**Collection in Java**

**What is Collection?**

A Collection represents a single unit of objects, i.e., a group.

## Advantages of the Java Collection Framework

The Java Collections Framework offers significant advantages that enhance development practices, code quality, and application performance:

1. **Reusability:** The framework provides a comprehensive set of common classes and utility methods applicable across various types of collections. This feature promotes code reusability, sparing developers the need to write duplicate code for common operations.
2. **Quality:** Leveraging the Java Collections Framework elevates the quality of programs. The components within the framework have been extensively tested and are widely used by a vast community of developers, ensuring reliability and stability in your applications.
3. **Speed:** Developers often report an increase in development speed when using the Collections Framework. It allows them to concentrate on the core business logic of their applications rather than on implementing generic collection functionalities, thus speeding up the development process.
4. **Maintenance:** The open-source nature of the Java Collections Framework, coupled with readily available API documentation, facilitates easier code maintenance. Code written using the framework can be easily understood and taken over by other developers, ensuring continuity and ease of maintenance.

**Reduces Effort to Design New APIs:** An additional benefit is the reduced necessity for API designers and implementers to create new collection mechanisms for each new API. They can instead rely on the standard collection interfaces provided by the framework, streamlining the API development process and ensuring consistency across Java applications

Hierarchy of Collection Framework

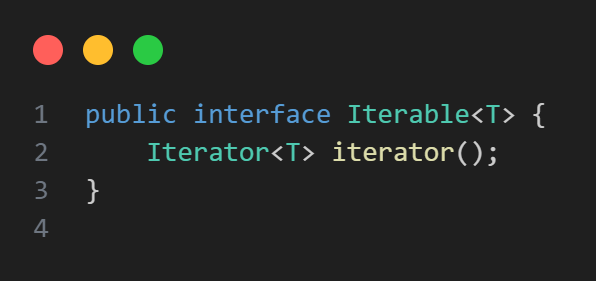


**1-Iterable**

## What is Iterable in Java?

Iterable is an interface in the java.lang package that represents a collection of elements that can be iterated (looped through) one by one.

## General Signature:



## Why use Iterable?

* It allows an object to be the target of the **enhanced for-loop** (also called the *for-each loop*).
* Collections like List, Set, Queue implement Iterable, so you can iterate over them easily.

## Methods of Iterable

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **forEach​(Consumer<? super T> action)** | **Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.** |
| **iterator()** | **Returns an iterator over elements of type T.** |
| **spliterator()** | **Creates a Spliterator over the elements described by this Iterable.** |

**2-Collection**

## Collection Interface in Java

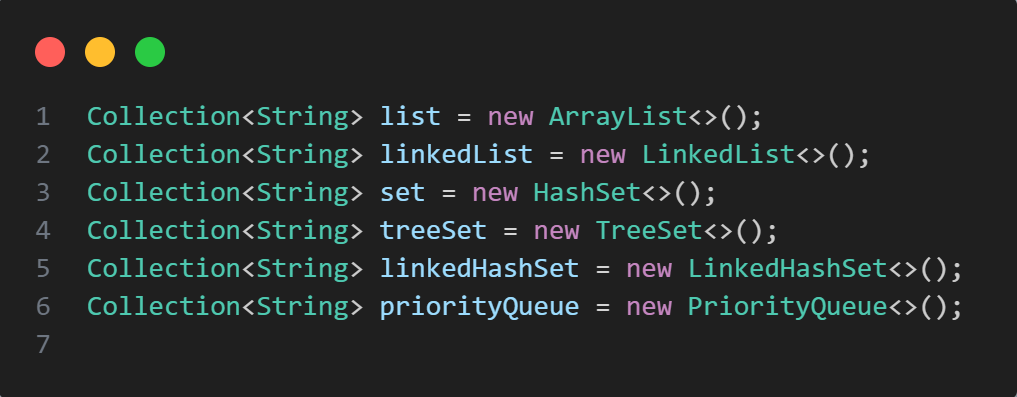
The Collection interface, in the java.util package, is the root interface for representing a group of objects, known as elements. It is the foundation for data structures like Lists, Sets, and Queues.

* **Represents a group of objects which can be manipulated (add, remove, check, etc.).**
* **Extends Iterable, so it supports iteration via for-each loops.**
* **Provides many methods to manage elements such as add(), remove(), size(), clear(), contains(), etc**.

