1. List

A list is an ordered collection that allows duplicates.

- **Implementations: ArrayList, LinkedList, Vector**

Common Methods:

- add(E element): Adds an element to the list.
- get(int index): Returns the element at the specified index.
- remove(int index): Removes the element at the specified index.
- contains(Object o): Returns true if the list contains the specified element.
- size(): Returns the number of elements in the list.
- indexOf(Object o): Returns the index of the first occurrence of the specified element.
- set(int index, E element): Replaces the element at the specified index with the specified element.
- isEmpty(): Returns true if the list is empty.

```java
import java.util.ArrayList;
import java.util.List;

public class ListExample {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();

        // Adding elements
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Accessing elements
        System.out.println("Element at index 1: " + list.get(1));  // Output: Banana

        // Removing an element
        list.remove(0);  // Removes "Apple"

        // Iterating through the list
        for (String fruit : list) {
            System.out.println(fruit);  // Output: Banana, Cherry
        }
    }
}
```

2. Set

A set is a collection that does not allow duplicates.

- **Implementations: HashSet, LinkedHashSet, TreeSet**

Common Methods:

- add(E element): Adds an element to the set.
- contains(Object o): Returns true if the set contains the specified element.
- remove(Object o): Removes the specified element from the set.
- size(): Returns the number of elements in the set.
- isEmpty(): Returns true if the set is empty.
- iterator(): Returns an iterator over the elements in the set.

2.1. TreeSet

A Set that orders its elements according to their natural ordering or a custom comparator.

Common Methods:

- add(E element): Adds the element.
- first(), last(): Returns the first or last element.
- ceiling(E e), floor(E e): Returns the least element greater than or equal to, or the greatest element less than or equal to the given element.

```java
import java.util.HashSet;
import java.util.Set;

public class SetExample {
    public static void main(String[] args) {
        Set<Integer> set = new HashSet<>();

        // Adding elements
        set.add(1);
        set.add(2);
        set.add(3);

        // Duplicate elements won't be added
        set.add(2);

        // Checking if an element exists
        System.out.println("Contains 2? " + set.contains(2));  // Output: true

        // Removing an element
        set.remove(1);

        // Iterating through the set
        for (Integer number : set) {
            System.out.println(number);  // Output: 2, 3
        }
    }
}
```

3. Map

A map is a collection that maps keys to values, with no duplicate keys allowed.

- **Implementations: HashMap, LinkedHashMap, TreeMap, Hashtable**

Common Methods:

- put(K key, V value): Associates the specified value with the specified key.
- get(Object key): Returns the value to which the specified key is mapped.
- remove(Object key): Removes the mapping for the specified key.
- containsKey(Object key): Returns true if the map contains the specified key.
- containsValue(Object value): Returns true if the map contains the specified value.
- size(): Returns the number of key-value mappings.
- keySet(): Returns a set view of the keys.
- values(): Returns a collection view of the values.

3.1. LinkedHashMap

A combination of HashMap and a linked list that maintains the order of elements.

Common Methods:

- put(K key, V value): Adds a key-value pair while maintaining insertion order.
- get(Object key): Returns the value for the specified key.
- remove(Object key): Removes the key-value pair.
- keySet(): Returns the set of keys in order.

```java
import java.util.HashMap;
import java.util.Map;

public class MapExample {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();

        // Adding key-value pairs
        map.put("Apple", 1);
        map.put("Banana", 2);
        map.put("Cherry", 3);

        // Accessing a value by key
        System.out.println("Value for Banana: " + map.get("Banana"));  // Output: 2

        // Removing a key-value pair
        map.remove("Apple");

        // Iterating through the map
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

4. Queue

A queue is a collection designed for holding elements prior to processing (FIFO order).

- **Implementations: LinkedList, PriorityQueue, ArrayDeque**

Common Methods:

- add(E element): Inserts the specified element into the queue.
- offer(E element): Inserts the element and returns true if successful.
- poll(): Retrieves and removes the head of the queue, or returns null if empty.
- peek(): Retrieves the head of the queue without removing it, or returns null if empty.
- remove(): Removes the head of the queue.

```java
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        // Adding elements
        queue.add("First");
        queue.add("Second");
        queue.add("Third");

        // Peek at the front element
        System.out.println("Peek: " + queue.peek());  // Output: First

        // Removing an element
        System.out.println("Removed: " + queue.poll());  // Output: First

        // Iterating through the queue
        for (String element : queue) {
            System.out.println(element);  // Output: Second, Third
        }
    }
}
```

6. Stack

A stack is a collection that follows LIFO (Last-In-First-Out) order.

- **Implementation: Stack**

Common Methods:

- push(E element): Pushes an element onto the stack.
- pop(): Removes and returns the element at the top of the stack.
- peek(): Returns the element at the top of the stack without removing it.
- empty(): Checks if the stack is empty.
- search(Object o): Returns the 1-based position of an element in the stack.

```java
import java.util.Stack;

public class StackExample {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        // Pushing elements onto the stack
        stack.push(1);
        stack.push(2);
        stack.push(3);

        // Peek at the top element
        System.out.println("Peek: " + stack.peek());  // Output: 3

        // Popping elements
        System.out.println("Popped: " + stack.pop());  // Output: 3

        // Checking if the stack is empty
        System.out.println("Is stack empty? " + stack.empty());  // Output: false
    }
}
```

7. Deque (Double-ended Queue)

A deque is a linear collection that supports insertion and removal at both ends.

- **Implementations: ArrayDeque, LinkedList**

Common Methods:

- addFirst(E e), addLast(E e): Inserts an element at the front or rear of the deque.
- removeFirst(), removeLast(): Removes the first or last element of the deque.
- getFirst(), getLast(): Retrieves but does not remove the first or last element.
- pollFirst(), pollLast(): Retrieves and removes the first or last element.
- peekFirst(), peekLast(): Retrieves the first or last element without removing it.

```java
import java.util.ArrayDeque;
import java.util.Deque;

public class DequeExample {
    public static void main(String[] args) {
        Deque<String> deque = new ArrayDeque<>();

        // Adding elements at both ends
        deque.addFirst("First");
        deque.addLast("Last");

        // Peek at both ends
        System.out.println("First element: " + deque.peekFirst());  // Output: First
        System.out.println("Last element: " + deque.peekLast());    // Output: Last

        // Removing elements
        deque.removeFirst();
        deque.removeLast();

        System.out.println("Deque size: " + deque.size());  // Output: 0
    }
}
```

8. PriorityQueue

A queue where elements are ordered based on their priority.

Common Methods:

- add(E element): Adds the specified element.
- offer(E element): Adds the element if possible.
- peek(): Retrieves the head of the queue without removing it.
- poll(): Retrieves and removes the head of the queue.
- remove(Object o): Removes a single instance of the specified element.

```java
import java.util.PriorityQueue;

public class PriorityQueueExample {
    public static void main(String[] args) {
        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>();

        // Adding elements
        priorityQueue.add(5);
        priorityQueue.add(1);
        priorityQueue.add(3);

        // Accessing elements in priority order
        System.out.println("Peek: " + priorityQueue.peek());  // Output: 1

        // Removing elements
        System.out.println("Poll: " + priorityQueue.poll());  // Output: 1
        System.out.println("Poll: " + priorityQueue.poll());  // Output: 3
    }
}
```