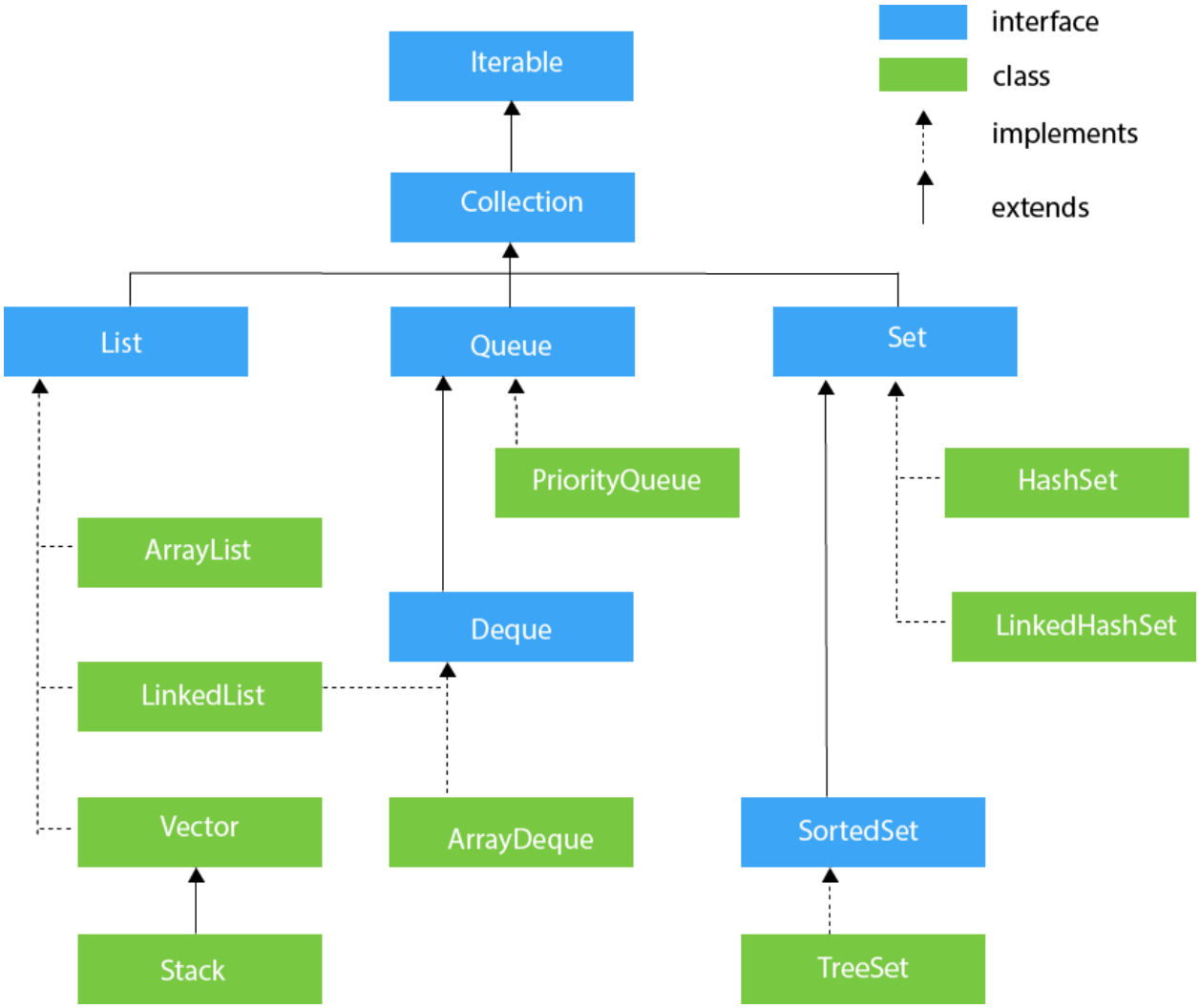
