Stream API

Legend: 🟡 Intermediate • 🟢 Terminal • 🟠 Short-Circuit • 🔁 Stateful

Creation

- `Stream.of(T... vals)` → from values
- `Stream.iterate(seed, f)` → infinite sequence: $\text{seed}, f(\text{seed}), f^2(\text{seed}), \ldots\ldots$
- `Stream.iterate(seed, hasNext, f)` → finite version
- `Stream.generate(Supplier)` → infinite stream from supplier
- `Stream.concat(a, b)` → joins two streams

Transformations (Intermediate)

- `filter(Predicate)` 🟡 → only elements matching predicate
- `map(Function)` 🟡 → transform each element
- `flatMap(Function<T, Stream<R>>)` 🟡 → map + flatten
- `distinct()` 🟡🔁 → remove duplicates
- `sorted()` / `sorted(Comparator)` 🟡🔁 → natural/custom sort
- `peek(Consumer)` 🟡 → debug/tap into pipeline
- `limit(n)` 🟡🔁🟠 → first `n` elements
- `takeWhile(Predicate)` 🟡🔁🟠 → stop at 1st false
- `dropWhile(Predicate)` 🟡🔁 → skip until 1st false

Terminals

- `forEach(Consumer)` 🟢 → perform action for each
- `count()` 🟢 → # of elements
- `min(Comparator)` 🟢 → `Optional<T>` min element
- `toList()` 🟢 → materialize as list

Match Operations (Terminal + Short-circuit)

- `anyMatch(Predicate)` 🟢🟠 → true if any match
- `allMatch(Predicate)` 🟢🟠 → true if all match
- `noneMatch(Predicate)` 🟢🟠 → true if none match

Reduce Variants (Terminal)

- `reduce(id, BinaryOp)` 🟢 → accumulate with identity
- `reduce(BinaryOp)` 🟢 → `Optional<T>`, no id
- `reduce(id, BiFn, combiner)` 🟢 → `U` type parallel reduction