## Methods of Collection

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **boolean add(E e)** | **Adds an element to the collection.** |
| **boolean remove(Object o)** | **Removes an element from the collection.** |
| **int size()** | **Returns the number of elements in the collection.** |
| **boolean contains(Object o)** | **Checks if the collection contains a specific element.** |
| **boolean isEmpty()** | **Checks if the collection is empty.** |
| **Iterator<E> iterator()** | **Returns an iterator over the elements.** |

## Initialization of Collection Interface in Java



**3-List**

## List Interface in Java

The List interface in java.util represents an ordered collection (also called a sequence). Lists allow duplicate elements and provide precise control over where each element is inserted. Elements in a list can be accessed by their integer index (position).

## List characteristics:

* **Ordered: Maintains insertion order.**
* **Indexed Access: Elements can be accessed by their position (index).**
* **Allows Duplicates: The same element can appear multiple times.**

## Methods of List

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **void add(int index, E element)** | **Inserts element at a specified position** |
| **boolean add(E e)** | **Adds element to the end of the list** |
| **E get(int index)** | **Retrieves element at the specified index** |
| **E remove(int index)** | **Removes element at the specified index** |
| **boolean remove(Object o)** | **Removes the first occurrence of the specified element** |
| **int size()** | **Returns the number of elements in the list** |
| **int indexOf(Object o)** | **Returns the index of the first occurrence of element** |
| **int lastIndexOf(Object o)** | **Returns the index of the last occurrence of element** |
| **E set(int index, E element)** | **Replaces the element at the specified index** |
| **List<E> subList(int fromIndex, int toIndex)** | **Returns a view of a portion of the list** |

## Initialization of List Interface in Java



**Complexity of List Interface in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **Adding Element in List Interface** | O(1) | O(1) |
| **Remove Element from List Interface** | O(N) | O(N) |
| **Replace Element in List Interface** | O(N) | O(N) |
| **Traversing List Interface** | O(N) | O(N) |

**4-LikedList**

### **1. What is a LinkedList?**

* **It’s a data structure consisting of nodes where each node contains data and a reference to the next node (and the previous node in a doubly linked list).**
* **Unlike arrays, elements are not stored in contiguous memory locations.**
* **Allows efficient insertion and deletion, especially at the beginning or middle of the list.**

### **2. Where is it in Java?**

* **Part of the java.util package.**
* **Implements the List, Deque, and Queue interfaces.**
* **Can be used as a List, Queue, or Deque.**

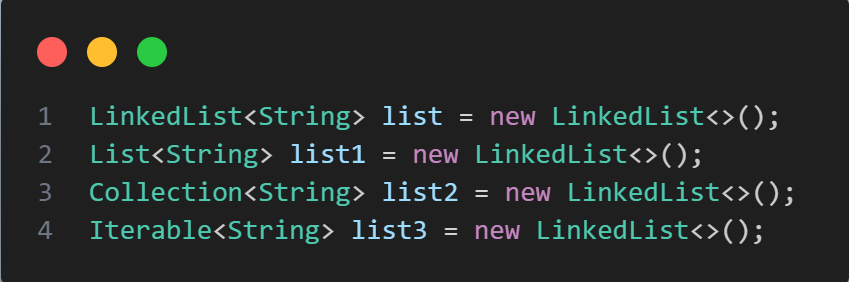
### **3. Advantages:**

* **Fast insertion and deletion at the start, end, or middle.**
* **Supports iteration over elements.**
* **Slower than ArrayList for random access because it requires traversing nodes sequentially.**

## Methods of LinkedList

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Adds an element at the end of the list** |
| **addFirst(E e)** | **Adds an element at the beginning** |
| **addLast(E e)** | **Adds an element at the end** |
| **remove()** | **Removes the first element** |
| **removeFirst()** | **Removes the first element** |
| **removeLast()** | **Removes the last element** |
| **getFirst()** | **Returns the first element without removing** |
| **getLast()** | **Returns the last element without removing** |
| **contains(Object o)** | **Checks if the list contains an element** |
| **size()** | **Returns the number of elements** |