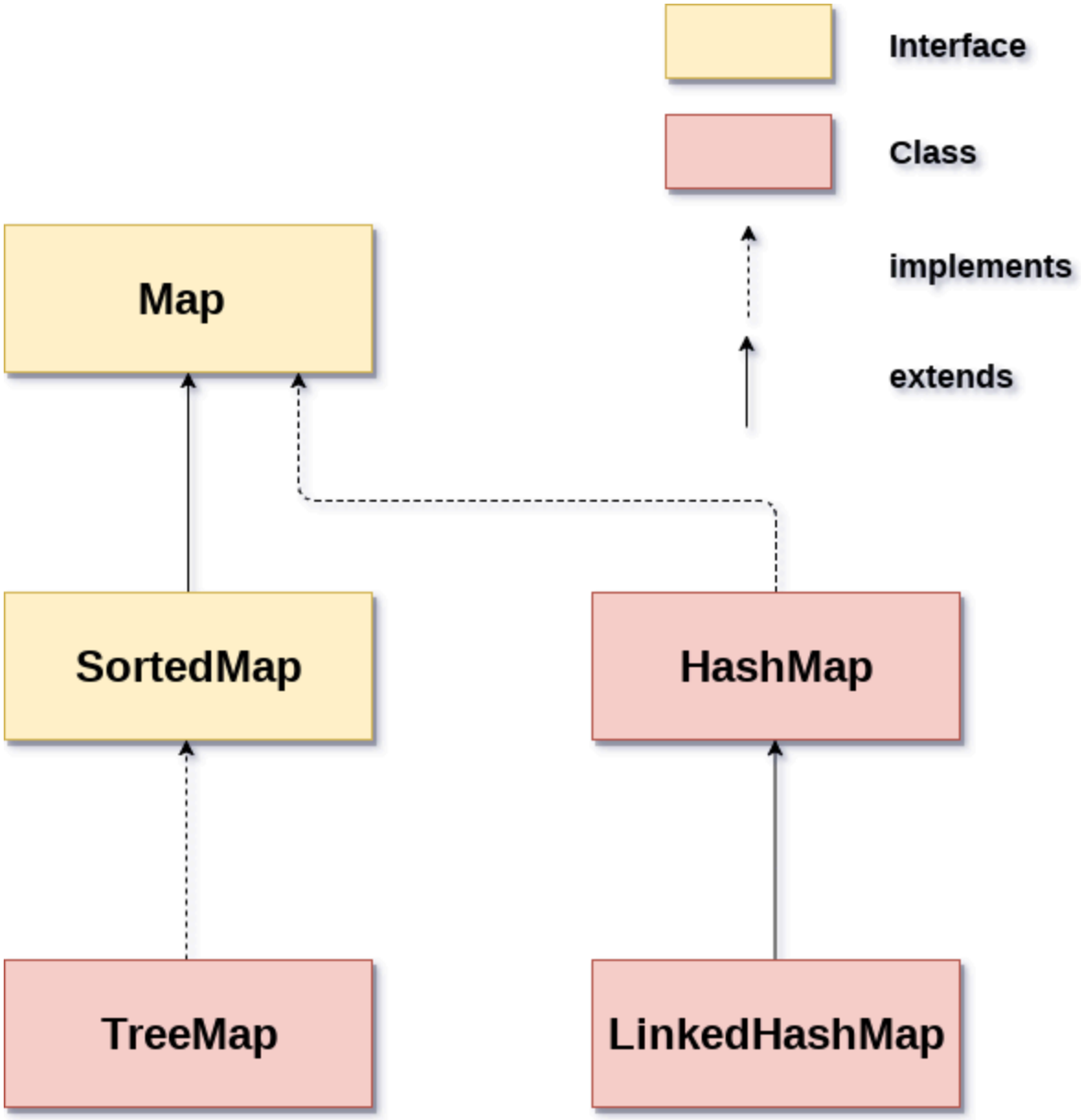
Collections Framework:





Java Collection is a framework that provides a unified architecture to store and manipulate a group of objects.

The Java collection framework is a set of classes and interfaces that provide a standard way of representing and manipulating collections of objects in the Java programming language and implement commonly used data structures, such as lists, sets, and maps.

The Java Collection Framework includes several interfaces and classes.

Collection:

A Collection represents a group of objects, known as its elements. It is an object that can hold references to other objects. The Collection interface is the root of the collection hierarchy. It is the base interface for all collections in the Java Collection Framework. It defines the core methods that all collections must implement, such as add(), remove(), and contains().

As shown above, the `Collection` interface provides a simple and convenient way to perform common operations on a collection of objects. It is often used as a starting point when working with collections in Java.

The Java collection framework includes several interfaces, which define common behaviors for different types of collections. Some of the interfaces which are defined under the `java.util.Collection` interfaces are:

- `java.util.List`
- `java.util.set`
- `java.util.Queue`

List:

It is an ordered collection of objects, where each element has a distinct position in the list. The List interface extends the Collection interface

and adds several methods for working with lists, such as methods for accessing elements by their position in the list, and methods for searching and sorting lists. Lists *can contain duplicate elements,* and their elements can be accessed by their position in the list.

the `List` interface provides a convenient way to work with ordered collections of elements. It is commonly used when you need to maintain the order of elements in a collection, or when you need to access elements by their index in the list.

Set:

Set in the Java Collection Framework is a collection of unique elements, where *duplicate elements are not allowed*. The Set interface extends the Collection interface and adds a few methods for working with sets, such as methods for checking if an element is in a set and methods for adding and removing elements from a set.

the `Set` interface provides a convenient way to work with collections of elements that do not allow duplicate elements. It is commonly used when you need to ensure that a collection only contains unique elements.

Queue:

In the Java collection framework, a queue is a data structure that is used to store elements in a First-In-First-Out (FIFO) manner. This means that the first element that is added to the queue will be the first one to be removed.

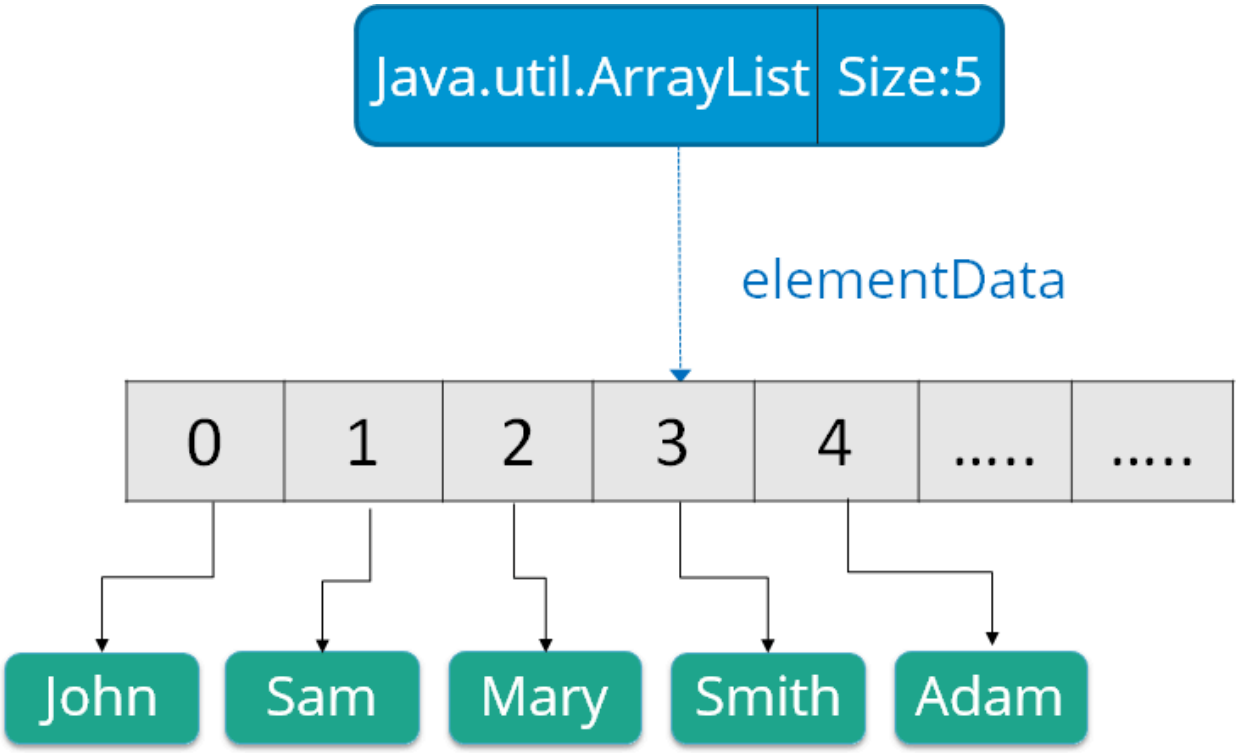
Map:

