# Unit-3

Collections Framework

# Unit-3 contents

**Introduction To Collection Framework:** Collections Overview**,**

**The Collection Interfaces**: The collection Interface, The List Interface, The Set Interface, The SortedSet Interface, The NavigableSet Interface, The Queue Interface, The Deque Interface.

**The Collection Classes**: The ArrayList Class, The LinkedList class, The HashSet Class, The LinkedHashSet Class, The TreeSet Class, The PriorityQueue Class, The ArrayDeque Class, The EnumSet Class.

Accessing a Collection Via an Iterator, Spliterators.

# Collections framework

- A *collection* is an object that represents a group of objects (such as the classic Vector class).

- A collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details.

The collections framework consists of

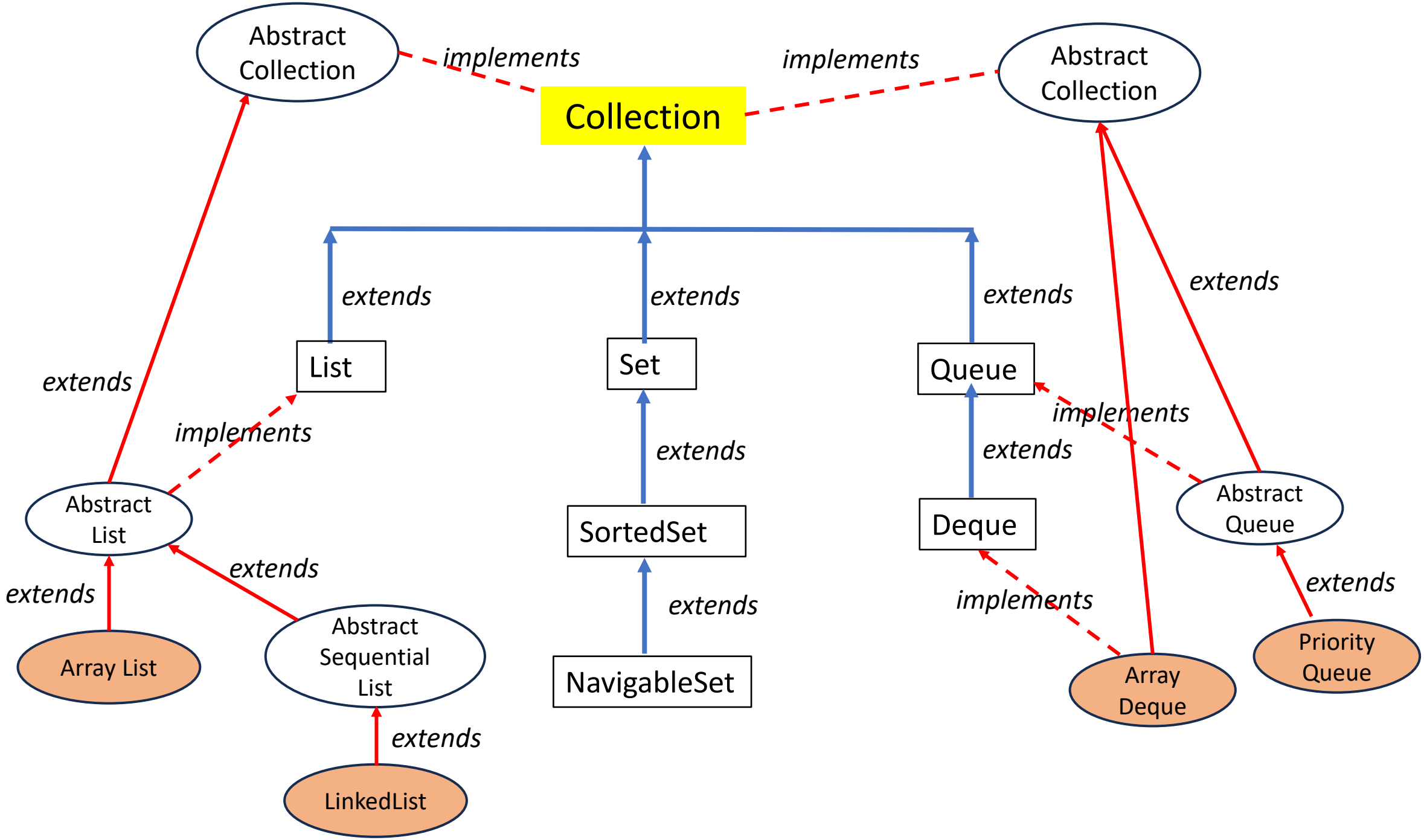
- Collection Interfaces
- General-purpose implementation
- Legacy implementation
- Special-purpose implementation
- Concurrent Implementation
- Wrapper Implementation
- Convenience Implementation
- Abstract Implementation
- Algorithms
- Infrastructure
- Array Utilities

