Java Iterator

In Java, both `Iterator` and `ListIterator` are used to **traverse** collections (like lists, sets), but they have **some key differences** in functionality and usage.

🔁 `Iterator` in Java

✅ Overview:

- Available in the `java.util` package.
- Can be used to **iterate over any Collection** (e.g., List, Set).
- **Forward traversal only**.
- Can **remove** elements while iterating.

✅ Methods:

Method	Description
`hasNext()`	Returns `true` if there are more elements.
`next()`	Returns the next element in the iteration.
`remove()`	Removes the last element returned by `next()`.

✅ Example of `Iterator`:

```java
import java.util.*;

public class IteratorExample {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>(Arrays.asList("Alice", "Bob", "Charlie"));

        Iterator<String> it = names.iterator();

        while (it.hasNext()) {
            String name = it.next();
            System.out.println(name);
            if (name.equals("Bob")) {
                it.remove(); // Safe removal
            }
        }

        System.out.println("After removal: " + names);
    }
}
```

🔁 `ListIterator` in Java

✅ Overview:

- Only available for **List** implementations (like **`ArrayList`**, **`LinkedList`**).
- **Bidirectional traversal** (forward and backward).
- Can **add, remove, or modify** elements while iterating.

✅ Methods:

Method	Description
`hasNext()`	Checks if there is a next element.
`next()`	Moves forward and returns next element.
`hasPrevious()`	Checks if there is a previous element.
`previous()`	Moves backward and returns previous element.
`add(E e)`	Adds element before next.
`remove()`	Removes last returned element (via next/previous).
`set(E e)`	Replaces last element returned.

✅ Example of `ListIterator`:

```java
import java.util.*;

public class ListIteratorExample {
    public static void main(String[] args) {
        List<String> fruits = new ArrayList<>(Arrays.asList("Apple", "Banana", "Cherry"));

        ListIterator<String> listIt = fruits.listIterator();

        // Forward traversal
        while (listIt.hasNext()) {
            String fruit = listIt.next();
            System.out.println("Forward: " + fruit);
            if (fruit.equals("Banana")) {
                listIt.set("Blueberry"); // Replaces "Banana" with "Blueberry"
            }
        }

        // Backward traversal
        while (listIt.hasPrevious()) {
            String fruit = listIt.previous();
            System.out.println("Backward: " + fruit);
        }

        System.out.println("Updated list: " + fruits);
    }
}
```

🔍 Summary Comparison:

Feature	`Iterator`	`ListIterator`
Applicable to	Any Collection	Only List
Direction	Forward only	Forward & Backward
Add/Set Elements	No	Yes

Feature	`Iterator`	`ListIterator`
Remove Elements	Yes	Yes
Traverse from index	No	Yes (`list.listIterator(index)`)

Would you like interview-style questions based on this topic too?

Great! Here are some **interview-style questions** on `Iterator` and `ListIterator` in Java, along with **sample answers**:

✅ Basic-Level Questions

1. ❓ What is the difference between Iterator and ListIterator in Java?

✅ **Answer:**

- `Iterator` can be used with any collection, but only supports **forward traversal**.
- `ListIterator` is used only with **List implementations** and supports **both forward and backward traversal**.
- `ListIterator` allows element **modification (set), addition (add), and removal**, whereas `Iterator` only allows **removal**.

---

2. ❓ Can we use Iterator to traverse a Map?

✅ **Answer:**

- Directly, no. But we can use an `Iterator` on the `entrySet()`, `keySet()`, or `values()` of a `Map`.

📌 Example:

```java
Iterator<Map.Entry<Integer, String>> it = map.entrySet().iterator();
```

---

3. ❓ What happens if you modify a list (add/remove) directly while iterating with Iterator?

✅ **Answer:**
It will throw a `ConcurrentModificationException` because the collection was structurally modified without using the iterator's own methods (like `remove()`).

✅ Intermediate-Level Questions

4. ❓ How can you iterate a list in reverse order using ListIterator?

✅ **Answer:**
Use `list.listIterator(list.size())` to start from the end, then use `hasPrevious()` and `previous()`.

📌 Example:

```java
ListIterator<String> it = list.listIterator(list.size());
while (it.hasPrevious()) {
    System.out.println(it.previous());
}
```

5. ❓ What is the difference between `remove()` in Iterator and `remove()` in ListIterator?

✅ **Answer:**

- Both remove the last element returned by `next()` (or `previous()` for `ListIterator`).
- In `ListIterator`, removal can be done in both forward and backward directions.

✅ Advanced-Level Questions

6. ❓ Explain internal working of Iterator. How does it detect concurrent modification?

✅ **Answer:**

- Most collections maintain a **modCount** field (modification count).
- The iterator captures modCount during creation.
- Before every operation (like `next()`), it checks if the current modCount equals the expectedModCount.
- If not, it throws `ConcurrentModificationException`.

7. ❓ Can we modify an element while iterating using Iterator or ListIterator?

✅ **Answer:**

- With `Iterator`, we **cannot modify** (only remove).
- With `ListIterator`, we can **modify using** `set()` and **add using** `add()` safely.

8. ❓ Which Iterator should you prefer if you want thread-safe iteration?

✅ **Answer:**

- Use `CopyOnWriteArrayList` or `Collections.synchronizedList()`.
- Or use `ConcurrentHashMap` for maps.
- The iterators from these collections are safe for concurrent access.

Would you like **practice programs** or **mock test questions** on this topic next?