Mistake & Solution

Java Streams 10 Mistakes to Avoid

Palmurugan C
@palmuruganc

Java Streams Mistakes

❌ Mistake-1

Ignoring *Optional* when using *findFirst()* or *findAny()*

```
String result = list.stream()
        .filter( String s → s.startsWith("A"))
        .findFirst().get();
```

✅ Solution

```
Optional<String> res = list.stream()
        .filter( String s → s.startsWith("A"))
        .findFirst();
res.ifPresent(System.out::println);
```

Java Streams Mistakes

✗ Mistake-2

Modifying State of External Variables

```
List<Integer> numbers = new ArrayList<>();
Stream.of( ...values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
        .filter( Integer num → num % 2 == 0)
        .forEach(numbers::add);
```

✓ Solution

Use collectors:

```
List<Integer> result =
        Stream.of( ...values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
                .filter( Integer num → num % 2 == 0)
                .toList();
```

Java Streams Mistakes

❌ Mistake-3

Using Streams for Simple Iterations

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.forEach(System.out::println);
```

✅ Solution

Use streams only for tasks like filtering, mapping, or reducing.

Java Streams Mistakes

✗ Mistake-4

Forgetting *.close()* on Stream Sources

```
  lines = Files.lines(filePath);
lines.forEach(System.out::println);
```

✓ Solution

Use try-with-resources

```
try (Stream<String> lines = Files.lines(path)) {
    lines.forEach(System.out::println);
}
```

Java Streams Mistakes

❌ Mistake-5

Using ***limit()*** Without ***sorted()***

- Calling .limit() before .sorted() may result in incorrect data if the order matters.

✅ Solution

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
names.stream()
        .sorted()
        .forEach(System.out::println);
```

Java Streams Mistakes

❌ Mistake-6

Misusing *.collect()* for Mutable Reduction

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
names.stream()
        .sorted()
        .collect(Collectors.toList());
```

✅ Solution

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
List<String> res = names.stream()
        .sorted()
        .toList();
```

Java Streams Mistakes

❌ Mistake-7

Not Handling Nulls in Streams

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
names.stream().filter(Objects::nonNull);
```

✅ Solution

- Preprocess collections to remove nulls, or filter them explicitly.

Java Streams Mistakes

❌ Mistake-8

Inefficient Use of Intermediate Operations

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
List<String> result = names.stream()
        .map(String::toLowerCase)
        .map( String name → name.substring(0, 1).toUpperCase() + name.substring( beginIndex: 1))
        .map(String::trim)
        .collect(Collectors.toList());
```

✅ Solution

Combine operations where possible to reduce overhead.

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
List<String> result = names.stream()
        .map( String name →
                name.substring(0, 1).toUpperCase() + name.substring( beginIndex: 1).toLowerCase())
        .toList();
```

Java Streams Mistakes

❌ Mistake-9

Misunderstanding Terminal Operations

- ***.forEach()*** is not the same as ***.map() + .collect()***.

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");

List<String> upperCaseNames = new ArrayList<>();
names.stream()
        .map(String::toUpperCase)
        .forEach(upperCaseNames::add); // Not the intended use of forEach()
```

✅ Solution

Use terminal operations like .toList() for transformations, and .forEach() for side effects.

```
List<String> names = Arrays.asList("Alice", "Charlie", "Bob");
List<String> upperCaseNames = names.stream()
        .map(String::toUpperCase)
        .toList();
```

Java Streams Mistakes

❌Mistake-10

Overusing Parallel Streams

- Using ***.parallel()*** indiscriminately can degrade performance, especially for **small datasets** or **I/O-heavy tasks**.

✅Solution

- Use parallel streams only for **large, CPU-bound tasks** where parallelism adds value.

Follow Me

"Stay ahead with cutting-edge technical tips and best practices—follow me on LinkedIn today!"

@palmuruganc