ArrayList Usage with parallelStream(): When It's Safe and When It's Not

---

✅Safe Usage of ArrayList with `parallelStream()`

Pattern:

```
List<Integer> result = list.parallelStream()
    .filter(a -> a % 2 == 0)
    .map(a -> a * 2)
    .collect(Collectors.toList());
```

Or in Java 16+:

```
List<Integer> result = list.parallelStream()
    .filter(a -> a % 2 == 0)
    .map(a -> a * 2)
    .toList();
```

Why it's safe:

- No shared mutable state.
- Each element is processed completely by one thread.
- `collect()` or `toList()` handles thread safety internally using proper synchronization and result merging logic.

Thread behavior:

- Each thread handles its own subset of elements.
- All operations (`filter`, `map`, etc.) for a given element are executed **by the same thread**.
- No race conditions occur.

❌Unsafe Usage of ArrayList with `parallelStream()`

Pattern:

```
List<Integer> unsafeList = new ArrayList<>();
list.parallelStream()
    .filter(a -> a % 2 == 0)
    .map(a -> a * 2)
    .forEach(unsafeList::add);
```

Why it's unsafe:

- `unsafeList` is a shared mutable object (not thread-safe).
- Multiple threads try to write to the same `ArrayList` concurrently.
- Can lead to:
- Lost or duplicated elements
- Corrupted internal state
- `ArrayIndexOutOfBoundsException`
- Incorrect size after operations

Even if it seems fine sometimes, this is just due to timing — under different workloads, errors can surface.

⚠️Example of Failure

```
List<Integer> unsafeList = new ArrayList<>();
IntStream.range(0, 10000).parallel().forEach(unsafeList::add);
System.out.println("Expected: 10000, Actual: " + unsafeList.size());
```

May output: `Expected: 10000, Actual: 9972` or throw an exception.

✅Safe Alternatives

1. **Use** `** or **`
2. **Use thread-safe collections if necessary:**

```
List<Integer> safeList = Collections.synchronizedList(new ArrayList<>());
list.parallelStream()
    .filter(a -> a % 2 == 0)
    .map(a -> a * 2)
    .forEach(safeList::add);
```

Note: Even this is not ideal — prefer `collect()` when possible.

✅Summary

Use Case	Safe?	Recommendation
`collect(Collectors.toList())`	✅Yes	Preferred
`.toList()` (Java 16+)	✅Yes	Preferred
`forEach(arrList::add)` on shared ArrayList	❌No	Avoid

Use Case	Safe?	Recommendation
`forEach()` on synchronized list	⚠️Safer	But not ideal

✅ Best Practice: Avoid shared mutable state when using `parallelStream()`. Use `collect()` to safely aggregate results.