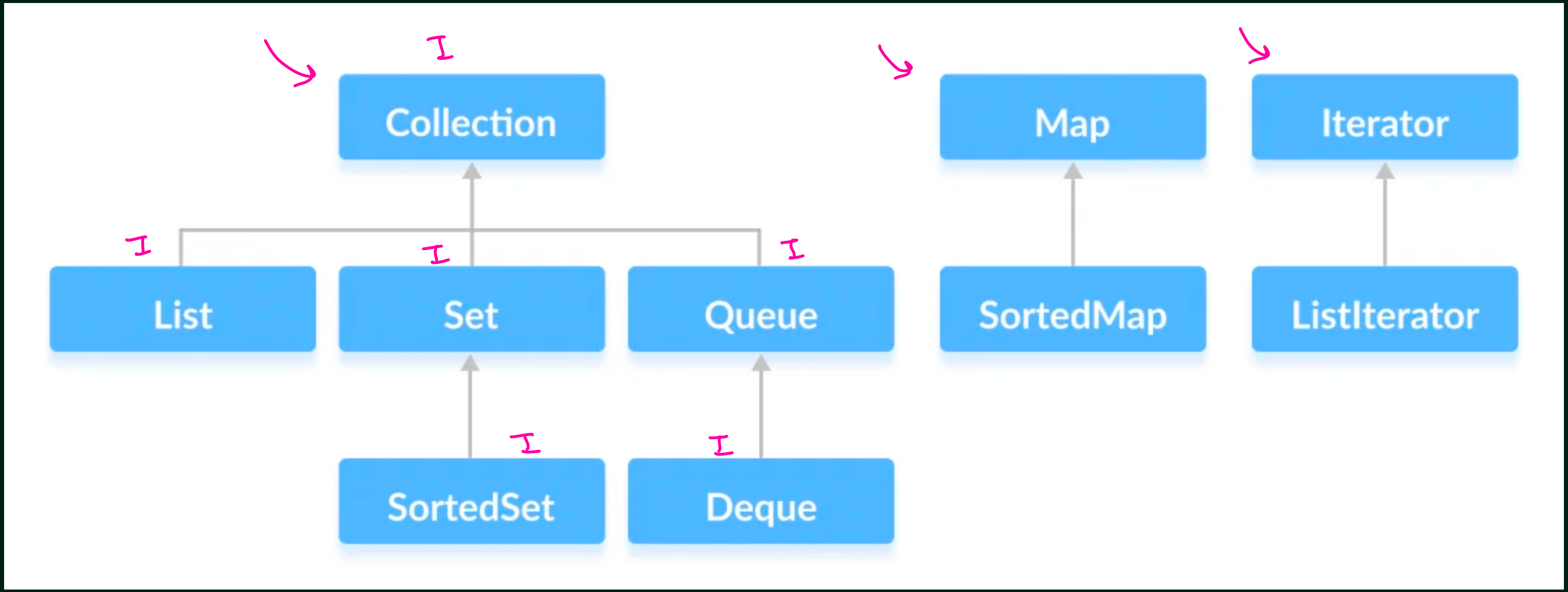
Collection Framework and Lists

In This Lecture

1. Java Collection Framework ✓
2. Java Collection Interface ✓
3. Java List Interface ✓
4. Java ArrayList ✓
5. Java LinkedList ✓
6. Java Vector ✓
7. Java Stack ✓

