Here’s a **cheat sheet** of the main Java Stream operations grouped by type:

**1️⃣ Creating Streams**

* Stream.of(T...)
* Arrays.stream(array)
* list.stream()
* Stream.empty()
* Stream.generate()
* Stream.iterate()

**2️⃣ Intermediate Operations *(return a new Stream)***

**Filtering & Slicing**

* filter(Predicate) → keep elements matching a condition
* distinct() → remove duplicates
* limit(n) → keep first n elements
* skip(n) → skip first n elements

**Mapping & Transforming**

* map(Function) → transform elements
* mapToInt, mapToLong, mapToDouble → primitive specializations
* flatMap(Function) → flatten nested streams

**Sorting**

* sorted() → natural order
* sorted(Comparator) → custom order

**Peeking**

* peek(Consumer) → debug/log elements in the pipeline

**3️⃣ Terminal Operations *(produce a result and close the stream)***

**Matching** *(short-circuiting)*

* anyMatch(Predicate) → true if at least one matches
* allMatch(Predicate) → true if all match
* noneMatch(Predicate) → true if none match

**Finding** *(short-circuiting)*

* findFirst() → Optional first element
* findAny() → Optional element (useful in parallel streams)

**Reduction**

* forEach(Consumer) / forEachOrdered() → iterate elements
* toArray() → collect into array
* reduce() → combine into a single value
* collect(Collectors...) → collect into list, set, map, etc.

**Statistics**

* count() → number of elements
* min(Comparator) → smallest element
* max(Comparator) → largest element

**4️⃣ Short-Circuiting Operations**

These can end the stream early:

* limit(n)
* anyMatch, allMatch, noneMatch
* findFirst, findAny

**can we join 2 streams into 1 ?**

Yes ✅ — you can join two streams in Java using **Stream.concat()** or by combining them into a collection first.

**Example with Stream.concat():**

Stream<String> s1 = Stream.of("A", "B");

Stream<String> s2 = Stream.of("C", "D");

Stream<String> combined = Stream.concat(s1, s2);

combined.forEach(System.out::println);

// Output: A B C D

**Using Stream.of() + flatMap():**

Stream<String> merged = Stream.of(s1, s2)

.flatMap(s -> s);

⚠ **Note:** A stream can be consumed only once — you can’t reuse s1 or s2 after they’ve been joined.

**What is “groupingBy” in streams ?**

groupingBy in Java Streams is a **collector** that groups elements of a stream into a Map based on a **classification function**.

It comes from Collectors.groupingBy() in java.util.stream.Collectors.

**Basic example – group strings by length:**

List<String> words = Arrays.asList("apple", "banana", "cherry", "dog");

Map<Integer, List<String>> grouped =

words.stream()

.collect(Collectors.groupingBy(String::length));

System.out.println(grouped);

// {3=[dog], 5=[apple], 6=[banana, cherry]}

* **Key** → length of the string
* **Value** → list of words with that length

**Advanced example – group and count:**

Map<Integer, Long> countByLength =

words.stream()

.collect(Collectors.groupingBy(

String::length,

Collectors.counting()

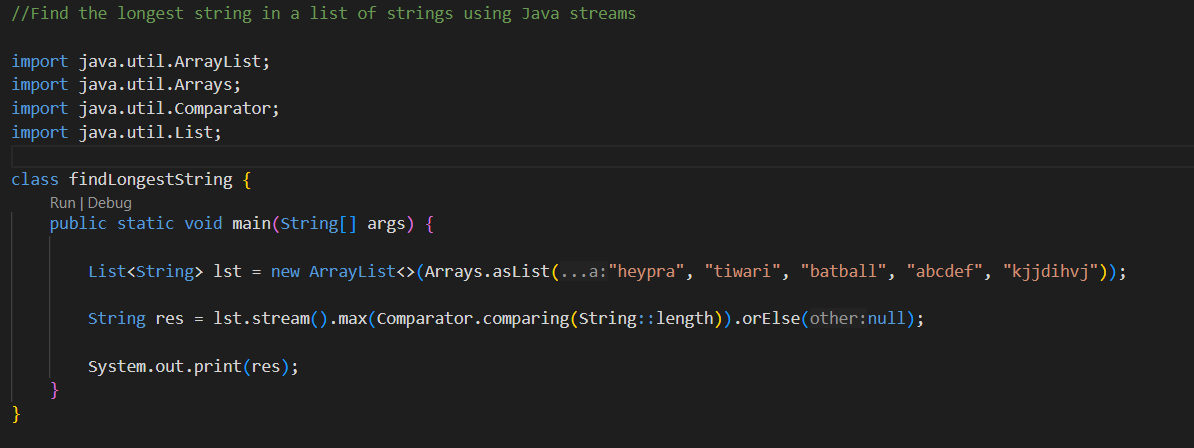
));

