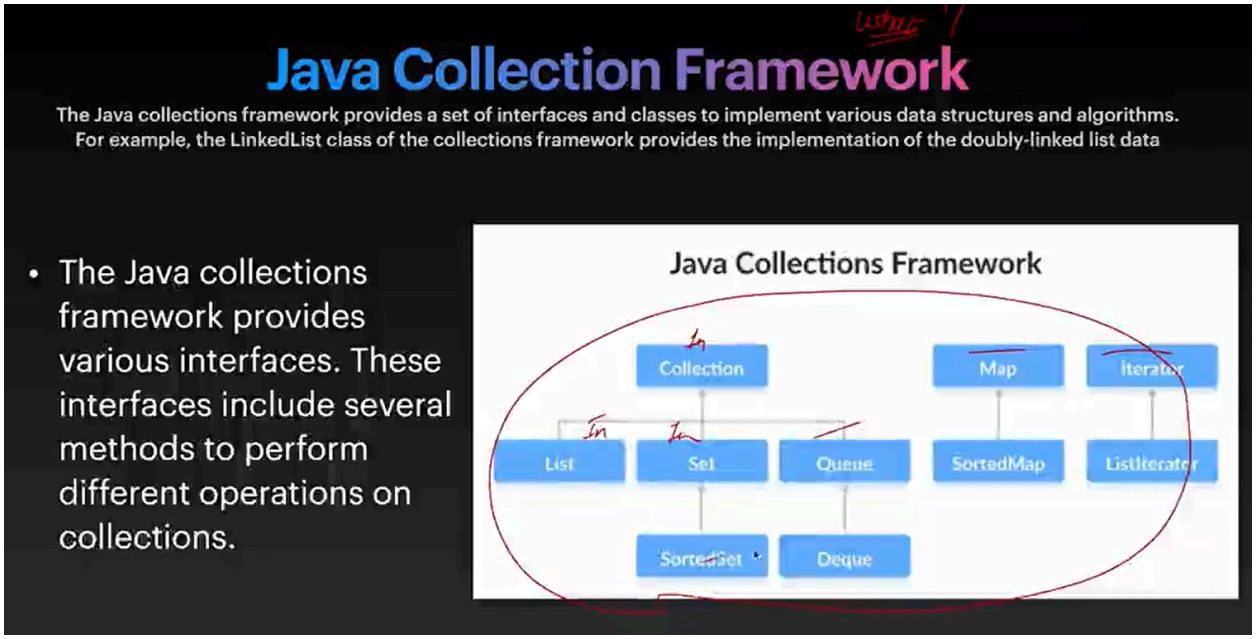
Java Collection Framework-1

INSTRUCTOR: Love Babbar

Module 1

23 November 2024

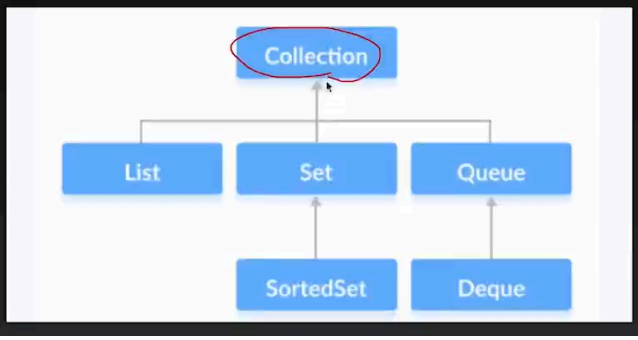


Used to utilize implementation of data structures like LL, Tree, HashMap, ArrayList

Java Collection Interface

- The Collection interface is the root interface of the collections framework hierarchy.

- Java does not provide direct implementations of the Collection interface but provides implementations of its subinterfaces like List, Set, and Queue. To learn more, visit: Java Collection Interface



Methods of Collection

The Collection interface includes various methods that can be used to perform different operations on objects. These methods are available in all its subinterfaces.

- `add()` - inserts the specified element to the collection
- `size()` - returns the size of the collection
- `remove()` - removes the specified element from the collection
- `iterator()` - returns an iterator to access elements of the collection
- `addAll()` - adds all the elements of a specified collection to the collection
- `removeAll()` - removes all the elements of the specified collection from the collection
- `clear()` - removes all the elements of the collection

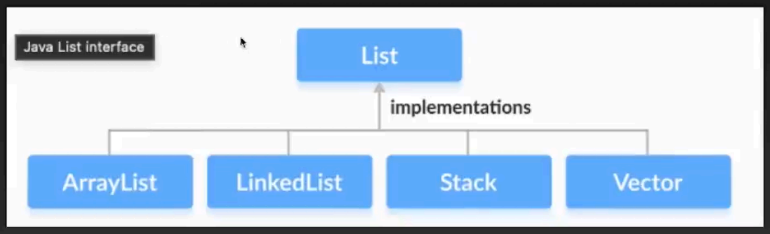
Java List Interface

In Java, the List interface is an ordered collection that allows us to store and access elements sequentially.