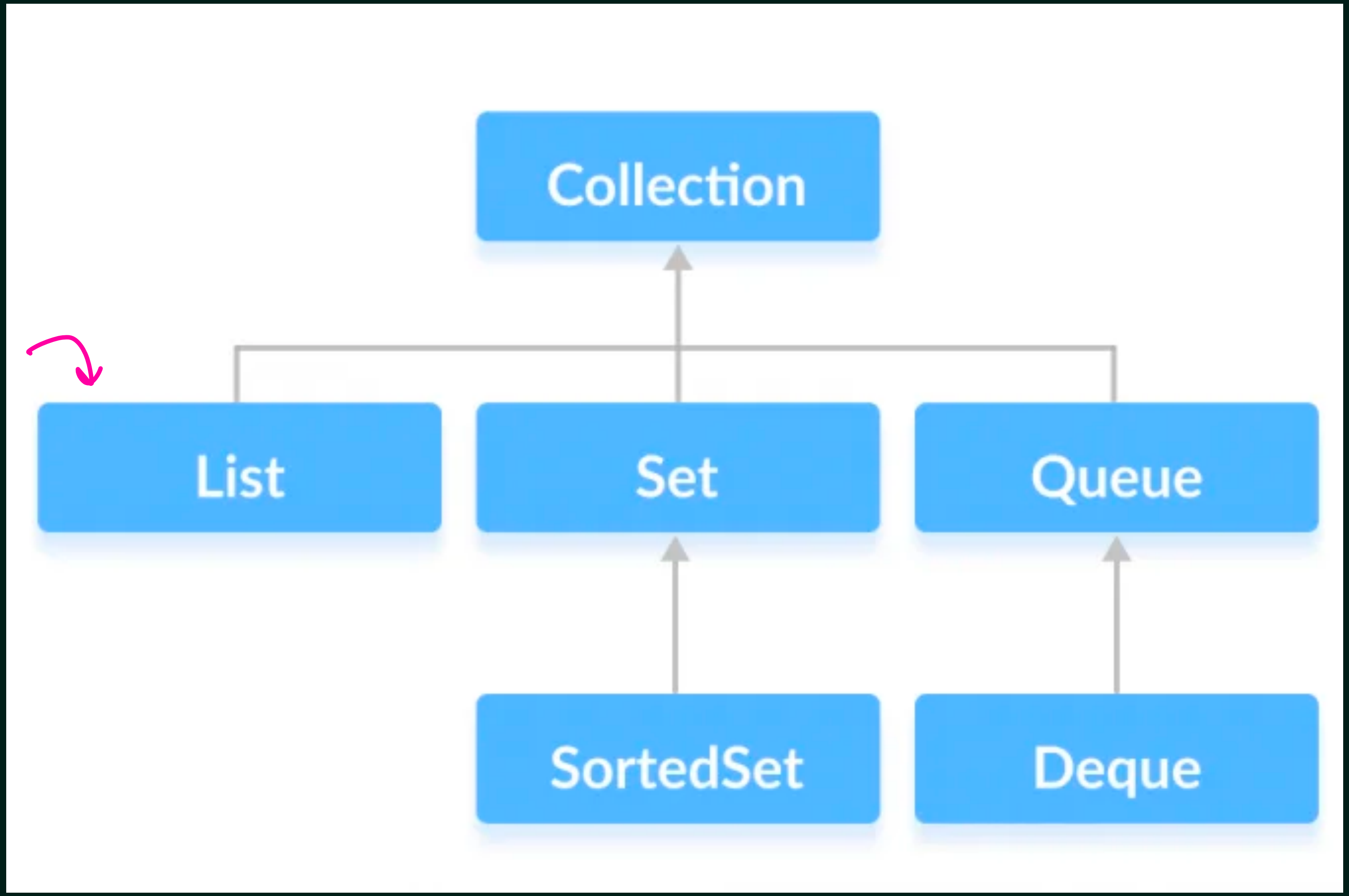
Java Collection Framework

The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms. These interfaces include several methods to perform different operations on collections.



Java Collection Interface

The Collection interface is the root interface of the Java collections framework.



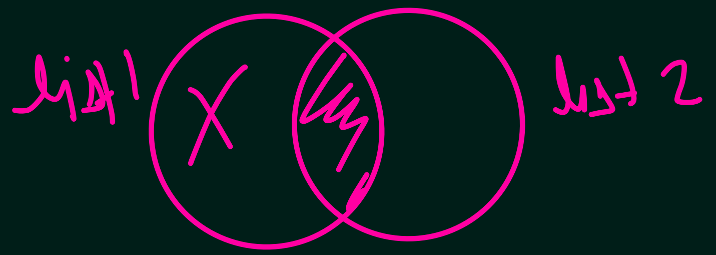
Java Collection Interface

The Collection interface includes various methods that can be used to perform different operations on objects.

