Java Set Implementations

In Java, the `Set` interface represents a collection that **does not allow duplicate elements**. Java provides three commonly used implementations of the `Set` interface:

✅ 1. HashSet

- **Unordered**
- **Fastest** among all Set implementations
- **Backed by a HashMap**
- **Allows `null`** (only one null element)

🔹 Characteristics:

- No insertion order preserved.
- Constant time performance for basic operations (**`add`**, **`remove`**, **`contains`**).
- Not synchronized (not thread-safe).

✅ Example:

```
java

import java.util.HashSet;

public class HashSetExample {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<>();
        set.add("Apple");
        set.add("Banana");
        set.add("Orange");
        set.add("Apple"); // Duplicate, will not be added

        System.out.println(set); // Order is not guaranteed
    }
}
```

✅ 2. LinkedHashSet

- **Ordered version of HashSet**
- Maintains **insertion order**
- **Slightly slower** than HashSet

🔹 Characteristics:

- Maintains order in which elements were inserted.
- Still allows **only one `null`**.

- Ideal when **you want uniqueness + order**.

✅ Example:

```
java

import java.util.LinkedHashSet;

public class LinkedHashSetExample {
    public static void main(String[] args) {
        LinkedHashSet<String> set = new LinkedHashSet<>();
        set.add("Banana");
        set.add("Apple");
        set.add("Orange");
        set.add("Banana"); // Duplicate

        System.out.println(set); // [Banana, Apple, Orange]
    }
}
```

✅ 3. TreeSet

- **Sorted set**
- Elements are **sorted in natural order** (e.g., numbers in ascending, strings alphabetically)
- **Does not allow `null`**

🔹 Characteristics:

- Implements **`NavigableSet`** interface.
- Based on **Red-Black Tree (self-balancing BST)**.
- Slower than **`HashSet`** and **`LinkedHashSet`**.

✅ Example:

```
java

import java.util.TreeSet;

public class TreeSetExample {
    public static void main(String[] args) {
        TreeSet<String> set = new TreeSet<>();
        set.add("Banana");
        set.add("Apple");
        set.add("Orange");
        // set.add(null); // Throws NullPointerException

        System.out.println(set); // [Apple, Banana, Orange]
    }
}
```

🔁 Quick Comparison Table:

Feature	HashSet	LinkedHashSet	TreeSet
Order	No	Insertion Order	Sorted (natural/comparator)

Feature	HashSet	LinkedHashSet	TreeSet
Performance	Fastest	Slightly Slower	Slowest
Null Allowed	Yes (one)	Yes (one)	❌ No
Thread-Safe	❌ No	❌ No	❌ No
Use Case	Fast lookup	Order + uniqueness	Sorted + uniqueness

Would you like to practice each one with small code examples or quiz questions?

Here's a **hands-on implementation of test cases** for `HashSet`, `LinkedHashSet`, and `TreeSet` using **Java + TestNG**.

✅ Project Structure:

Assume you have the following Java Maven/TestNG setup:

```
bash

src/
 └── test/
      └── java/
           └── set_examples/
                ├── HashSetTest.java
                ├── LinkedHashSetTest.java
                └── TreeSetTest.java
```

✅ 1. `HashSet` Test Case

```
java

package set_examples;

import org.testng.Assert;
import org.testng.annotations.Test;

import java.util.HashSet;
import java.util.Set;

public class HashSetTest {

    @Test
    public void testHashSetUniqueness() {
        Set<String> fruits = new HashSet<>();
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Apple"); // Duplicate

        Assert.assertEquals(fruits.size(), 2);
        Assert.assertTrue(fruits.contains("Banana"));
        Assert.assertFalse(fruits.contains("Mango"));
    }

    @Test
    public void testHashSetAllowsNull() {
        Set<String> items = new HashSet<>();
        items.add(null);
        items.add(null); // Only one null allowed

        Assert.assertEquals(items.size(), 1);
        Assert.assertTrue(items.contains(null));
    }
}
```

✅ 2. `LinkedHashSet` Test Case

```java
package set_examples;

import org.testng.Assert;
import org.testng.annotations.Test;

import java.util.Iterator;
import java.util.LinkedHashSet;

public class LinkedHashSetTest {

    @Test
    public void testLinkedHashSetOrder() {
        LinkedHashSet<String> items = new LinkedHashSet<>();
        items.add("Pen");
        items.add("Pencil");
        items.add("Eraser");

        Iterator<String> iterator = items.iterator();
        Assert.assertEquals(iterator.next(), "Pen");
        Assert.assertEquals(iterator.next(), "Pencil");
        Assert.assertEquals(iterator.next(), "Eraser");
    }

    @Test
    public void testLinkedHashSetNull() {
        LinkedHashSet<String> items = new LinkedHashSet<>();
        items.add(null);
        items.add(null);

        Assert.assertEquals(items.size(), 1);
    }
}
```

✅ 3. `TreeSet` Test Case

```java
package set_examples;

import org.testng.Assert;
import org.testng.annotations.Test;

import java.util.Iterator;
import java.util.TreeSet;

public class TreeSetTest {

    @Test
    public void testTreeSetSorting() {
        TreeSet<String> animals = new TreeSet<>();
        animals.add("Zebra");
        animals.add("Elephant");
        animals.add("Cat");

        Iterator<String> iterator = animals.iterator();
        Assert.assertEquals(iterator.next(), "Cat");
        Assert.assertEquals(iterator.next(), "Elephant");
        Assert.assertEquals(iterator.next(), "Zebra");
    }

    @Test(expectedExceptions = NullPointerException.class)
    public void testTreeSetDoesNotAllowNull() {
        TreeSet<String> items = new TreeSet<>();
        items.add(null); // Throws NullPointerException
```

```
    }
}
```

✅ How to Run (Maven/TestNG)

If you're using Maven, add TestNG to your **`pom.xml`**:

```xml
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.8.0</version>
  <scope>test</scope>
</dependency>
```

Then, use the IDE or command line:

```bash
mvn test
```

✅ Summary:

Test Class	Key Focus
`HashSetTest`	Uniqueness & null handling
`LinkedHashSetTest`	Insertion order + null
`TreeSetTest`	Natural sorting + no null allowed

Would you like to include negative test cases, or JUnit version too?