Java List Implementations

In Java, `List` is an interface in the **Java Collections Framework** that represents an **ordered collection (sequence)** of elements. The most commonly used implementations of `List` are:

- `ArrayList`
- `LinkedList`

Both store elements in an ordered way and allow duplicate entries, but they differ in how they store and access data internally.

✅ `List` Interface Overview

```
java

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

Key features:

- Maintains insertion order
- Allows duplicates
- Can be accessed via index (`list.get(0)`)

🔹 `ArrayList`

✅ Characteristics:

- Backed by a **dynamic array**
- **Fast random access** using index (O(1))
- **Slow insertions/deletions** in the middle (O(n)) – requires shifting elements

📌 Example:

```
java

import java.util.*;

public class ArrayListExample {
    public static void main(String[] args) {
        List<String> arrayList = new ArrayList<>();
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Mango");

        System.out.println(arrayList.get(1)); // Banana
        arrayList.remove("Apple");
        System.out.println(arrayList); // [Banana, Mango]
```

```
    }
}
```

🔹 `LinkedList`

✅ Characteristics:

- Implemented as a **doubly-linked list**
- **Fast insertions/deletions** (especially at start/middle)
- **Slower access** to elements by index (O(n))

📌 Example:

```
java

import java.util.*;

public class LinkedListExample {
    public static void main(String[] args) {
        List<String> linkedList = new LinkedList<>();
        linkedList.add("Red");
        linkedList.add("Green");
        linkedList.add("Blue");

        linkedList.addFirst("Yellow"); // LinkedList specific method
        System.out.println(linkedList); // [Yellow, Red, Green, Blue]

        linkedList.remove(2); // removes "Green"
        System.out.println(linkedList); // [Yellow, Red, Blue]
    }
}
```

🔄 ArrayList vs LinkedList Comparison

Feature	ArrayList	LinkedList
Internal Structure	Dynamic Array	Doubly Linked List
Access by Index	Fast (O(1))	Slow (O(n))
Insertion/Removal	Slow (O(n))	Fast (O(1) or O(n))
Memory	Less Overhead	More Overhead
Use Case	Frequent access	Frequent insertions/deletions

📝 When to Use What?

- Use `ArrayList` when you need **fast access** and **rare insertions/deletions**.
- Use `LinkedList` when you do **frequent insertions/deletions**, especially at the **beginning or middle** of the list.

Would you like an example of performance difference or when to use them in test automation (like Selenium test data list)?

Great! Let's break this into two parts:

✅ 1. Performance Difference Example

Let's see how **`ArrayList`** and **`LinkedList`** behave when inserting elements **in the middle of the list**.

🔧 Java Code to Compare Insertion Performance

```
java

import java.util.*;

public class ListPerformanceTest {
    public static void main(String[] args) {
        List<Integer> arrayList = new ArrayList<>();
        List<Integer> linkedList = new LinkedList<>();

        // Add 100000 elements
        for (int i = 0; i < 100000; i++) {
            arrayList.add(i);
            linkedList.add(i);
        }

        // Insert at the middle
        long start1 = System.nanoTime();
        arrayList.add(50000, 99999);
        long end1 = System.nanoTime();
        System.out.println("ArrayList insertion time: " + (end1 - start1) + " ns");

        long start2 = System.nanoTime();
        linkedList.add(50000, 99999);
        long end2 = System.nanoTime();
        System.out.println("LinkedList insertion time: " + (end2 - start2) + " ns");
    }
}
```

📌 Output Example (will vary based on machine):

```
less

ArrayList insertion time: 450000 ns
LinkedList insertion time: 70000 ns
```

📊 **Result**: **`LinkedList`** is faster for insertion in the middle.

✅ 2. Selenium Example Using List

You often use **`List<WebElement>`** when working with Selenium. Here's how:

🔍 Example: Use of List in Selenium Test Script

```
java

import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.List;

public class SeleniumListExample {
```

```
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        // Find all links on the page
        List<WebElement> links = driver.findElements(By.tagName("a"));

        System.out.println("Total links: " + links.size());

        for (WebElement link : links) {
            System.out.println(link.getText() + " --> " + link.getAttribute("href"));
        }

        driver.quit();
    }
}
```

📌 Notes:

- **`List<WebElement>`** stores all elements matching the criteria.
- Iterating over the list allows validation or actions on each item.
- Internally, Selenium uses an **`ArrayList`** to store WebElements.

🧠 Summary

Use Case	Recommended List Type
Test data access	**`ArrayList`**
Dynamic form fields, add/remove	**`LinkedList`**
Selenium WebElements collection	**`ArrayList`**

Would you like a hands-on project using **`ArrayList`** or **`LinkedList`** with Selenium, like checking all product names on a webpage or validating dropdown items?

Perfect! Here's a **hands-on mini-project** using `ArrayList` in a Selenium test case to:

✅ Use Case:

Validate dropdown items on a webpage against expected values.

🧪 🔽 Selenium Dropdown Validation Using `ArrayList`

🧰 Prerequisites

- Java
- Selenium WebDriver
- ChromeDriver
- TestNG or JUnit (optional)

📄 HTML (Sample Page You Can Use Locally)

Save this as **`dropdown.html`**:

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Dropdown Test</title>
</head>
<body>
    <h2>Select Your Country</h2>
    <select id="country">
        <option value="india">India</option>
        <option value="usa">USA</option>
        <option value="uk">UK</option>
        <option value="canada">Canada</option>
    </select>
</body>
</html>
```

🧪 Selenium Java Test with `ArrayList`

```
java

import org.openqa.selenium.*;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.Select;
import java.util.*;

public class DropdownValidationTest {
    public static void main(String[] args) {
        // Setup WebDriver
        WebDriver driver = new ChromeDriver();
        driver.get("file:///path/to/your/dropdown.html"); // Replace with local file path

        // Step 1: Expected dropdown values
        List<String> expectedCountries = new ArrayList<>();
        expectedCountries.add("India");
        expectedCountries.add("USA");
```

```
        expectedCountries.add("UK");
        expectedCountries.add("Canada");

        // Step 2: Get dropdown element
        WebElement countryDropdown = driver.findElement(By.id("country"));
        Select select = new Select(countryDropdown);

        // Step 3: Fetch all actual options
        List<WebElement> options = select.getOptions();
        List<String> actualCountries = new ArrayList<>();

        for (WebElement option : options) {
            actualCountries.add(option.getText());
        }

        // Step 4: Compare both lists
        if (expectedCountries.equals(actualCountries)) {
            System.out.println("✅ Dropdown values are as expected.");
        } else {
            System.out.println("❌ Mismatch found!");
            System.out.println("Expected: " + expectedCountries);
            System.out.println("Actual  : " + actualCountries);
        }

        driver.quit();
    }
}
```

✅ Output:

```
sql

✅ Dropdown values are as expected.
```

Or if there's a mismatch:

```
less

❌ Mismatch found!
Expected: [India, USA, UK, Canada]
Actual  : [India, UK, USA, Canada]
```

🧠 What You Learn:

- How to use **`ArrayList`** to manage expected data.
- How to extract values from a dropdown using Selenium.
- How to validate actual vs expected using **`.equals()`** method.

Would you like a **LinkedList use case** next, or do you want to extend this project to **multiple dropdowns / table validations / test data from Excel**?