- **int size():** Returns the number of elements in the collection.
- **boolean isEmpty():** Returns **true** if the collection contains no elements.
- **boolean contains(Object o):** Returns **true** if the collection contains the specified element.
- **boolean add(E e):** Adds the specified element to the collection. Returns **true** if the collection changed as a result.
- **boolean remove(Object o):** Removes a single instance of the specified element from the collection, if it is present.
- **boolean containsAll(Collection<> c):** Returns **true** if the collection contains all elements of the specified collection.

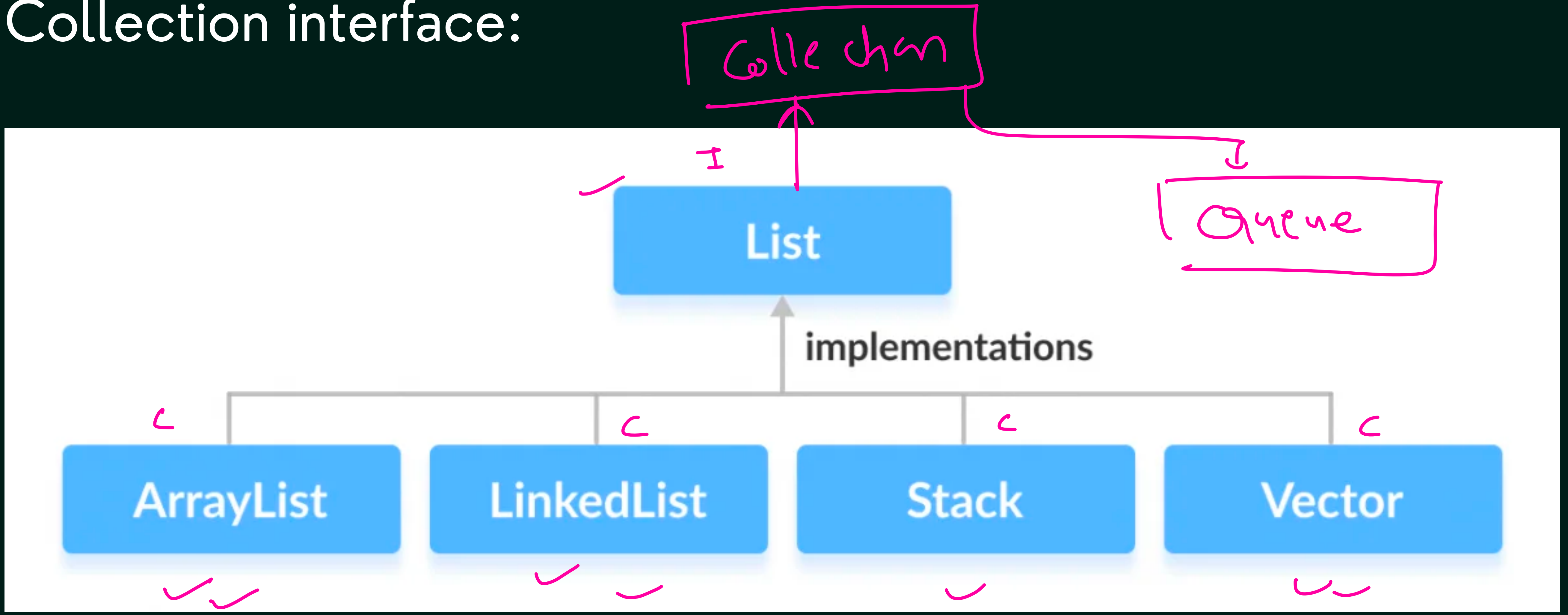
Java Collection Interface

- **boolean addAll(Collection<> c):** Adds all elements from the specified collection to the collection.
- **boolean removeAll(Collection<> c):** Removes all elements in the collection that are also contained in the specified collection.
- **boolean retainAll(Collection<> c):** Removes all elements from the collection that are not present in the specified collection.
- **void clear():** Removes all elements from the collection.
- **Object[] toArray():** Returns an array containing all elements in the collection.