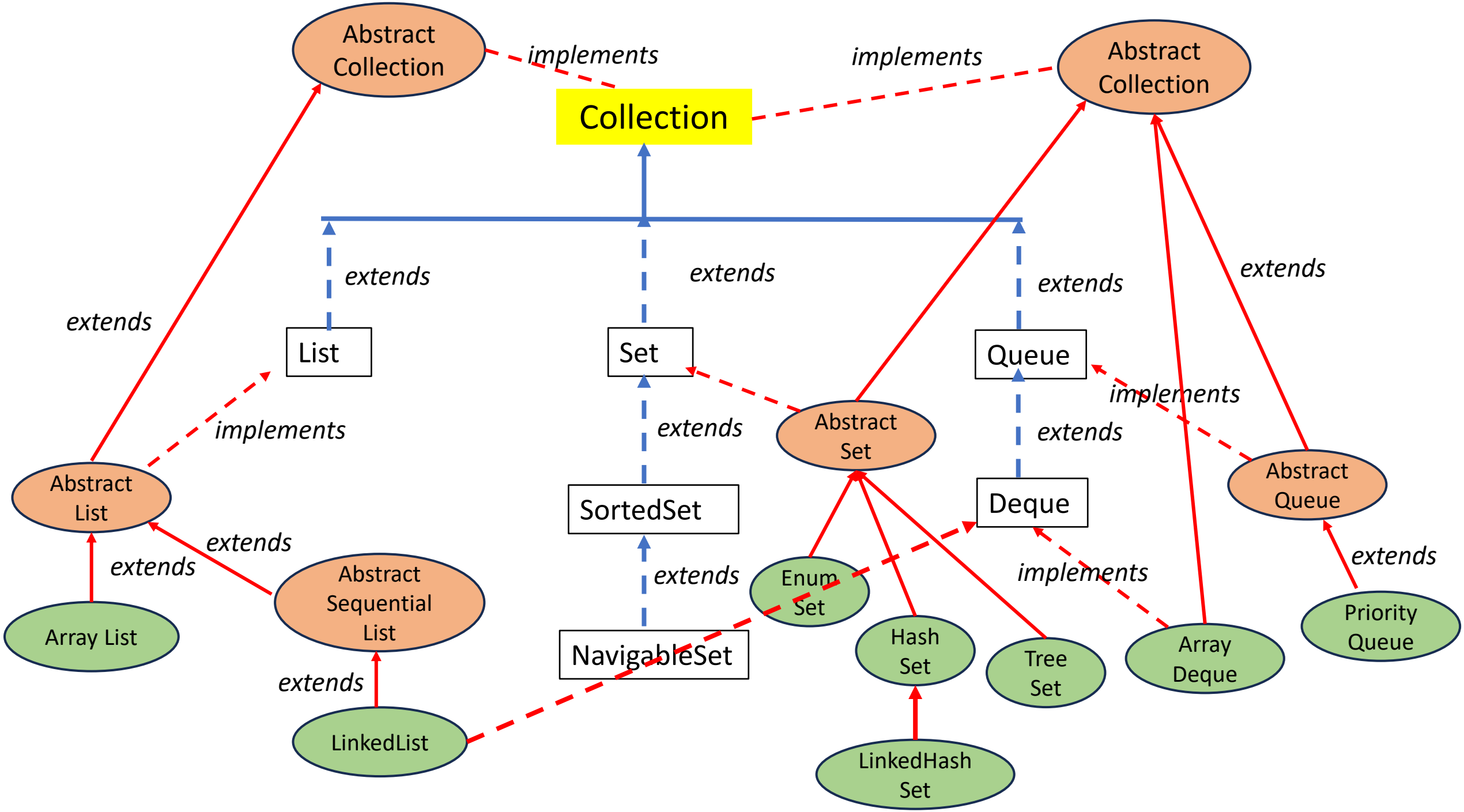
# Interfaces

| Interface | Description |
|---|---|
| Collection | Enables you to work with groups of objects; it is at the top of the collections hierarchy. |
| Deque | Extends **Queue** to handle a double-ended queue. |
| List | Extends **Collection** to handle sequences (lists of objects). |
| NavigableSet | Extends **SortedSet** to handle retrieval of elements based on closest-match searches. |
| Queue | Extends **Collection** to handle special types of lists in which elements are removed only from the head. |
| Set | Extends **Collection** to handle sets, which must contain unique elements. |
| SortedSet | Extends **Set** to handle sorted sets. |