The `java.util.Map` interface in the Java collection framework is used to represent a mapping of keys to values. It allows storing elements in the form of key-value pairs and provides methods for accessing, modifying, and iterating over the elements in the map.

Array list:

ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. Also, the size of

the list is increased dynamically if the elements are added more than the initial size.



Array List - Java Collections

Syntax:

ArrayList object = new ArrayList ();

Some of the methods in array list are listed below:

Method	Description
boolean add(Collection c)	Appends the specified element to the end of a list.
void add(int index, Object element)	Inserts the specified element at the specified position.
void clear()	Removes all the elements from this list.
int lastIndexOf(Object o)	Return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object clone()	Return a shallow copy of an ArrayList.
Object[] toArray()	Returns an array containing all the elements in the list.
void trimToSize()	Trims the capacity of this ArrayList instance to be the list's current size.

Methods in ArrayList — Java Collections

It uses a dynamic array to store the duplicate element of different data types.
The ArrayList class maintains the insertion order and is non-synchronized.
The elements stored in the ArrayList class can be randomly accessed.

It provides fast random access to its elements, but slow insertion and deletion at arbitrary positions.

Linked List:

Linked List is a sequence of links which contains items. Each link

contains a connection to another link.

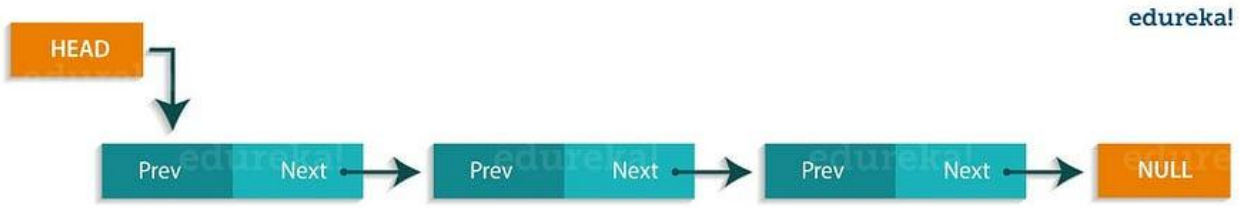
***Syntax*:** Linkedlist object = new Linkedlist();

Java Linked List class uses two types of Linked list to store the elements:

- *Singly Linked List*
- *Doubly Linked List*

Singly Linked List:

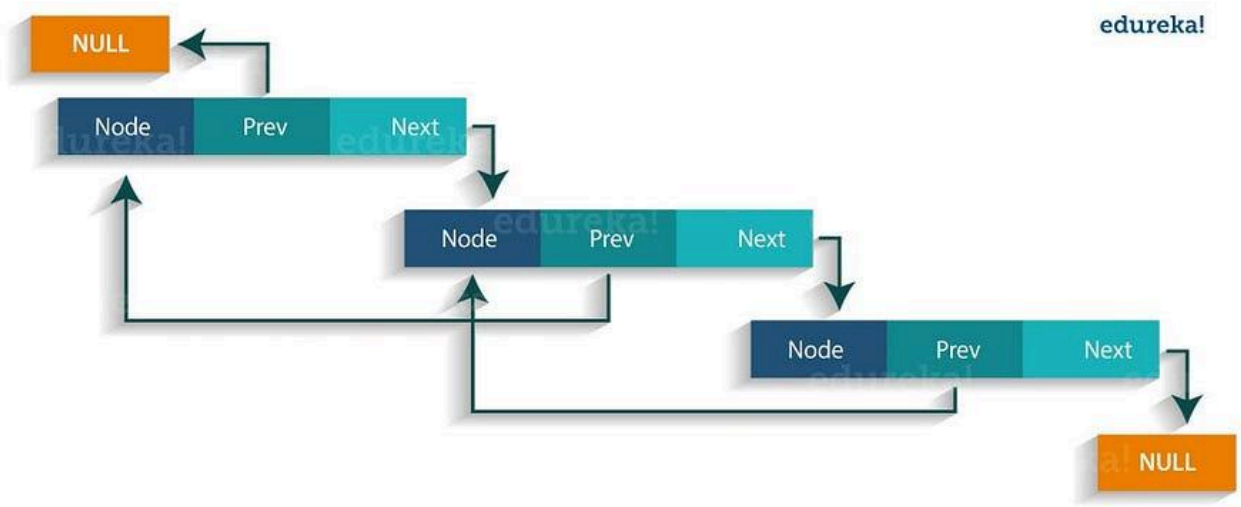
In a singly Linked list, each node in this list stores the data of the node and a pointer or reference to the next node in the list. Refer to the below image to get a better understanding of single Linked list.



Single Linked List - Java Collections

Doubly Linked List:

In a doubly Linked list, it has two references, one to the next node and another to the previous node. You can refer to the below image to get a better understanding of doubly linked list.



Doubly Linked List - Java Collections

Some of the methods in the linked list are listed below:

Method	Description
boolean add(Object o)	It is used to append the specified element to the end of the vector.
boolean contains(Object o)	Returns true if this list contains the specified element.
void add (int index, Object element)	Inserts the element at the specified element in the vector.
void addFirst(Object o)	It is used to insert the given element at the beginning.
void addLast(Object o)	It is used to append the given element to the end.
int size()	It is used to return the number of elements in a list
boolean remove(Object o)	Removes the first occurrence of the specified element from this list.
int indexOf(Object element)	Returns the index of the first occurrence of the specified element in this list, or -1.
int lastIndexOf(Object element)	Returns the index of the last occurrence of the specified element in this list, or -1.

Linked List:

The `java.util.LinkedList` class in the Java collection framework inherits the AbstractList class and implements List and Deque interfaces. It provides efficient methods for adding, removing, and accessing elements at the beginning and end of the list.

Linked List is a sequence of links which contains items. Each link

contains a connection to another link.

***Syntax*:** Linkedlist object = new Linkedlist();

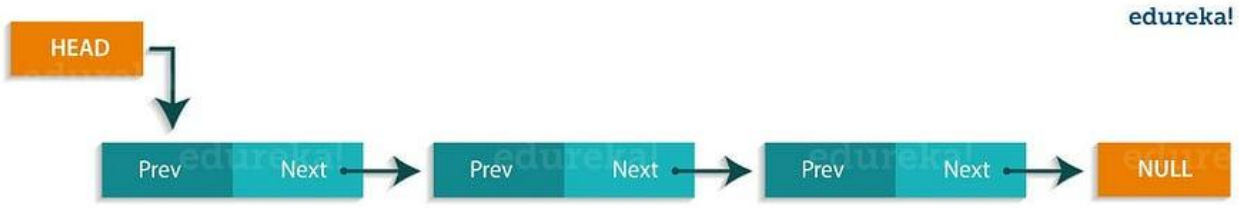
Java Linked List class uses two types of Linked list to store

the elements:

- *Singly Linked List*
- *Doubly Linked List*

Singly Linked List:

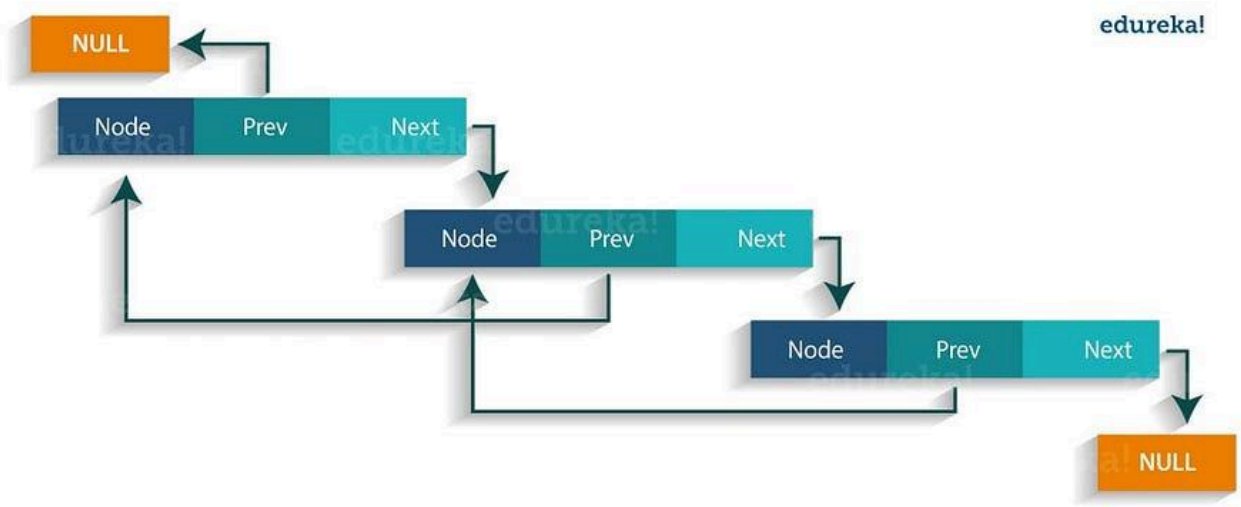
In a singly Linked list, each node in this list stores the data of the node and a pointer or reference to the next node in the list. Refer to the below image to get a better understanding of single Linked list.



Single Linked List - Java Collections

Doubly Linked List:

In a doubly Linked list, it has two references, one to the next node and another to the previous node. You can refer to the below image to get a better understanding of doubly linked list.



Doubly Linked List - Java Collections

Some of the methods in the linked list are listed below:

Method	Description
boolean add(Object o)	It is used to append the specified element to the end of the vector.
boolean contains(Object o)	Returns true if this list contains the specified element.
void add (int index, Object element)	Inserts the element at the specified element in the vector.
void addFirst(Object o)	It is used to insert the given element at the beginning.
void addLast(Object o)	It is used to append the given element to the end.
int size()	It is used to return the number of elements in a list
boolean remove(Object o)	Removes the first occurrence of the specified element from this list.
int indexOf(Object element)	Returns the index of the first occurrence of the specified element in this list, or -1.
int lastIndexOf(Object element)	Returns the index of the last occurrence of the specified element in this list, or -1.

Difference Between ArrayList and LinkedList

ArrayList and LinkedList both implement the List interface and maintain insertion order. Both are non-synchronized classes.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.
5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.