## Initialization of LikedList Class in Java



**Complexity of LinkedList Class in Java**

| **Operation** | **Time Complexity** | **Auxiliary Space** | **Explanation** |
| --- | --- | --- | --- |
| [**Insertion at Beginning**](https://www.geeksforgeeks.org/insert-a-node-at-front-beginning-of-a-linked-list/) | O(1) | O(1) | Constant-time pointer updates. |
| [**Insertion at End**](https://www.geeksforgeeks.org/insert-node-at-the-end-of-a-linked-list/) | O(n) | O(1) | Traversal required to find the last node. |
| [**Insertion at Position**](https://www.geeksforgeeks.org/insert-a-node-at-a-specific-position-in-a-linked-list/) | O(n) | O(1) | Traversal to the desired position, then constant-time pointer updates. |
| [**Deletion at Beginning**](https://www.geeksforgeeks.org/remove-first-node-of-the-linked-list/) | O(1) | O(1) | Constant-time pointer update. |
| [**Deletion at End**](https://www.geeksforgeeks.org/remove-last-node-of-the-linked-list/) | O(n) | O(1) | Traversal required to find the second last node. |
| [**Deletion at Position**](https://www.geeksforgeeks.org/delete-a-linked-list-node-at-a-given-position/) | O(n) | O(1) | Traversal to the desired position, then constant-time pointer updates. |
| [**Searching in Linked list**](https://www.geeksforgeeks.org/search-an-element-in-a-linked-list-iterative-and-recursive/) | O(n) | O(1) | Traversal through the list to find the desired value. |

**5- ArrayList**

### **What is ArrayList in Java?**

* **ArrayList is a resizable array implementation of the List interface.**
* **It allows dynamic resizing, meaning it can grow or shrink as needed.**
* **Elements are stored in a contiguous memory block internally, so it supports fast random access.**
* **It permits duplicate elements and maintains the insertion order.**

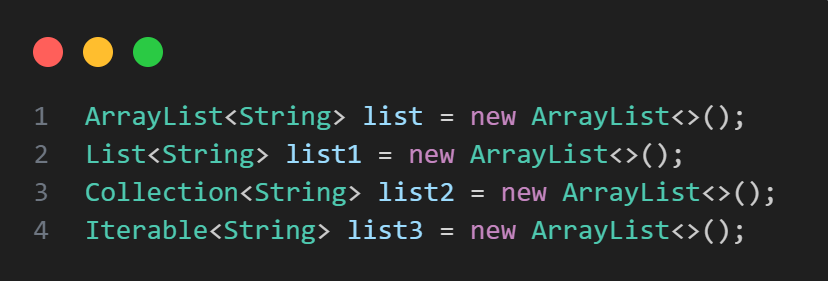
### **Key Features**

* **Implements the List interface.**
* **Supports fast access (get(index)) since it uses an array internally.**
* **Adding elements at the end is generally fast (amortized O(1)).**
* **Inserting or deleting elements in the middle can be slow (O(n)) due to shifting.**
* **Not synchronized by default (not thread-safe).**

## Methods of ArrayList

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Adds an element at the end** |
| **add(int index, E e)** | **Inserts element at specified position** |
| **get(int index)** | **Returns element at specified position** |
| **remove(int index)** | **Removes element at specified position** |
| **remove(Object o)** | **Removes first occurrence of specified element** |
| **size()** | **Returns number of elements** |
| **contains(Object o)** | **Checks if element exists** |
| **clear()** | **Removes all elements** |

## Initialization of ArrayList Class in Java



**Complexity of ArrayList Class in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **Inserting Element in ArrayList** | O(1) | O(N) |
| **Removing Element from ArrayList** | O(N) | O(1) |
| **Traversing Elements in ArrayList** | O(N) | O(N) |
| **Replacing Elements in ArrayList** | O(1) | O(1) |

**6- Vector**

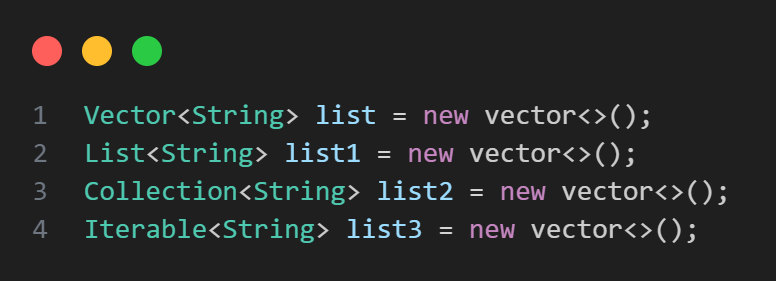
### **Vector Class in Java**

* **Vector is a legacy class in Java that implements a dynamic array.**
* **It is very similar to ArrayList but synchronized, meaning it is thread-safe.**
* **Because of synchronization, Vector operations are slower than ArrayList in single-threaded environments.**
* **It implements the List interface.**
* **You can add, remove, and access elements by index.**
* **It automatically resizes itself when capacity is exceeded.**
* **Commonly used in legacy code, but nowadays ArrayList with external synchronization is preferred**