# Collection Implementation

| Interface | Hash Table | Resizable Array | Balanced Tree | Linked List | Hash Table + Linked List |
|---|---|---|---|---|---|
| Set | HashSet | | TreeSet | | LinkedHashSet |
| List | | ArrayList | | LinkedList | |
| Deque | | ArrayDeque | | LinkedList | |
| Map | HashMap | | TreeMap | | LinkedHashMap |


Abstract Collection
implements
Collection
implements
Abstract Collection
extends
extends
extends
extends
List
Set
Queue
extends
implements
extends
implements
Abstract List
extends
SortedSet
extends
Deque
Abstract Queue
extends
extends
extends
implements
extends
Array List
Abstract Sequential List
NavigableSet
Array Deque
Priority Queue
extends
LinkedList


Abstract Collection
implements
Collection
implements
Abstract Collection
extends
extends
extends
extends
extends
List
Set
Queue
implements
extends
extends
Abstract List
Abstract Set
SortedSet
Deque
Abstract Queue
extends
extends
extends
extends
implements
extends
Array List
Abstract Sequential List
Enum Set
Hash Set
Tree Set
Array Deque
Priority Queue
extends
NavigableSet
LinkedList
LinkedHash Set

# Collection Interface

- **Collection** is the root interface of all the collections hierarchy, that represents a group of objects, known as its elements. **Collection** declares the core methods that all collections will have.

| Method | Description |
|---|---|
| boolean add(E ***obj***) | Adds ***obj*** to the invoking collection. Returns true if ***obj*** was added to the collection. Otherwise returns false. |
| boolean addAll(Collection<? extends E> c) | Adds all elements of c to the invoking collection. Returns true if the collection were changed . Otherwise return false |
| boolean remove(Object ***obj***) | Removes ***obj*** to the invoking collection. Returns true if ***obj*** was added to the collection. Otherwise returns false. |
| boolean removeAll(Collection<?> c) | Removes all elements of c to the invoking collection. Returns true if the collection were changed . Otherwise return false |
| boolean removeIf(Predicate<? Super E> predicate) | Removes all of the elements of this collection that satisfy the given predicate. |
| void clear() | Removes all elements from the invoking collection. |
| boolean contains(Object obj) | Returns true if obj is an element of the invoking collection. Otherwise returns false |
| boolean conatinsAll(Collection<?> c) | Returns true if the invoking collection contains all elements of c. Otherwise returns false. |

| Method | Description |
| --- | --- |
| Iterator<E> iterator() | Returns an iterator for the invoking collection. |
| Object[ ] toArray( ) | Returns an array of elements from the invoking collection. |
| <T> T[ ]  toArray(T[ ] a) | Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array. |
| boolean isEmpty() | Return true if the invoking collection is empty . Otherwise, returns false. |
| int size() | Returns the number of elements of the collection. |
| boolean equals(Object obj) | Returns true if the invoking collection and ***obj*** are equal. Otherwise, returns false. |

# List Interface

A List is an ordered Collection (sometimes called a *sequence*). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:

| | |
|---|---|
| Positional access | ➢ manipulates elements based on their numerical position in the list. This includes methods such as ***get, set, add, addAll, and remove.*** |
| Search | ➢ searches for a specified object in the list and returns its numerical position. Search methods include ***indexOf and lastIndexOf.*** |
| Iteration | ➢ extends Iterator semantics to take advantage of the list's sequential nature. The ***listIterator*** methods provide this behavior. |
| Range-view | ➢ The ***sublist*** method performs arbitrary *range operations* on the list. |

The Java platform contains two general-purpose List implementations. ArrayList, which is usually the better-performing implementation, and LinkedList which offers better performance under certain circumstances.