Java List Interface

The List interface extends the Collection interface and adds methods that are specific to lists, which are ordered collections that allow duplicate elements. Here are some methods that are present in the List interface but not in the Collection interface:



Java List Interface

The List interface extends the Collection interface and adds methods that are specific to lists, which are ordered collections that allow duplicate elements. Here are some methods that are present in the List interface but not in the Collection interface:

- **get(int index):** Retrieves the element at the specified index in the list.
- **set(int index, E element):** Replaces the element at the specified index with the given element.
- **add(int index, E element):** Inserts the specified element at the specified position in the list, shifting the current elements to the right.
- **remove(int index):** Removes the element at the specified index from the list and shifts the remaining elements to the left.

Java List Interface

- **indexOf(Object o):** Returns the index of the first occurrence of the specified element in the list, or -1 if the element is not present.
- **lastIndexOf(Object o):** Returns the index of the last occurrence of the specified element in the list, or -1 if the element is not present.
- **listIterator():** Returns a list iterator over the elements in the list.
- **listIterator(int index):** Returns a list iterator over the elements in the list, starting at the specified index.
- **subList(int fromIndex, int toIndex):** Returns a view of the portion of the list between the specified fromIndex (inclusive) and toIndex (exclusive).

Java ArrayList

In Java, we need to declare the size of an array before we can use it. Once the size of an array is declared, it's hard to change it.

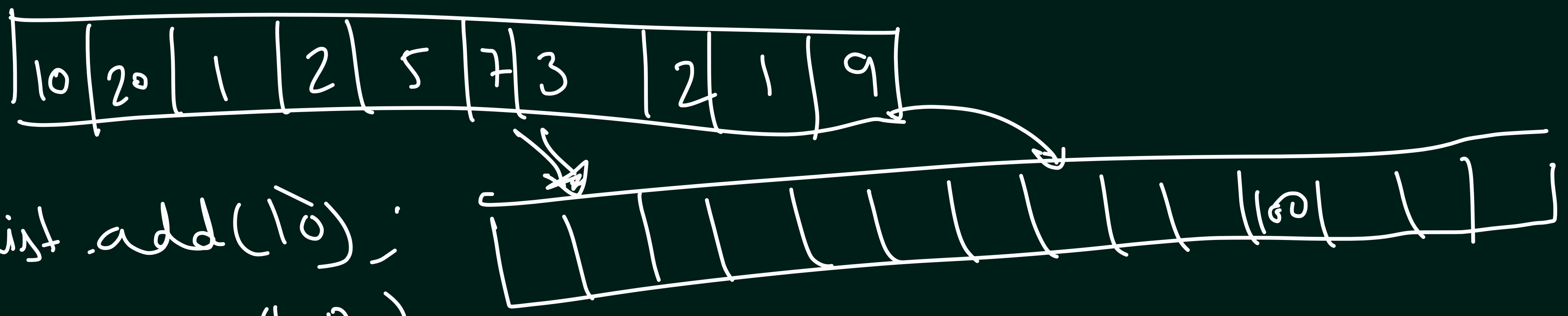
To handle this issue, we can use the ArrayList class. It allows us to create resizable arrays.

Unlike arrays, arraylists can automatically adjust their capacity when we add or remove elements from them. Hence, arraylists are also known as dynamic arrays.

$$newSize = (oldsize \times 3)/2 + 1$$

Arraylist list = new Arraylist<>()

$$(10 \times 3)/2 + 1 = 16$$



list.add(10);

(100)

Internal Working of ArrayList

Initially, the array has a certain capacity, and as elements are added, it fills up. When the capacity is reached, the **ArrayList** creates a new larger array and copies the elements from the old array to the new one. This process of resizing and copying is transparent to the user.

However, frequent resizing operations can lead to performance overhead, so the **ArrayList** increases its capacity by a certain factor to minimize the frequency of resizing.

