The fourth builds a queue with the specified capacity and comparator.
- The last three, create queues with the elements of the collection passed in.
- If no comparator is specified, then the default comparator for the type of data stored in the queue is used.
- The default comparator will order the queue in ascending order.
- Although you can iterate through a PriorityQueue using an iterator, the order of that iteration is undefined.
  - ✔ Better to use methods such as offer( ) and poll( ).

7. The Dequeue Interface

- Declares the behavior of a double-ended queue.
  - Double-ended queues can function as standard, first-in, first-out queues
  - or as last-in, first- out stacks.
  - Implemented by ArrayDequeue class

The ArrayDequeue Class

- Extends AbstractCollection and implements the Dequeue interface.
- Creates a dynamic array and has no capacity restrictions.
- Has the following constructors

```
ArrayDeque( )
ArrayDeque(int size)
ArrayDeque(Collection<? extends E> c)
```

8. The Set Interface

- The Set interface defines a set.
- It extends Collection and specifies the behavior of a collection that does not allow duplicate elements.
  - The add( ) method returns false if an attempt is made to add duplicate elements to a set.
- Extended by SortedSet interface and implemented by HashSet and SortedHashSet classes

The HashSet Class

- Creates a collection that uses a hash table for storage.
- The advantage of hashing is that it allows the execution time of add( ), contains( ),remove( ), and size( ) to remain constant even for large sets.

8. The Set Interface

- HashSet has the following constructors

```
HashSet( )
HashSet(Collection<? extends E> c)
HashSet(int capacity)
HashSet(int capacity, float fillRatio)
```

- The fourth constructor initializes both the capacity and the fill ratio (also called load factor) of the hash set.
  - The fill ratio must be between 0.0 and 1.0.
  - Default fill ratio is 0.75.
  - It determines how full the hash set can be before it is resized upward.
  - When number of elements is greater than the capacity of the hash set multiplied by its fill ratio, the hash set is expanded.

8. The Set Interface

The LinkedHashSet Class

- The LinkedHashSet class extends HashSet and adds no members of its own.
- LinkedHashSet maintains a linked list of the entries in the set, in the order in which they were inserted.
- This allows insertion-order iteration over the set.
  - That is, when cycling through a LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

9. The SortedSet Interface

- The SortedSet interface extends Set and declares the behavior of a set sorted in ascending order.

The TreeSet Class

- TreeSet creates a collection that uses a tree for storage.
- Objects are stored in sorted, ascending order.
- Access and retrieval times are quite fast.
  - Excellent choice to store large amounts of sorted information.

```
TreeSet( )
TreeSet(Collection<? extends E> c)
TreeSet(Comparator<? super E> comp)
TreeSet(SortedSet<E> ss)
```

10. The Map Interface

- A map is an object that stores key/value pairs.
- The keys must be unique, but the values may be duplicated.
- They don’t implement the Iterable interface.
  - You cannot cycle through a map using a for-each style for loop.
- Collections do not implement the Collection interface.
  - You can obtain a collection-view of a map.
    - entrySet( ) method returns a Set that contains the elements in the map.
    - KeySet( ) method returns a set that contains the keys in the map
    - Values( ) method is used to get a collection-view of the values.
  - For all three collection-views, the collection is backed by the map.
    - Changing one affects the other.

10. The Map Interface

- The Map interface is declared as:- interface Map<K, V>
- Extended by SortedMap interface
- Implemented by HashMap, LinkedHashMap, and HashTable classes

The HashMap Class

- Uses a hash table to store the map.
  - ✔ Execution time of get( ) and put( ) to remain constant even for large sets.
- Has the following constructors:

```
HashMap( )
HashMap(Map<? extends K, ? extends V> m)
HashMap(int capacity)
HashMap(int capacity, float fillRatio)
```

10. The Map Interface

The LinkedHashMap Class

- LinkedHashMap extends HashMap.
- It maintains a linked list of the entries in the map, in the order in which they were inserted.
  - ✔ This allows insertion-order iteration over the map.
- LinkedHashMap defines the following constructors:

```
LinkedHashMap( )
LinkedHashMap(Map<? extends K, ? extends V> m)
LinkedHashMap(int capacity)
LinkedHashMap(int capacity, float fillRatio)
LinkedHashMap(int capacity, float fillRatio, boolean Order)
```

11. The SortedMap Interface

- The SortedMap interface extends Map. It ensures that the entries are maintained in ascending order based on the keys.
- Sorted maps allow very efficient manipulations of submaps
  - To obtain a submap, use headMap( ), tailMap( ), or subMap( ).
  - The returned is backed by the invoking map.

The TreeMap Class

- Extends AbstractMap and implements the NavigableMap interface.
- It creates maps stored in a tree structure.
- Provides an efficient means of storing key/value pairs in sorted order and allows rapid retrieval.

11. The SortedMap Interface

- TreeMap defines the following constructors:

  TreeMap( )

  TreeMap(Comparator<? **super** K> comp)

  TreeMap(Map<? **extends** K, ? **extends** V> m)

  TreeMap(SortedMap<K, ? **extends** V> sm)

12. Collections class

- Class Collections provides static methods that search, sort, and perform other operations on collections.
- Provides several high-performance algorithms for manipulating collection elements.
- The set of methods that begins with unmodifiable returns views of the various collections that cannot be modified.
- methods, such as synchronizedList( ) and synchronizedSet( ), are used to obtain synchronized (thread-safe) copies of the various collections.
- The set of checked methods, such as checkedCollection( ) monitors insertions into the collection for type compatibility at run time

13. Arrays class

- The Arrays class provides various static utility methods that are useful to work with arrays.
- These methods help to bridge the gap between collections and arrays.
- The asList( ) method returns a List that is backed by a specified array.
  - both the list and the array refer to the same location.

14. Legacy Collections

- Early versions of java.util did not include the Collections Framework
- Instead, it defined several classes that provided an ad hoc method of storing objects.
- When collections were added (by J2SE 1.2), several of the original classes were re-engineered to support the collection interfaces.
- All the legacy classes are synchronized.

Vector

- Similar to ArrayList.
- Contains many methods that are not part of collection framework.
- Re-engineered to extend AbstractList and to implement the List and Iterable interfaces.

14. Legacy Collections

- Here are the Vector constructors:

```
Vector( )
Vector(int size)
Vector(int size, int incr)
Vector(Collection<? extends E> c)
```

Stack

- Is subclass of Vector and Implements first-in first-out data structure, Stack.
- Contains all methods of Vector class and also provides methods like push()

14. Legacy Collections

Hashtable

- Re-engineered to implement the Map interface.
- It is similar to HashMap.
- The Hashtable constructors are shown here:

```
Hashtable( )
Hashtable(int size)
Hashtable(int size, float fillRatio)
Hashtable(Map<? extends K, ? extends V> m)
```

15. Factory Methods

- Java SE 9 adds new static factory methods to interfaces List, Set and Map that enable you to create small immutable collections
  - they cannot be modified once they are created
- The convenience factory methods instead return custom collection objects that are truly immutable and optimized to store small collections.
- factory method of to create an immutable List<String>.
- Method of has overloads for Lists of zero to 10 elements and an additional overload that can receive any number of elements.