- How to use it ?

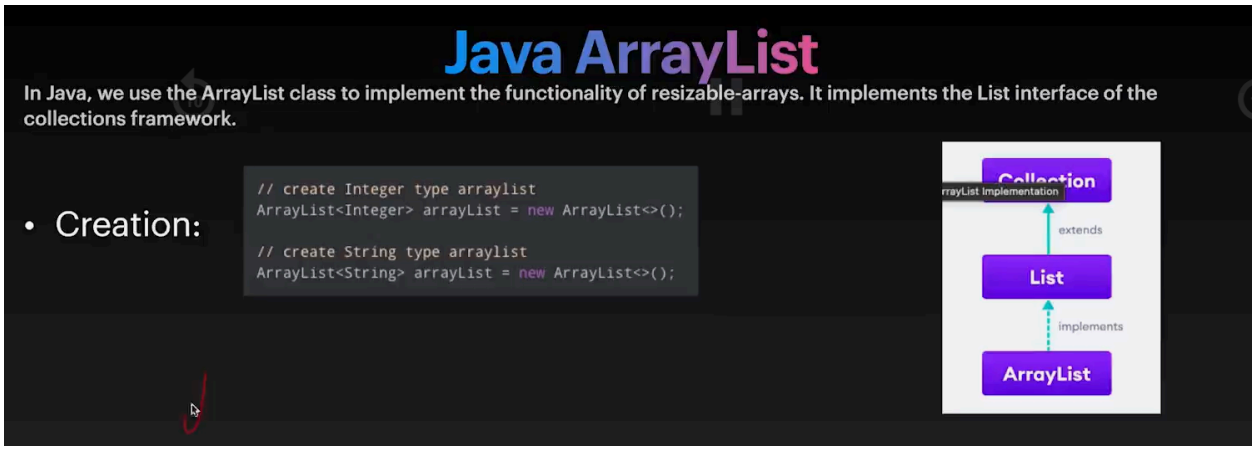
```
// ArrayList implementation of List
List<String> list1 = new ArrayList<>();

// LinkedList implementation of List
List<String> list2 = new LinkedList<>();
```