$10 \rightarrow 11 \rightarrow 12$ X

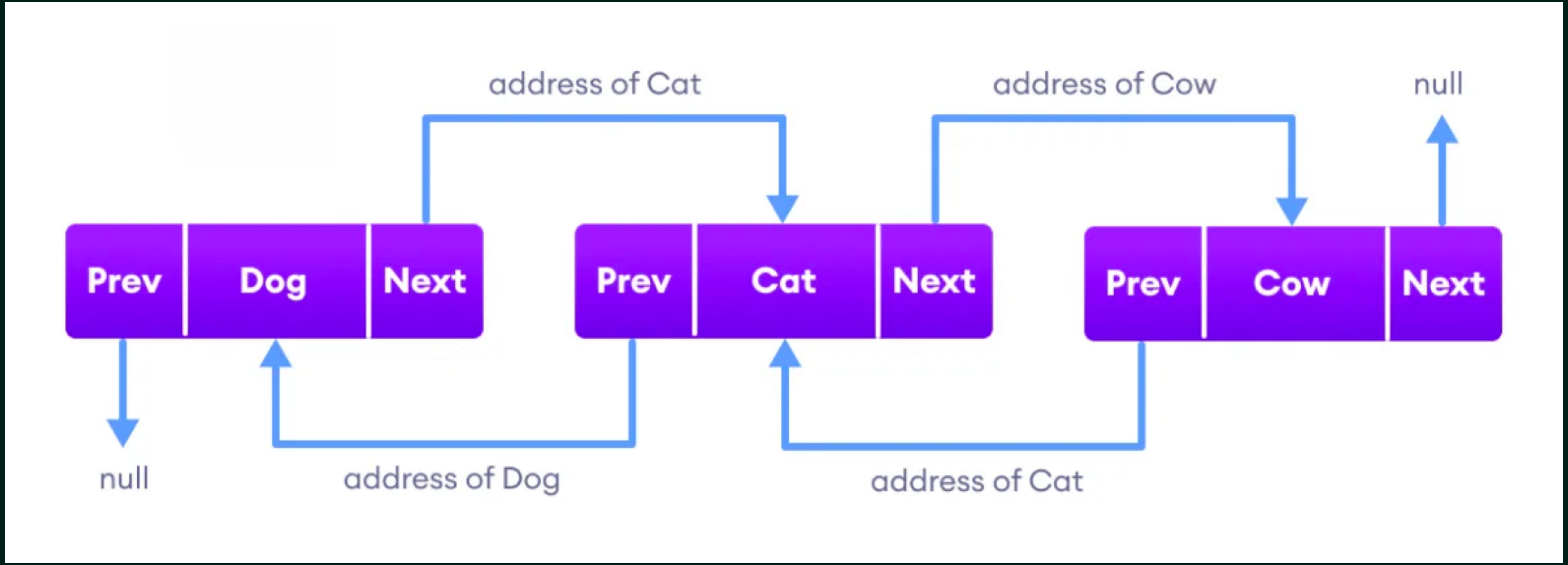
$10 \rightarrow 16 \rightarrow 25 \longrightarrow$ _ _