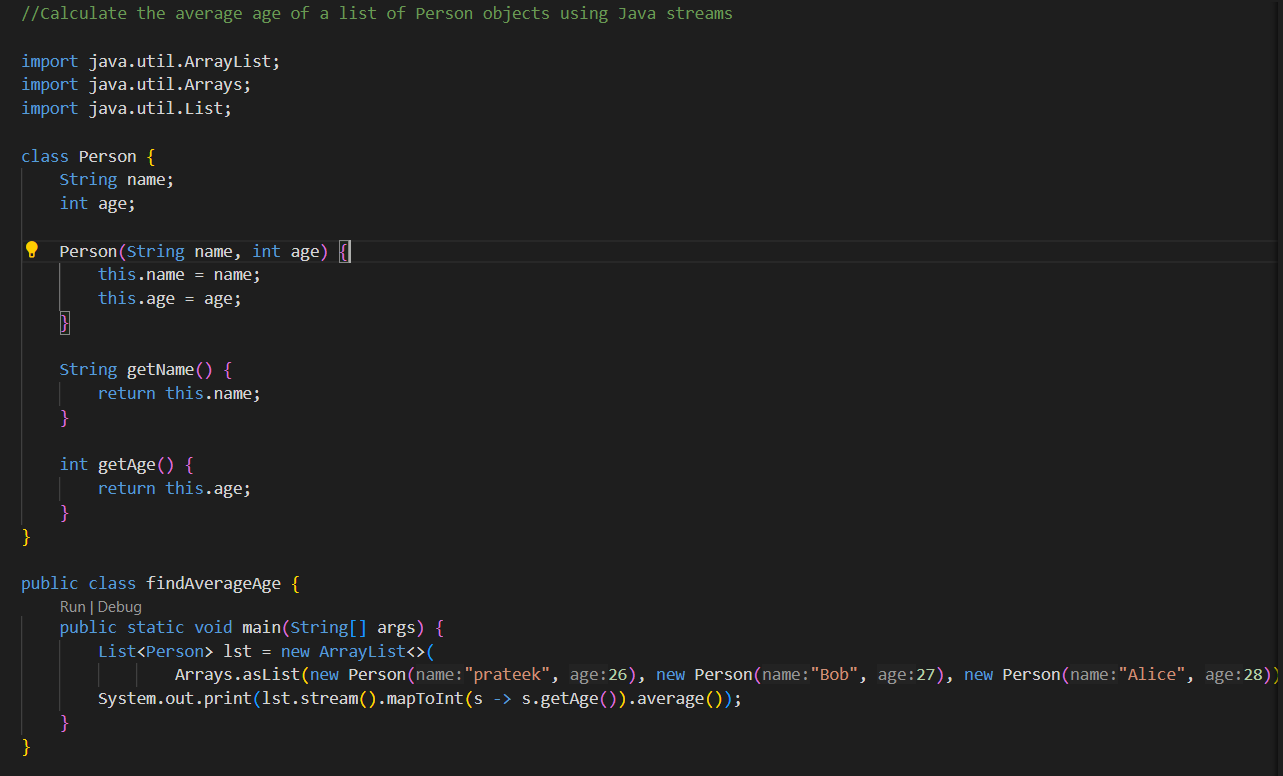
System.out.println(countByLength);