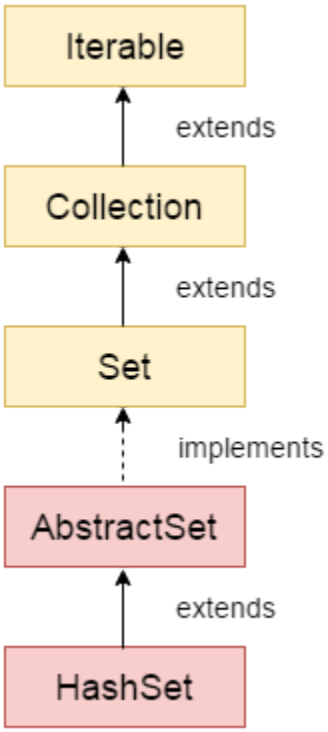
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

- When the rate of addition or removal rate is more than the read scenarios, then go for the LinkedList. On the other hand, when the frequency of the read scenarios is more than the addition or removal rate, then ArrayList takes precedence over LinkedList.
- Since the elements of an ArrayList are stored more compact as compared to a LinkedList; therefore, the ArrayList is more cache-friendly as compared to the LinkedList. Thus, chances for the cache miss are less in an ArrayList as compared to a LinkedList. Generally, it is considered that a LinkedList is poor in cache-locality.
- Memory overhead in the LinkedList is more as compared to the ArrayList. It is because, in a LinkedList, we have two extra links (next and previous) as it is required to store the address of the previous and the next nodes, and these links consume extra space. Such links are not present in an ArrayList.

Java HashSet

The `java.util.HashSet` class in the Java collection framework is used to store a collection of unique elements in a set. It provides a hash table-based implementation of the `java.util.Set` interface.

It provides fast insertion, deletion, and lookup, but does not maintain the order of its elements.



Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.

- The initial default capacity of HashSet is 16, and the load factor is 0.75.

Difference between List and Set

A list can contain duplicate elements whereas Set contains unique elements only.

TreeSet:

The `java.util.TreeSet` class in the Java collection framework is used to store a collection of unique elements in a set that is sorted in ascending order. It provides a red-black tree-based implementation of the `java.util.Set` interface which uses a tree to store the elements and does not allow duplicate elements.

It provides fast insertion, deletion, and lookup, and maintains the order of its elements according to their natural ordering or a comparator.

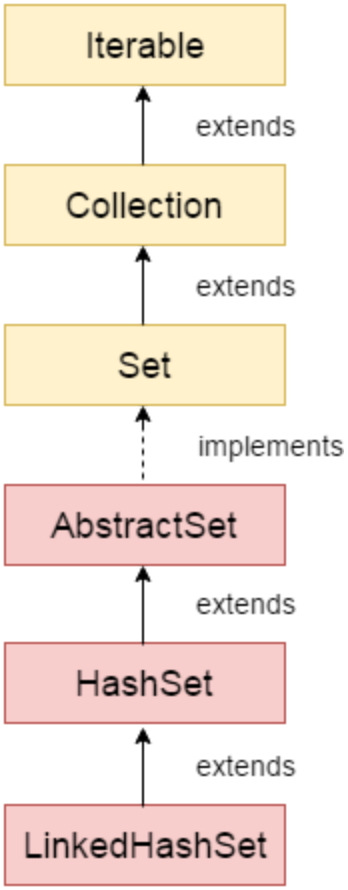
- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.

- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.
- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quite fast.
- Java TreeSet class doesn't allow null elements.
- Java TreeSet class is non-synchronized.
- Java TreeSet class maintains ascending order.
- The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

Internal Working of The TreeSet Class

TreeSet is being implemented using a binary search tree, which is self-balancing just like a Red-Black Tree. Therefore, operations such as a search, remove, and add consume O(log(N)) time. The reason behind this is there in the self-balancing tree. It is there to ensure that the tree height never exceeds O(log(N)) for all of the mentioned operations. Therefore, it is one of the efficient data structures in order to keep the large data that is sorted and also to do operations on it.