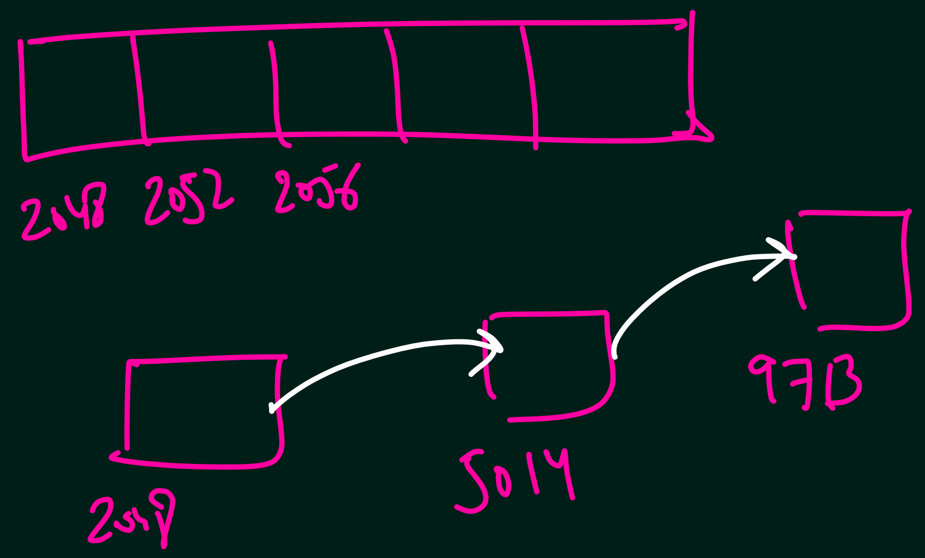
CODING SHUTTLE

Java LinkedList

The LinkedList class of the Java collections framework provides the functionality of the linked list data structure (doubly linkedlist).

Elements in linked lists are not stored in sequence. Instead, they are scattered and connected through links (Prev and Next).





Java Vector

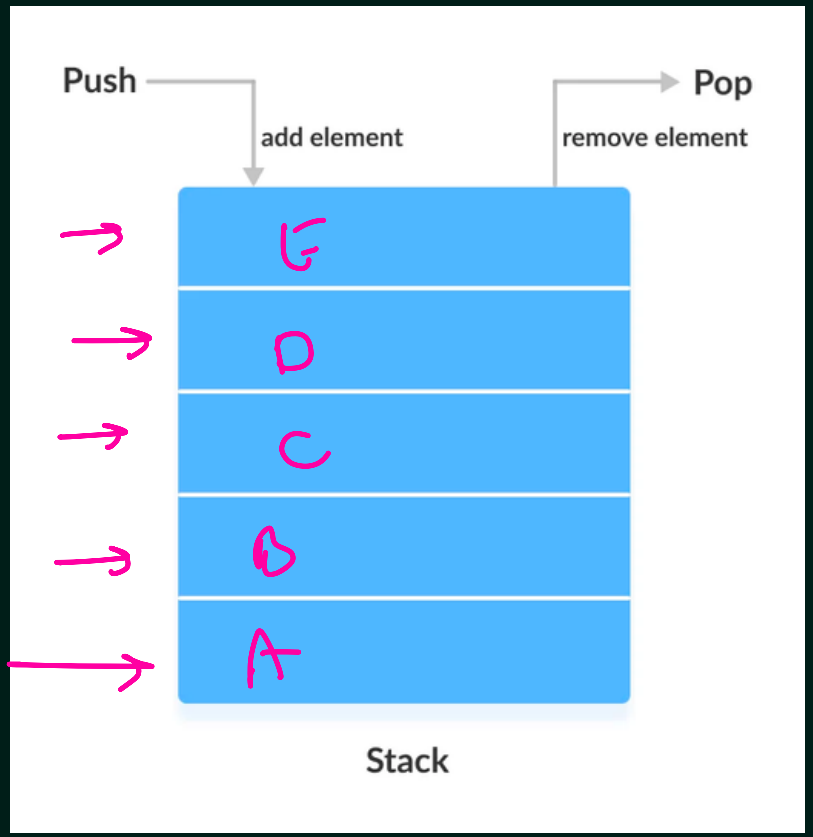
The Vector class synchronizes each individual operation. This means whenever we want to perform some operation on vectors, the Vector class automatically applies a lock to that operation.

It is because when one thread is accessing a vector, and at the same time another thread tries to access it, an exception called ConcurrentModificationException is generated. Hence, this continuous use of lock for each operation makes vectors less efficient.
However, in array lists, methods are not synchronized.

Java Stack

In stack, elements are stored and accessed in Last In First Out manner. That is, elements are added to the top of the stack and removed from the top of the stack

1. **void push(E item):** Pushes the given element onto the top of the stack.
2. **E pop():** Removes and returns the element at the top of the stack. Throws an EmptyStackException if the stack is empty.
3. **E peek():** Returns the element at the top of the stack without removing it. Throws an EmptyStackException if the stack is empty.
4. **boolean empty():** Returns true if the stack is empty, false otherwise.


Push
add element
Pop
remove element
E
D
C
B
A
Stack