## Methods of Vector

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Adds an element to the end of the vector** |
| **add(int index, E element)** | **Inserts an element at the specified index** |
| **get(int index)** | **Returns the element at the specified index** |
| **remove(int index)** | **Removes the element at the specified index** |
| **remove(Object o)** | **Removes the first occurrence of the element** |
| **size()** | **Returns the number of elements in the vector** |
| **clear()** | **Removes all elements from the vector** |
| **contains(Object o)** | **Checks if the vector contains the specified element** |
| **isEmpty()** | **Checks if the vector is empty** |
| **indexOf(Object o)** | **Returns the index of the first occurrence of the element** |
| **lastIndexOf(Object o)** | **Returns the index of the last occurrence** |
| **elementAt(int index)** | **Returns the element at the specified index (similar to get)** |
| **firstElement()** | **Returns the first element** |
| **lastElement()** | **Returns the last element** |
| **toArray()** | **Returns an array containing all elements** |
| **trimToSize()** | **Trims the capacity of the vector to its current size** |
| **clone()** | **Creates a shallow copy of the vector** |

## Initialization of Vector Class in Java



**Complexity of Vector Class in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **add(E e)** | **Amortized O(1)** | **O(1)** |
| **add(int index, E element)** | **O(n)** | **O(1)** |
| **get(int index)** | **O(1)** | **O(1)** |
| **remove(int index)** | **O(n)** | **O(1)** |
| **remove(Object o)** | **O(n)** | **O(1)** |
| **contains(Object o)** | **O(n)** | **O(1)** |
| **size()** | **O(1)** | **O(1)** |
| **clear()** | **O(n)** | **O(1)** |
| **indexOf(Object o)** | **O(n)** | **O(1)** |
| **lastIndexOf(Object o)** | **O(n)** | **O(1)** |
| **clone()** | **O(n)** | **O(n)** |
| **trimToSize()** | **O(n)** | **O(n)** |

**7- Stack**

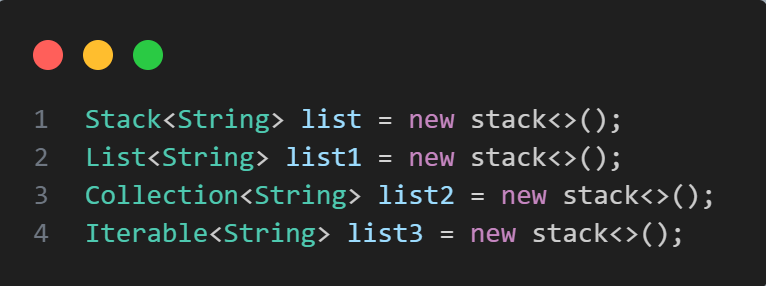
### **Stack Class in Java**

* **The Stack class is part of the java.util package.**
* **It represents a Last-In-First-Out (LIFO) stack of objects.**
* **It extends Vector, so it inherits all Vector methods.**
* **It provides standard stack operations: push, pop, peek, etc.**
* **It’s synchronized (thread-safe) because it inherits from Vector.**
* **Generally, Deque (like ArrayDeque) is preferred nowadays for stack behavior, but Stack is still widely used.**

## Methods of Stack

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **push(E item)** | **Pushes an item onto the top of the stack** |
| **pop()** | **Removes and returns the top item from the stack** |
| **peek()** | **Returns (but does not remove) the top item** |
| **empty()** | **Checks if the stack is empty** |
| **search(Object o)** | **Returns the 1-based position from the top of the stack (or -1 if not found)** |

