- Collections Framework:
  - Unified architecture for storing and manipulating groups of objects
  - Core interfaces: Collection, List, Set, Queue, Map
  - List Interface:
  - It is a part of java.util package
  - Extends the collection interface
  - Represents an ordered collection(sequence),allowing duplicate elements
  - Index-based operations are supported(acces,insert,delete,update)

The List Interface:

Common methods:

add(E e), get(int index), set(int index), remove(int index), size(), isEmpty()

- Implementations of List Interface:

1. ArrayList
   Package: java.util.ArrayList

   Features:
   - Dynamic resizing array-based implementation.
   - Faster for random access (get operations)
   - Slower for insertions and deletions in the middle (Shifting is required).

When to use:

Frequent read operations and occasional inserts/removals

2.LinkedList

Package: java.util.LinkedList

Features:

- Doubly-linked list implementation
- Better performance for insertions and deletions in the middle (no shifting is required)
- Slower for random access.

When to use:

Frequent insertions/removals and less frequent access by index

3. Vector:

Package: java.util.vector

Features:

- Synchronized, making it thread safe.
- Slower than ArrayList due to synchronization overhead.
- Can grow dynamically like ArrayList

```java
package Vector;

import java.util.Vector;

public class VectorExample {
    public static void main(String[] args) {
        Vector<Integer> numbers = new Vector<>();

        //Adding elements
        numbers.add(10);
        numbers.add(20);
        numbers.add(30);
        numbers.add(40);
        numbers.add(50);
        numbers.add(60);
        numbers.add(70);
        numbers.add(80);
        numbers.add(90);
        numbers.add(100);
        numbers.add(110);


        //Accessing the elements by index
        System.out.println("First number is  " +
numbers.get(0));

        //Iterate through the vector
        for(Integer number : numbers){
            System.out.println(number);
        }


        System.out.println("Size : "+ numbers.size());
        System.out.println("Capacity
"+numbers.capacity());
    }
}
```

When to use:

If Thread-Safety is required

4. Stack:

Package: java.util.Stack

Features:

- Subclass of Vector
- Implements LIFO (Last in, First Out) structure
- Includes methods like pop(), push(), peek(), empty()

When to use:

When stack behavior is needed.

The List Interface:

Feature	ArrayList	LinkedList	Vector	Stack
Underlying Structure	Dynamic Array	Doubly Linked List	Dynamic Array	Dynamic Array
Thread Safety	No	No	Yes	Yes
Insertion Performance	Slower{O(n)}	Faster {O(1) at ends}	Slower{O(n)}	Slower{O(n)}
Access Performance	Faster {O(1)}	Slower{O(n)}	Faster {O(1)}	Faster {O(1)}
When to use	Frequent Access	Frequent insert/remove	Thread-Safety needed	LIFO required

Stack: Track Book Reading by stack

```
package Book;

import java.util.Stack;

public class BookTracker {
    public static void main(String[] args) {
        Stack<String> books = new Stack<>();

        // Adding books
        books.push("The Language of SQL ");
        books.push("Core java & advance java");
        books.push("Data Structures & Algorithms");

        //Current Book
        System.out.println("Currently Reading : "+
books.peek());

        //Finished reading one book
        System.out.println("Finished Reading : " +
books.pop());

        //Displaying remaining books
        System.out.println("Books in Stack : "+ books);
    }
}
```

R&D:

Thread-safety

Synchronizaton

Stack (Java SE 21 & JDK 21)