```java
package exampleUserInput;

import java.util.ArrayList;
import java.util.Scanner;

public class SimpleArrayListExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<String> names = new ArrayList<>();

        System.out.println("Enter 3 Names: ");
        for (int i=0;i<3;i++){
            System.out.println("Enter name "+(i+1)+" : ");
            String name= scanner.nextLine();
            names.add(name);
        }

        scanner.close();

        System.out.println("You have Entered: "+ names);
    }
}
```

Quick Revision:

Collection Framework:

Architecture for storing and manipulating group of objects

Collection Interface:

An Interface which provides a common structure for all collection classes.

- List Interface: Represents an ordered collection and also allows duplicates.
- Set Interface:
- Queue Interface:
- Map Interface:

List Interface:

Package: java.util.list

Extends: Collection Interface

Characteristics:

- Stores ordered elements(sequential)
- Allows duplicate elements

Common methods of list interface:

add(), get(), set(), remove(), size(),isEmpty()

Implementations of list interface:

1. ArrayList

   Package: java.util.arrayList

   Features :

- Uses dynamic array internally
- Fast for accessing elements
- Slow for insertions and deletions in the middle

When to use:

When frequent read operations are required

If insertions and deletions are less.

2. LinkedList:

   Package: java.util.linkedList

   Features:

- Implements doubly linked list internally
- Efficient for insertions and deletions
- Slower access by index

When to use:

When frequent insertions and deletions are required.

3. Vector

Package: java.util.vector

Features:

- Uses dynamic array like arraylist
- Its thread safe because its all methods are synchronized
- Slower than the arraylist because of synchronization

When to use:

- When multiple threads needs to access the list concurrently(BookMyShow)

4. Stack

Package: java.util.stack

Features:

Follows LIFO(Last In First Out) principle
A subclass of a vector

Methods:

Push() : adds an element to the top
Pop(): Removes and returns the top element
Peek(): Returns the top element without removing it.
Empty(): Checks the stack is empty or not.

When to use:

When we need LIFO Behaviour(undo operations,browser history)

Stack (Java SE 21 & JDK 21)

```
package stackE;

import java.util.Stack;

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

        //Adding few books in stack
        books.push("Black Book JAVA");
        books.push("The Basics of SQL");
        books.push("C# Basics");

        //To get current book(Top of Stack)
        System.out.println("Currently Reading:
"+books.peek());

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

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

Capacity of arraylist:

Upto to JDK 6 the capacity grow with the formula

NewCapacity = (oldCapacity * 3/2)+1;

NewCapacity = (10 * 3/2) + 1

NewCapacity = 16

In JDK 7 and above formula changes to

NewCapacity = oldCapacity + (oldCapacity >> 1)

NewCapacity = 10 + (10 >> 1)

= 10 +5

= 15

1010 -> 10

0101 -> 5

ArrayList Vs Vector:

```
package exampleSpeed;

import java.util.ArrayList;
import java.util.Vector;

public class AVExample {
    public static void main(String[] args) {
        Vector<Integer> vector = new Vector<>();
        ArrayList<Integer> arrayList = new ArrayList<>();

        long startTime, endTime;

        // Measuring vector performance
        startTime = System.nanoTime();
        for (int i =0; i < 100000; i++){
            vector.add(i);
        }
        endTime = System.nanoTime();
        System.out.println("Time taken by Vector : "+
(endTime-startTime) + "ns");


System.out.println("==========================================
===========");

//         Measuring ArrayList performance
        startTime = System.nanoTime();
        for (int i=0; i < 100000; i++){
            arrayList.add(i);
        }
        endTime=System.nanoTime();
        System.out.println("Time taken by arrayList: "+
(endTime-startTime)+ "ns");
    }
}
```

O/P:

Time taken by Vector : 19656300ns

===

Time taken by arrayList: 9932100ns

System (Java SE 21 & JDK 21)-> nanoTime();

	ArrayList	**LinkedList**	**Vector**	**Stack**
Structure	Dynamic Array	Doubly LinkedList	Dynamic Array	Dynamic Array
Thread Safety	No	No	Yes	Yes
When to use	Frequent access	Frequent insert/ remove	Thread-safety is required	LIFO required

SET Interface:

- Stores unique elements and do not allow duplicates.
- Two implementations:
  1. HashSet
  2. LinkedHashSet