## Initialization of Stack Class in Java



**Complexity of Stack Class in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **push(E item)** | **O(1) amortized** | **O(1)** |
| **pop()** | **O(1)** | **O(1)** |
| **peek()** | **O(1)** | **O(1)** |
| **empty()** | **O(1)** | **O(1)** |
| **search(Object o)** | **O(n)** | **O(1)** |

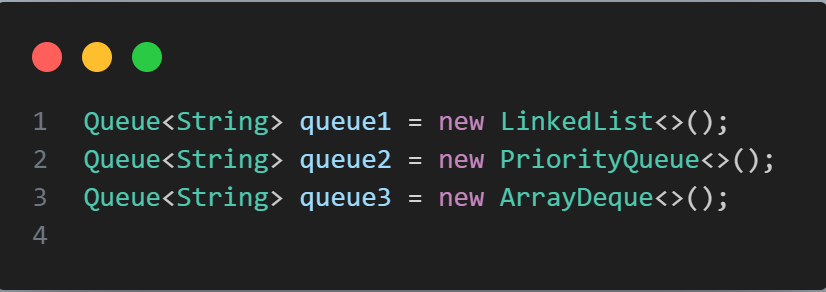
**8-** Queue

* **The Queue<E> interface is part of java.util and represents a collection designed for holding elements prior to processing (FIFO - First In First Out). It defines typical queue operations**
* **used.**

## Methods of Queue

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Inserts the specified element; throws exception if no space** |
| **offer(E e)** | **Inserts the specified element; returns false if no space** |
| **remove()** | **Retrieves and removes the head; throws exception if empty** |
| **poll()** | **Retrieves and removes the head; returns null if empty** |
| **element()** | **Retrieves but does not remove the head; throws exception if empty** |
| **peek()** | **Retrieves but does not remove the head; returns null if empty** |

## Initialization of Queue Interface in Java



**Complexity of Queue Interface in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **add(e) / offer(e)** | **O(1) amortized** | **O(1)** |
| **remove() / poll()** | **O(1)** | **O(1)** |
| **element() / peek()** | **O(1)** | **O(1)** |
| **contains(Object o)** | **O(n)** | **O(1)** |
| **size()** | **O(1)** | **O(1)** |

**9- PriorityQueue**

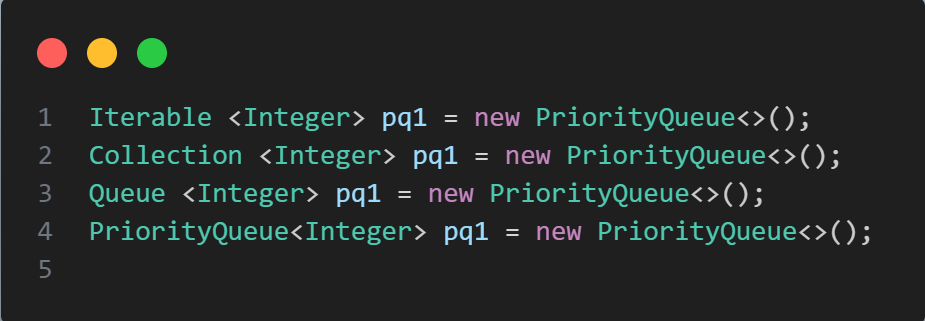
### **PriorityQueue in Java**

* **Implements a priority queue based on a heap data structure (by default a min-heap).**
* **Elements are ordered according to their natural ordering (if they implement Comparable), or by a provided Comparator.**
* **Not synchronized (not thread-safe).**
* **Allows duplicates and null elements are not allowed.**
* **Useful for efficient retrieval of the smallest (or highest priority) element**

## Methods of PriorityQueue

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Inserts the specified element into the priority queue** |
| **offer(E e)** | **Same as add(), inserts an element** |
| **peek()** | **Retrieves, but does not remove, the head (smallest element)** |
| **poll()** | **Retrieves and removes the head of the queue** |
| **remove(Object o)** | **Removes a single instance of the specified element** |
| **contains(Object o)** | **Checks if the queue contains the specified element** |
| **size()** | **Returns the number of elements in the queue** |
| **clear()** | **Removes all elements** |
| **toArray()** | **Returns an array containing all elements** |
| **comparator()** | **Returns the comparator used, or null if natural ordering is used** |