Java LinkedHashSet Class



Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

The important points about the Java LinkedHashSet class are:

- Java LinkedHashSet class contains unique elements only like HashSet.
- Java LinkedHashSet class provides all optional set operations and permits null elements.
- Java LinkedHashSet class is non-synchronized.
- Java LinkedHashSet class maintains insertion order.

Java Queue Interface

The interface Queue is available in the java.util package and does extend the Collection interface. It is used to keep the elements that are processed in the First In First Out (FIFO) manner. It is an ordered list of objects, where insertion of elements occurs at the end of the list, and removal of elements occur at the beginning of the list.

Being an interface, the queue requires, for the declaration, a concrete class, and the most common classes are the LinkedList and PriorityQueue in Java. Implementations done by these classes are not thread safe. If it is required to have a thread safe implementation, PriorityBlockingQueue is an available option.

- FIFO concept is used for insertion and deletion of elements from a queue.
- The Java Queue provides support for all of the methods of the Collection interface including deletion, insertion, etc.
- PriorityQueue, ArrayBlockingQueue and LinkedList are the implementations that are used most frequently.
- The NullPointerException is raised, if any null operation is done on the BlockingQueues.
- Those Queues that are present in the *util* package are known as Unbounded Queues.
- Those Queues that are present in the *util.concurrent* package are known as bounded Queues.
- All Queues barring the Deques facilitates removal and insertion at the head and tail of the queue; respectively. In fact, deques support element insertion and removal at both ends.

PriorityQueue Class

PriorityQueue is also class that is defined in the collection framework that gives us a way for processing the objects on the basis of priority. It is already described that the insertion and deletion of objects follows FIFO pattern in the Java queue. However, sometimes the elements of the queue are needed to be processed according to the priority, that's where a PriorityQueue comes into action.

Deque Interface

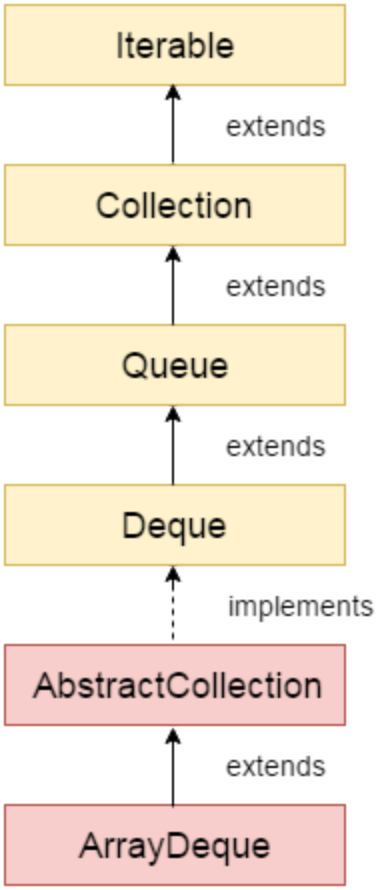
The interface called Deque is present in java.util package. It is the subtype of the interface queue. The Deque supports the addition as well as the removal of elements from both ends of the data structure. Therefore, a deque can be used as a stack or a queue. We know that the stack supports the Last In First Out (LIFO) operation, and the operation First In First Out is supported by a queue. As a deque supports both, either of the mentioned operations can be performed on it. Deque is an acronym for "**double ended queue".**

ArrayDeque class

We know that it is not possible to create an object of an interface in Java. Therefore, for instantiation, we need a class that implements the Deque interface, and that class is ArrayDeque. It grows and shrinks as per usage. It also inherits the AbstractCollection class.

The important points about ArrayDeque class are:

- **Unlike Queue, we can add or remove elements from both sides.**
- **Null elements are not allowed in the ArrayDeque.**
- **ArrayDeque is not thread safe, in the absence of external synchronization.**
- **ArrayDeque has no capacity restrictions.**
- **ArrayDeque is faster than LinkedList and Stack.**


Iterable
extends
Collection
extends
Queue
extends
Deque
implements
AbstractCollection
extends
ArrayDeque