# List Interface

- The **List** interface extends **Collection** and declares the behavior of a collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- It may contain duplicate elements.

interface List<E>

List defines some of its own methods in addition to collection methods

| **Method** | **Description** |
|---|---|
| void add(int index, E obj) | Returns an iterator for the invoking collection. |
| boolean addAll(int index, Collection<? extends E> c) | Returns an array of elements from the invoking collection. |
| E get(int index) | Returns the object stored at the specified index with in the invoking collection |
| int indexOf(Object obj) | Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list ,-1 is returned. |
| int lastIndexOf(Object obj) | Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list ,-1 is returned. |

| Method | Description |
| --- | --- |
| ListIterator<E> listIterator() | Returns an iterator to the start of the invoking list. |
| ListIterator<E> listIterator(int index) | Returns an iterator to the invoking list that begins at the specified index. |
| E remove(int index) | Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted and the indexes of subsequent elements are decremented by one. |
| Default void replaceAll(UnaryOperator<E> opToApply) | Updates each element in the list with the value obtained from the opToApply function. |
| E set(int index,E obj) | Assigns obj to the location specified by index within the invoking list. Returns the old value. |
| default void sort(Comparator<? Super E> ***comp***) | Sorts the list using the comparator specified by ***comp*** |
| List<E> subList(int start, int end) | Returns a list that includes Elements from start to end-1 in the invoking list. Elements in the returned list are also referenced by the invoking object. |

# Collection classes

| Class | Description |
|---|---|
| AbstractCollection | Implements most of the **Collection** interface. |
| AbstractList | Extends **AbstractCollection** and implements most of the **List** interface. |
| AbstractQueue | Extends **AbstractCollection** and implements parts of the **Queue** interface. |
| AbstractSequentialList | Extends **AbstractList** for use by a collection that uses sequential rather than random access of its elements. |
| LinkedList | Implements a linked list by extending **AbstractSequentialList**. |
| ArrayList | Implements a dynamic array by extending **AbstractList**. |
| ArrayDeque | Implements a dynamic double-ended queue by extending **AbstractCollection** and implementing the **Deque** interface. |
| AbstractSet | Extends **AbstractCollection** and implements most of the **Set** interface. |
| EnumSet | Extends **AbstractSet** for use with **enum** elements. |
| HashSet | Extends **AbstractSet** for use with a hash table. |
| LinkedHashSet | Extends **HashSet** to allow insertion-order iterations. |
| PriorityQueue | Extends **AbstractQueue** to support a priority-based queue. |
| TreeSet | Implements a set stored in a tree. Extends **AbstractSet**. |

# ArrayList

- Resizable-array implementation of the **List** interface.

Class syntax

public class **ArrayList<E>**
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable

Constructors

| |
|---|
| **ArrayList**()<br>Constructs an empty list with an initial capacity of ten. |
| **ArrayList**(Collection<? extends **E**> c)<br>Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. |
| **ArrayList**(int initialCapacity)<br>Constructs an empty list with the specified initial capacity. |

# LinkedList

Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

public class **LinkedList<E>**
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable

Constructors

**LinkedList**()Constructs an empty list.

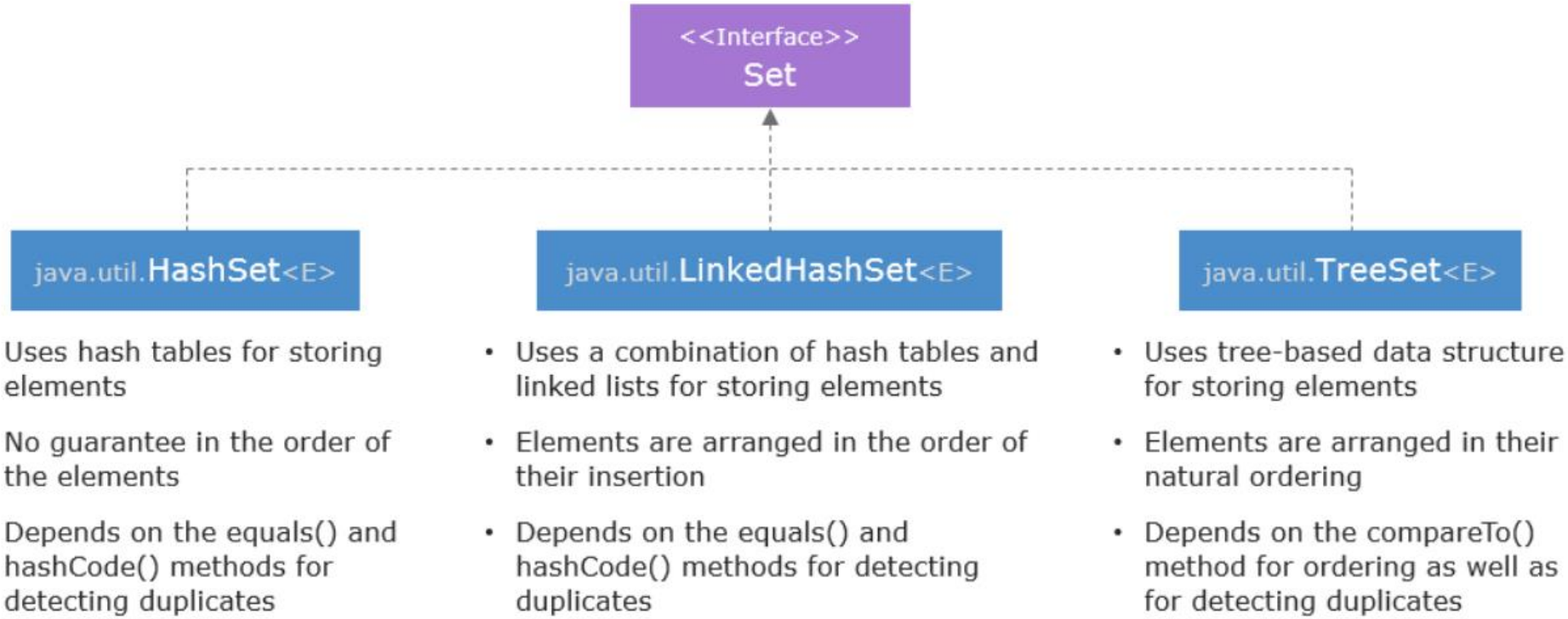
**LinkedList**(**Collection**<? extends **E**> c)Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

# Set Interface

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.

The `Set` interface contains *only* methods inherited from `Collection` and adds the restriction that duplicate elements are prohibited.

Java provides 3 general purpose implementations of set. HashSet ,TreeSet ,LinkedHashSet.



# HashSet

public class **HashSet<E>** extends AbstractSet<E>
implements Set<E>, Cloneable, Serializable

- This class implements the **Set** interface, backed by a hash table (actually a HashMap instance).
- It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time.
- Permits the **null** element.
- This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.
- Iterating over this set requires time proportional to the sum of the HashSet instance's size (the number of elements) plus the "capacity" of the backing HashMap instance (the number of buckets).

**HashSet**()
Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).

**HashSet**(**Collection**<? extends **E**> c)
Constructs a new set containing the elements in the specified collection.

**HashSet**(int initialCapacity)
Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).

| Constructors |
| --- |
| Constructor and Description |
| **HashSet**()<br>Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75). |
| **HashSet**(**Collection**<? extends **E**> c)<br>Constructs a new set containing the elements in the specified collection. |
| **HashSet**(int initialCapacity)<br>Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75). |
| **HashSet**(int initialCapacity, float loadFactor)<br>Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor. |

- s1.containsAll(s2) — returns true if s2 is a **subset** of s1. (s2 is a subset of s1 if set s1 contains all of the elements in s2.)

- s1.addAll(s2) — transforms s1 into the **union** of s1 and s2. (The union of two sets is the set containing all of the elements contained in either set.)

- s1.retainAll(s2) — transforms s1 into the **intersection** of s1 and s2. (The intersection of two sets is the set containing only the elements common to both sets.)

- s1.removeAll(s2) — transforms s1 into the (asymmetric) **set difference** of s1 and s2. (For example, the set difference of s1 minus s2 is the set containing all of the elements found in s1 but not in s2.)

Example : Identifying unique and duplicate elements

```
import java.util.*;
public class FindDups2 {
    public static void main(String[] args) {
        Integer a[]= {1,2,3,1,1,4,2,5,3,2,1,4,6,7,5,3,2,4,7,8,3,1,10,21,11,10};
        Set<String> uniques = new HashSet<String>();
        Set<String> dups = new HashSet<String>();
        for (Integer i : a){
            if (!uniques.add(i))
                dups.add(i); // Destructive set-difference
        }
        uniques.removeAll(dups);
        System.out.println("Unique elements: " + uniques);
        System.out.println("Duplicate elements: " + dups);
    }
}
```

public class **HashSet<E>** extends AbstractSet<E>
implements Set<E>, Cloneable, Serializable

public class **TreeSet<E>** extends AbstractSet<E>
implements NavigableSet<E>, Cloneable, Serializable

NavigableSet extends SortedSet Interface which provides the nearest value of the set

public class LinkedHashSet<E> extends HashSet<E>
implements Set<E>, Cloneable, Serializable

# Queue Interface

Apart from Collection operations Queue provides additional insertion, extraction and inspection operations.

Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either null or false, depending on the operation).

| **Operation** | **Throws Exception** | **Return special value** |
|---|---|---|
| Insert | add(e) | offer(e) |
| Remove | Remove() | poll() |
| Examine | element() | peek() |

Queue is not necessarily maintain the elements in the FIFO order

Queue is based on its implementation policy

- FIFO Queue
- Priority Queues (maintains natural order or comparator based)
- LIFO Queue(or Stack)

# Deque

A linear collection that supports element insertion and removal at both ends.

This interface defines methods to access the elements at both ends of the deque.

Methods are provided to insert, remove, and examine the element. Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either null or false, depending on the operation).

| | First Element (Head) | | Last Element (Tail) | |
|---|---|---|---|---|
| | *Throws exception* | *Special value* | *Throws exception* | *Special value* |
| **Insert** | `addFirst(e)` | `offerFirst(e)` | `addLast(e)` | `offerLast(e)` |
| **Remove** | `removeFirst()` | `pollFirst()` | `removeLast()` | `pollLast()` |
| **Examine** | `getFirst()` | `peekFirst()` | `getLast()` | `peekLast()` |

This interface extends the **Queue** interface.
When a deque is used as a queue, FIFO (First-In-First-Out) behavior results.

Comparison of Queue and Deque methods

| Queue Method | Equivalent Deque Method |
|---|---|
| add(e) | addLast(e) |
| offer(e) | offerLast(e) |
| remove() | removeFirst() |
| poll() | pollFirst() |
| element() | getFirst() |
| peek() | peekFirst() |

Deques can also be used as LIFO (Last-In-First-Out) stacks.
When a deque is used as a stack, elements are pushed and popped from the beginning of the deque.

Comparison of Stack and Deque methods

| Stack Method | Equivalent Deque Method |
|---|---|
| push(e) | addFirst(e) |
| pop() | removeFirst() |
| peek() | peekFirst() |

# ArrayDeque

public class **ArrayDeque<E>** extends AbstractCollection<E>
implements Deque<E>, Cloneable, Serializable

This class is likely to be faster than Stack class(which extends Vector) when used as a stack, and faster than LinkedList class when used as a queue.

| |
|---|
| **ArrayDeque**()Constructs an empty array deque with an initial capacity sufficient to hold 16 elements. |
| **ArrayDeque**(**Collection**<? extends **E**> c)Constructs a deque containing the elements of the specified collection, in the order they are returned by the collection's iterator. |
| **ArrayDeque**(int numElements)Constructs an empty array deque with an initial capacity sufficient to hold the specified number of elements. |

# PriorityQueue

public class **PriorityQueue<E>**
extends AbstractQueue<E>
implements Serializable

- An unbounded priority queue based on a priority heap.
- The elements of the priority queue are ordered according to their natural ordering or based on the comparator.
- The *head* of this queue is the *least* element with respect to the specified ordering.
- The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.

- It provides O(log(n)) time for the enqueuing and dequeuing methods (offer, poll, remove() and add)
- linear time for the remove(Object) and contains(Object) methods;
- and constant time for the retrieval methods (peek, element, and size).

| |
|---|
| **PriorityQueue**()<br>Creates a PriorityQueue with the default initial capacity (11) that orders its elements according to their **natural ordering**. |
| **PriorityQueue**(**Collection**<? extends **E**> c)<br>Creates a PriorityQueue containing the elements in the specified collection. |
| **PriorityQueue**(**Comparator**<? super **E**> comparator)<br>Creates a PriorityQueue with the default initial capacity and whose elements are ordered according to the specified comparator. |
| **PriorityQueue**(int initialCapacity)<br>Creates a PriorityQueue with the specified initial capacity that orders its elements according to their **natural ordering**. |
| **PriorityQueue**(int initialCapacity, **Comparator**<? super **E**> comparator)<br>Creates a PriorityQueue with the specified initial capacity that orders its elements according to the specified comparator. |
| **PriorityQueue**(**PriorityQueue**<? extends **E**> c)<br>Creates a PriorityQueue containing the elements in the specified priority queue. |
| **PriorityQueue**(**SortedSet**<? extends **E**> c)<br>Creates a PriorityQueue containing the elements in the specified sorted set. |