## Initialization of PriorityQueue Interface in Java



**Complexity of PriorityQueue Interface in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| add(E e) / offer(E e) | O(log n) | O(1) |
| peek() | O(1) | O(1) |
| poll() | O(log n) | O(1) |
| remove(Object o) | O(n) | O(1) |
| contains(Object o) | O(n) | O(1) |
| size() | O(1) | O(1) |
| clear() | O(n) | O(1) |

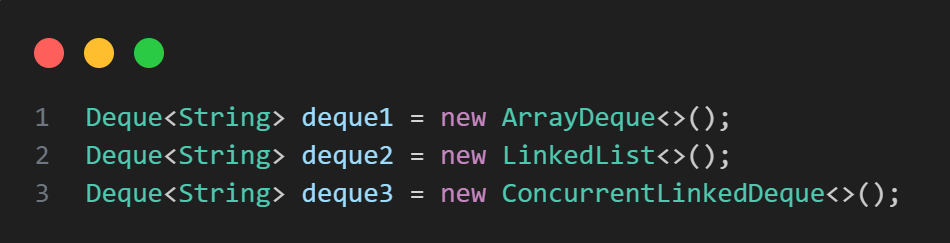
**10- Deque**

**Deque (Double Ended Queue) extends the Queue interface and allows insertion, removal, and inspection at both ends (head and tail).**

## Methods of Deque

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **addFirst(E e)** | **Inserts element at the front (head)** |
| **addLast(E e) / add(E e)** | **Inserts element at the end (tail)** |
| **offerFirst(E e)** | **Inserts element at front, returns false if fails (non-exception)** |
| **offerLast(E e) / offer(E e)** | **Inserts element at end, returns false if fails** |
| **removeFirst()** | **Removes and returns first element** |
| **removeLast()** | **Removes and returns last element** |
| **pollFirst()** | **Removes and returns first element or null if empty** |
| **pollLast()** | **Removes and returns last element or null if empty** |
| **getFirst()** | **Retrieves (without removing) first element** |
| **getLast()** | **Retrieves (without removing) last element** |
| **peekFirst()** | **Retrieves first element or null if empty** |
| **peekLast()** | **Retrieves last element or null if empty** |
| **push(E e)** | **Pushes element at the front (like stack push)** |
| **pop()** | **Pops (removes) element from front (like stack pop)** |
| **size()** | **Returns number of elements** |
| **contains(Object o)** | **Checks if deque contains the element** |
| **clear()** | **Removes all elements** |

## Initialization of Deque Interface in Java



**Complexity of Deque Interface in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **addFirst(E e)** | **O(1)** | **O(1)** |
| **addLast(E e)** | **O(1)** | **O(1)** |
| **removeFirst()** | **O(1)** | **O(1)** |
| **removeLast()** | **O(1)** | **O(1)** |
| **peekFirst() / peekLast()** | **O(1)** | **O(1)** |
| **size()** | **O(1)** | **O(1)** |
| **contains(Object o)** | **O(n)** | **O(1)** |
| **clear()** | **O(n)** | **O(1)** |

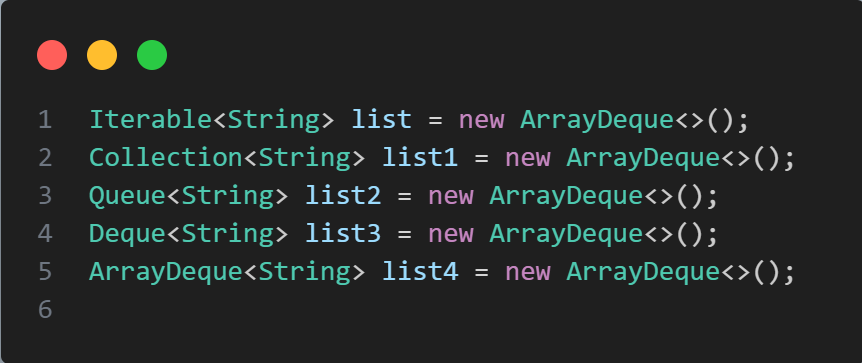
**11- ArrayDeque**

* **ArrayDeque (in java.util) is a resizable-array implementation of the Deque interface. It supports both stack and queue operations efficiently.**

