Assume you are expert in Java and Problem solver then What is a List in Java?

Absolutely! Let's break it down in a simple and clear way.

🔹 What is a List in Java?

In **Java**, a `List` is an **interface** that belongs to the `java.util` package and is part of the **Java Collections Framework**. It represents an **ordered collection** (also known as a **sequence**) of elements where:

- **Duplicates are allowed**
- **Insertion order is preserved**
- Elements can be accessed using **index (starting from 0)**

🔹 Key Features of a List:

- You can **add**, **remove**, and **search** for elements.
- Allows **null values** (depends on the implementation).
- Supports **iteration** using loops or iterators.
- Can be **resized dynamically** (unlike arrays which have fixed size).

🔹 List Interface Syntax:

java
CopyEdit
```
List<Type> listName = new ArrayList<>();
```

Here, `Type` is the data type of elements (like `Integer`, `String`, `Double`, etc.), and `ArrayList` is one of the classes that implements the `List` interface.

🔹 Common Implementations of List:

Class	Description
`ArrayList`	Fast in access, slower in insertion/deletion
`LinkedList`	Fast in insertion/deletion, slower in access
`Vector`	Synchronized version of ArrayList (legacy class)
`Stack`	Subclass of Vector that follows LIFO (legacy)

🔹 Example:

java
CopyEdit
```
import java.util.*;

public class ListExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>();

        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");

        System.out.println("Fruits: " + fruits);
        System.out.println("First fruit: " + fruits.get(0));
    }
}
```

Output:

yaml

CopyEdit

```
Fruits: [Apple, Banana, Mango]
First fruit: Apple
```

🔹 Useful List Methods:

Method	Description
`add(element)`	Adds element at end
`add(index, element)`	Inserts element at specific index
`get(index)`	Gets element at index
`set(index, element)`	Replaces element at index
`remove(index)`	Removes element at index
`size()`	Returns the number of elements
`contains(element)`	Checks if element exists
`isEmpty()`	Checks if list is empty
`clear()`	Removes all elements

Assume you are expert in Java and Problem solver then Difference between List, Set, and Map

Absolutely! Here's a **clear and concise comparison between `List`, `Set`, and `Map` in Java** from an expert's perspective:

🔸 1. `List`

- **Definition**: An **ordered** collection that **allows duplicates**.
- **Interface**: `java.util.List`
- **Common Implementations**:
  - `ArrayList`
  - `LinkedList`
  - `Vector`

✅ **Key Points:**

- Maintains **insertion order**.
- Elements are accessible by **index**.
- Allows **duplicate values**.

🧪 **Example:**
java
CopyEdit
```
List<String> list = new ArrayList<>();
list.add("Apple");
list.add("Banana");
list.add("Apple");  // Allowed (duplicates)
System.out.println(list); // [Apple, Banana, Apple]
```

🔸 2. `Set`

- **Definition**: An **unordered** collection that **does NOT allow duplicates**.
- **Interface**: `java.util.Set`
- **Common Implementations**:

- `HashSet` (no order)
- `LinkedHashSet` (insertion order)
- `TreeSet` (sorted order)

✅ **Key Points:**

- **No duplicate elements**.
- Doesn't support indexing.
- Best for **unique values only**.

🧪 **Example:**
java
CopyEdit
```
Set<String> set = new HashSet<>();
set.add("Apple");
set.add("Banana");
set.add("Apple");  // Ignored (duplicate)
System.out.println(set); // [Apple, Banana] or unordered
```

🔸 3. `Map`

- **Definition**: A collection of **key-value pairs**. Keys are **unique**, but values can be duplicated.
- **Interface**: `java.util.Map`
- **Common Implementations**:
  - `HashMap`
  - `LinkedHashMap`
  - `TreeMap`

✅ **Key Points:**

- Stores **key-value** pairs.
- **Keys must be unique**, but **values can repeat**.
- No concept of index; accessed by **key**.