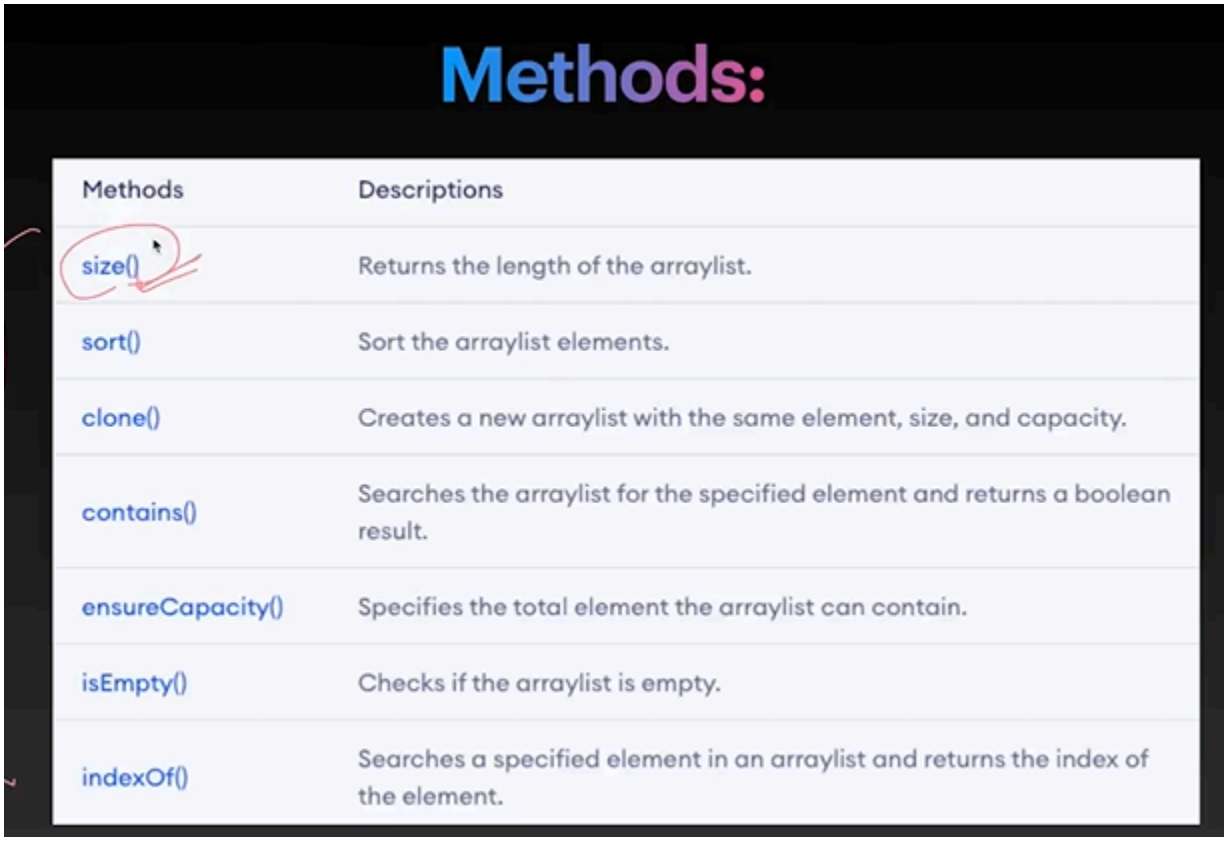


Methods:

Methods	Description
add()	adds an element to a list
addAll()	adds all elements of one list to another
get()	helps to randomly access elements from lists
iterator()	returns iterator object that can be used to sequentially access elements of lists
set()	changes elements of lists
remove()	removes an element from the list
removeAll()	removes all the elements from the list
clear()	removes all the elements from the list (more efficient than `removeAll()`)
size()	returns the length of lists
toArray()	converts a list into an array
contains()	returns `true` if a list contains specific element



ArrayList in java == vector in cpp



Iterator

```
//iterator() --> pending
Iterator<Integer> iterator = list.iterator();
while (iterator.hasNext()){
    System.out.print(" " +iterator.next());
}
```

hasNext() → returns true if has next element else false

And goes to next element if next element is present

next() → returns the next element

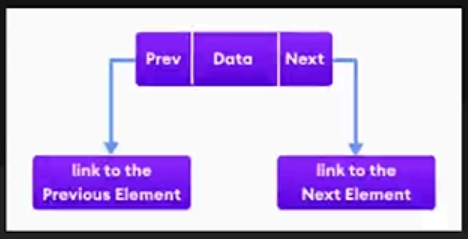
ensureCapacity() →

1. lets say suppose we have set the threshold for array is 75%
2. So if array is full upto 75% then it will automatically create the new array of size 1.5x.
3. And copy the values of old array to new array.
4. Changes the reference to new array.

Java LinkedList

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

- Each element in a linked list is known as a node. It consists of 3 fields:
- Prev - stores an address of the previous element in the list. It is null for the first element
- Next - stores an address of the next element in the list. It is null for the last element
- Data - stores the actual data



Methods:

Methods	Description
`contains()`	checks if the LinkedList contains the element
`indexOf()`	returns the index of the first occurrence of the element
`lastIndexOf()`	returns the index of the last occurrence of the element
`clear()`	removes all the elements of the LinkedList
`iterator()`	returns an iterator to iterate over LinkedList

Linked List as Queue & Duque

Since the LinkedList class also implements the Queue and the Deque interface, it can implement methods of these interfaces as well. Here are some of the commonly used methods:

Methods	Descriptions
`addFirst()`	adds the specified element at the beginning of the linked list
`addLast()`	adds the specified element at the end of the linked list
`getFirst()`	returns the first element
`getLast()`	returns the last element
`removeFirst()`	removes the first element
`removeLast()`	removes the last element
`peek()`	returns the first element (head) of the linked list
`poll()`	returns and removes the first element from the linked list
`offer()`	adds the specified element at the end of the linked list

Java Vector

The Vector class is an implementation of the List interface that allows us to create resizable-arrays similar to the ArrayList class.

- Vector vs ArrayList
  - In Java, both ArrayList and Vector implements the List interface and provides the same functionalities. However, there exist some differences between them.
  - 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. Instead, it uses the Collections.synchronizedList() method that synchronizes the list as a whole.
  - *It is recommended to use ArrayList in place of Vector because vectors less efficient.*

Methods:

- *Adding Elements:* add(else), add(index,ele), addAll(vector)
- *Access Elements*: get(index), iterator()
- *Removing Elements*: remove(index), removeAll(), clear()

Methods	Descriptions
`set()`	changes an element of the vector
`size()`	returns the size of the vector
`toArray()`	converts the vector into an array
`toString()`	converts the vector into a String
`contains()`	searches the vector for specified element and returns a boolean result

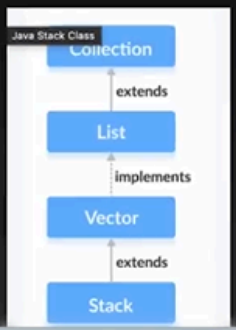
Java Stack

The Java collections framework has a class named Stack that provides the functionality of the stack data structure. The Stack class extends the Vector class.

- Creation:

```
// Create Integer type stack
Stack<Integer> stacks = new Stack<>();

// Create String type stack
Stack<String> stacks = new Stack<>();
```



Methods:

- push()
- pop()
- peek()
- search()
- empty()