## Methods of ArrayDeque

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Adds element at the tail (end)** |
| **addFirst(E e)** | **Adds element at the head (front)** |
| **addLast(E e)** | **Adds element at the tail (same as add)** |
| **offer(E e)** | **Inserts element at the tail; returns false if no space** |
| **offerFirst(E e)** | **Inserts element at the head** |
| **offerLast(E e)** | **Inserts element at the tail** |
| **remove()** | **Removes and returns head element** |
| **removeFirst()** | **Removes and returns the first element** |
| **removeLast()** | **Removes and returns the last element** |
| **poll()** | **Retrieves and removes the head or returns null if empty** |
| **pollFirst()** | **Retrieves and removes the first element or returns null** |
| **pollLast()** | **Retrieves and removes the last element or returns null** |
| **getFirst()** | **Returns the first element without removing** |
| **getLast()** | **Returns the last element without removing** |
| **peek()** | **Returns the head element or null if empty** |
| **peekFirst()** | **Returns the first element or null if empty** |
| **peekLast()** | **Returns the last element or null if empty** |
| **push(E e)** | **Pushes element at the front (stack behavior)** |
| **pop()** | **Pops element from the front** |
| **size()** | **Returns number of elements** |
| **clear()** | **Removes all elements** |
| **contains(Object o)** | **Checks if the deque contains an element** |
| **iterator()** | **Returns iterator from head to tail** |
| **descendingIterator()** | **Returns iterator from tail to head** |

## Initialization of ArrayDeque Class in Java



**Complexity of ArrayDeque Class in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **add, addFirst, addLast** | **O(1) amortized** | **O(1)** |
| **offer, offerFirst, offerLast** | **O(1) amortized** | **O(1)** |
| **remove, removeFirst, removeLast** | **O(1)** | **O(1)** |
| **poll, pollFirst, pollLast** | **O(1)** | **O(1)** |
| **getFirst, getLast, peek, peekFirst, peekLast** | **O(1)** | **O(1)** |
| **push** | **O(1)** | **O(1)** |
| **pop** | **O(1)** | **O(1)** |
| **size** | **O(1)** | **O(1)** |
| **clear** | **O(n)** | **O(1)** |
| **contains** | **O(n)** | **O(1)** |
| **iterator / descendingIterator** | **O(1) per element** | **O(1)** |

**12- Set**

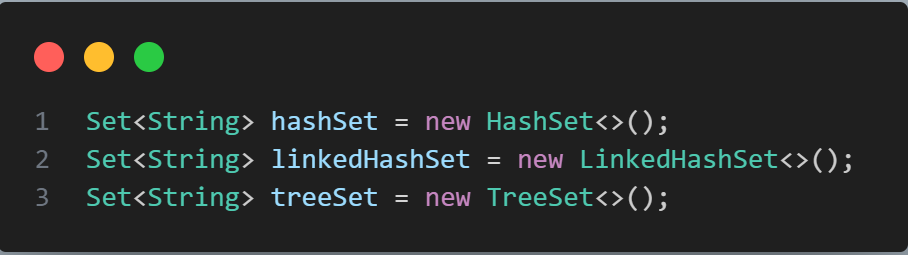
### **. What is Set?**

* **Set is a collection that does not allow duplicate elements.**
* **Defined in the java.util package.**
* **It is an interface, implemented by:**
* **HashSet → unordered, fast**
* **LinkedHashSet → maintains insertion order**
* **TreeSet → sorted, based on natural order or a comparator**

## Methods of Set

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Adds an element if it's not already present** |
| **addAll(Collection c)** | **Adds all elements from another collection** |
| **remove(Object o)** | **Removes a specific element** |
| **contains(Object o)** | **Checks if the set contains the element** |
| **clear()** | **Removes all elements** |
| **size()** | **Returns number of elements** |
| **isEmpty()** | **Checks if the set is empty** |
| **iterator()** | **Returns an iterator over elements** |

## Initialization of Set Interface in Java



**Complexity of Set Interface in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **add()** | **O(1) average** | **O(1)** |
| **remove()** | **O(1) average** | **O(1)** |
| **contains()** | **O(1) average** | **O(1)** |
| **iterator()** | **O(n)** | **O(1)** |
| **clear()** | **O(n)** | **O(1)** |