🧪 **Example:**
java
CopyEdit
```
Map<Integer, String> map = new HashMap<>();
map.put(1, "Apple");
map.put(2, "Banana");
map.put(1, "Mango");  // Key 1 is updated
System.out.println(map); // {1=Mango, 2=Banana}
```

🔍 Summary Table

Feature	List	Set	Map
Allows Duplicates	✅ Yes	❌ No	✅ Values only
Maintains Order	✅ Yes (insertion)	❌ Not always	✅ LinkedHashMap, TreeMap
Index Access	✅ Yes	❌ No	❌ No (key-based access)
Key-Value Pair	❌ No	❌ No	✅ Yes
Null Handling	✅ Multiple nulls	✅ One null	✅ One null key, many null values

Assume you are expert in Java and Problem solver then Provide notes on List is an interface, implemented by: ○ ArrayList ○ LinkedList ○ Vector ○ Stack

Absolutely! Here's a clear and concise note on the `List` interface in Java, and its key implementations: `ArrayList`, `LinkedList`, `Vector`, and `Stack`.

✅ Java `List` Interface and Its Implementations

🔷 What is `List`?

- `List` is a **child interface of Collection** in `java.util` package.
- It represents an **ordered collection (sequence)** that allows **duplicate elements**.
- You can **access elements by index**, insert or remove elements at specific positions.

java
CopyEdit
```
List<String> list = new ArrayList<>();
```

✅ Key Implementations of `List`:

---

1. ArrayList

📌 Description:

- Resizable-array implementation of `List`.
- **Faster access** by index.
- Good for **read-heavy operations**.

🔧 Characteristics:

- **Ordered**: Maintains insertion order.
- **Allows duplicates and nulls**.
- **Not synchronized** (not thread-safe).
- Internally uses a **dynamic array**.

🧪 Example:

java
CopyEdit
```
List<String> arrayList = new ArrayList<>();
arrayList.add("Java");
arrayList.add("Python");
System.out.println(arrayList.get(1)); // Output: Python
```

---

2. LinkedList

📌 Description:

- Doubly-linked list implementation of `List` and `Deque`.
- Good for **frequent insertions and deletions**.

🔧 Characteristics:

- Maintains **insertion order**.

- **Slower access** by index compared to ArrayList.
- **Allows duplicates and nulls**.
- **Not synchronized**.

🧪 Example:

java
CopyEdit
```
List<String> linkedList = new LinkedList<>();
linkedList.add("Node.js");
linkedList.addFirst("C++"); // Only in LinkedList
System.out.println(linkedList); // Output: [C++, Node.js]
```

---

3. Vector

📌 Description:

- Legacy class, similar to ArrayList but **synchronized**.
- Thread-safe but **slower** due to synchronization overhead.

🔧 Characteristics:

- Maintains **insertion order**.
- **Allows duplicates and nulls**.
- **Synchronized**.

🧪 Example:

java
CopyEdit
```
List<Integer> vector = new Vector<>();
vector.add(10);
vector.add(20);
```

```
System.out.println(vector.size()); // Output: 2
```

---

4. Stack

📌 Description:

- Subclass of Vector.
- **LIFO (Last-In-First-Out)** stack implementation.

🔧 Characteristics:

- Inherits Vector's features (synchronized, ordered).
- Provides **push(), pop(), peek()** methods.

🧪 Example:

java
CopyEdit

```
Stack<String> stack = new Stack<>();
stack.push("HTML");
stack.push("CSS");
System.out.println(stack.pop()); // Output: CSS
```

🔁 Summary Table

Feature	ArrayList	LinkedList	Vector	Stack
Order	Yes	Yes	Yes	Yes
Duplicates	Allowed	Allowed	Allowed	Allowed
Nulls	Allowed	Allowed	Allowed	Allowed
Thread-Safe	❌ No	❌ No	✅ Yes	✅ Yes

Performance	Fast for read	Fast for insert/delete	Slower due to sync	Slower + LIFO
Special Use	General list	Queue/Deque	Legacy use	LIFO stack