Vector is a resizable array in Java, found in the java.util package. It is part of the Collection Framework and works like an ArrayList, but it is synchronized, meaning it is safe to use in multi-threaded programs. However, this makes it a bit slower than ArrayList.

Key Features of Vector

It expands as elements are added.

The Vector class is synchronized in nature means it is thread-safe by default.

Like an ArrayList, it maintains insertion order.

It allows duplicates and nulls.

It implements List, RandomAccess, Cloneable and Serializable.

Constructors of Vector

1. Vector(): Creates a default vector of the initial capacity is 10.

```
Vector<E> v = new Vector<E>();
```

2. Vector(int size): Creates a vector whose initial capacity is specified by size.

```
Vector<E> v = new Vector<E>(int size);
```

3. Vector(int size, int incr): Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time a vector is resized upward.

```
Vector<E> v = new Vector<E>(int size, int incr);
```

4. Vector(Collection c): Creates a vector that contains the elements of collection c.

```
Vector<E> v = new Vector<E>(Collection c);
```

Different Operations of Vector Class

1. Adding Elements

To add the elements to the Vector, we use the add() method. This method is overloaded to perform multiple operations based on different parameters. They are listed below as follows:

add(Object): This method is used to add an element at the end of the Vector.

add(int index, Object): This method is used to add an element at a specific index in the Vector.

2. Updating Elements

To update an element in a Vector, use the set() method. It takes the index and the new element to replace the existing one at that position.

3. Removing Elements

To remove an element from a Vector, we can use the remove() method. This method is overloaded to perform multiple operations based on different parameters. They are:

remove(Object): This method is used to remove an object from the Vector. If there are multiple such objects, then the first occurrence of the object is removed.

remove(int index): Vector is indexed, so this method takes an integer value which simply removes the element present at that specific index in the Vector. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

4. Iterating the Vector

There are multiple ways to iterate through the Vector. The most famous ways are by using the basic for loop in combination with a get() method to get the element at a specific index and the advanced for a loop.

Deque Interface present in java.util package is a subtype of the queue interface. The Deque is related to the double-ended queue that supports adding or removing elements from either end of the data structure. It can either be used as a queue(first-in-first-out/FIFO) or as a stack(last-in-first-out/LIFO). Deque is the acronym for double-ended queue.

Null Handling: Most implementations do not allow null elements, as null is used as a special return value to indicate the absence of elements.

Thread-Safe Alternatives: Use ConcurrentLinkedDeque or LinkedBlockingDeque for thread-safe operations and avoid ArrayDeque in concurrent environments as it is not thread-safe.

Since Deque is an interface, objects cannot be created of the type deque. We always need a class that extends this list in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the Deque. This type-safe queue can be defined as:

```
// Obj is the type of the object to be stored in Deque
Deque<Obj> deque = new ArrayDeque<> ();
```

1. Adding Elements

In order to add an element in a deque, we can use the add() method. The difference between a queue and a deque is that in deque, the addition is possible from any direction. Therefore, there are other two methods available named addFirst() and addLast() which are used to add the elements at either end.

2. Removing Elements

In order to remove an element from a deque, there are various methods available. Since we can also remove from both ends, the deque interface provides us with removeFirst(), removeLast() methods. Apart from that, this interface also provides us with the poll(), pop(), pollFirst(), pollLast() methods where pop() is used to remove and return the head of the deque. However, poll() is used because this offers the same functionality as pop() and doesn't return an exception when the deque is empty.

3. Iterating through the Deque

Since a deque can be iterated from both directions, the iterator method of the deque interface provides us two ways to iterate. One from the first and the other from the back.

ArrayDeque class which is implemented in the collection framework provides us with a way to apply resizable-array. This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue. Array deques have no capacity restrictions and they grow as necessary to support usage. They are not thread-safe which means that in the absence of external synchronization,

ArrayDeque does not support concurrent access by multiple threads. ArrayDeque class is likely to be faster than Stack when used as a stack. ArrayDeque class is likely to be faster than LinkedList when used as a queue.

A **HashSet** is a collection of elements where every element is unique.

It is part of the java.util package and implements the Set interface.

To add elements to a HashSet, use the add() method:

To check whether an element exists in a HashSet, use the contains() method:

To remove an element, use the remove() method:

Use size() to count how many unique elements are in the set:

Loop through the elements of an HashSet with a for-each loop:

Java LinkedHashSet

A LinkedHashSet is a collection that stores unique elements and remembers the order they were added.

It is part of the java.util package and implements the Set interface.

Use HashSet when you only care about uniqueness and speed. Use LinkedHashSet when order matters.