**13- HashSet**

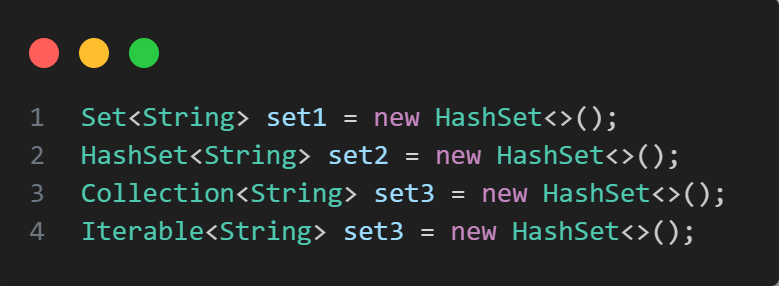
### **What is HashSet?**

* **HashSet is a class in java.util that implements the Set interface using a HashTable (backed by a HashMap). It stores unique elements, allows null, and offers no guaranteed order of elements.**

## Methods of HashSet

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| **add(E e)** | **Adds the element if not already present** |
| **addAll(Collection c)** | **Adds all elements from another collection** |
| **remove(Object o)** | **Removes the element if present** |
| **contains(Object o)** | **Checks if element exists** |
| **clear()** | **Removes all elements** |
| **size()** | **Returns the number of elements** |
| **isEmpty()** | **Checks if the set is empty** |
| **iterator()** | **Returns an iterator over the elements** |
| **clone()** | **Creates a shallow copy of the set** |
| **toArray()** | **Returns an array containing all elements** |

## Initialization of HashSet Class in Java



**Complexity of HashSet Class in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **add(E e)** | **O(1) average, O(n) worst** | **O(1)** |
| **remove(Object o)** | **O(1) average, O(n) worst** | **O(1)** |
| **contains(Object o)** | **O(1) average** | **O(1)** |
| **clear()** | **O(n)** | **O(1)** |
| **size()** | **O(1)** | **O(1)** |
| **isEmpty()** | **O(1)** | **O(1)** |
| **iterator()** | **O(1)** | **O(1)** |
| **clone()** | **O(n)** | **O(n)** |

**14- LikedHashSet**

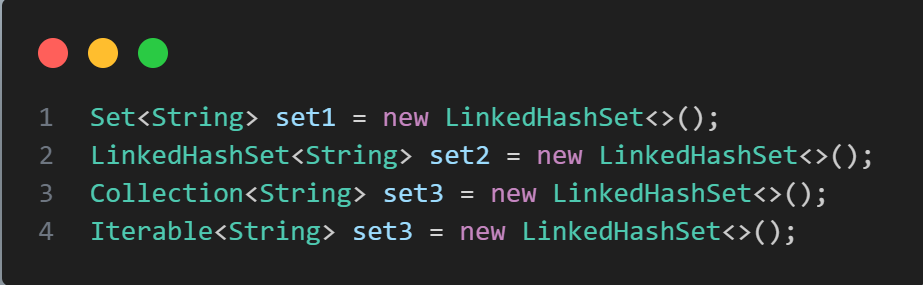
### **What is LinkedHashSet?**

* **LinkedHashSet is a class in java.util that implements the Set interface.**
* **It does not allow duplicate elements.**
* **It maintains insertion order, unlike HashSet.**
* **Internally uses a hash table + linked list.**

## Methods of LinkedHashSet

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| add(E e) | Adds an element if not already present |
| addAll(Collection<? extends E> c) | Adds all elements from another collection |
| remove(Object o) | Removes the specified element |
| clear() | Removes all elements |
| contains(Object o) | Checks if the element exists |
| isEmpty() | Checks if the set is empty |
| size() | Returns the number of elements |
| iterator() | Returns an iterator over the elements (in insertion order) |
| toArray() | Converts the set to an array |

## Initialization of LikedHashSet Class in Java



**Complexity of LikedHashSet Class in Java**

| **Operation** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| **add(E e)** | **O(1) (amortized)** | **O(1)** |
| **remove(Object o)** | **O(1)** | **O(1)** |
| **contains(Object o)** | **O(1)** | **O(1)** |
| **iterator()** | **O(n)** | **O(1)** |
| **clear()** | **O(n)** | **O(1)** |
| **addAll()** | **O(m)** | **O(1)** |
| **add(E e)** | **O(1) (amortized)** | **O(1)** |
| **remove(Object o)** | **O(1)** | **O(1)** |

**Thanks**

**By Mohamed Sayed**