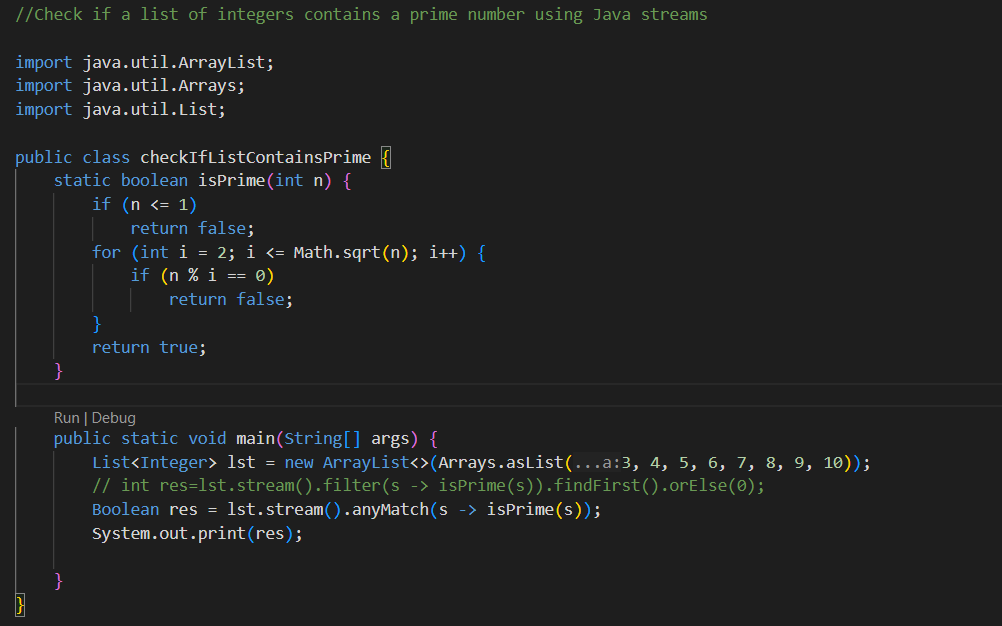
// {3=1, 5=1, 6=2}

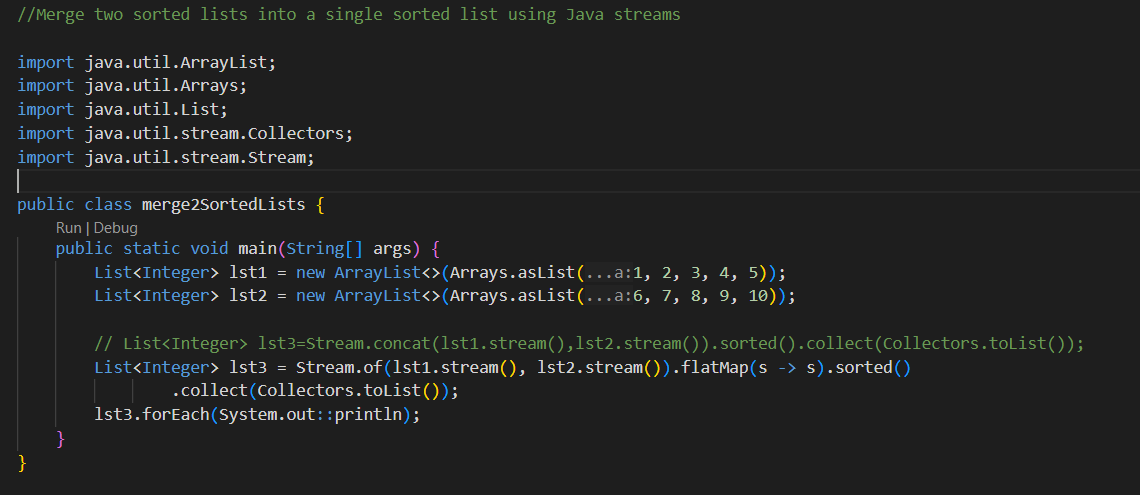
**Key points:**

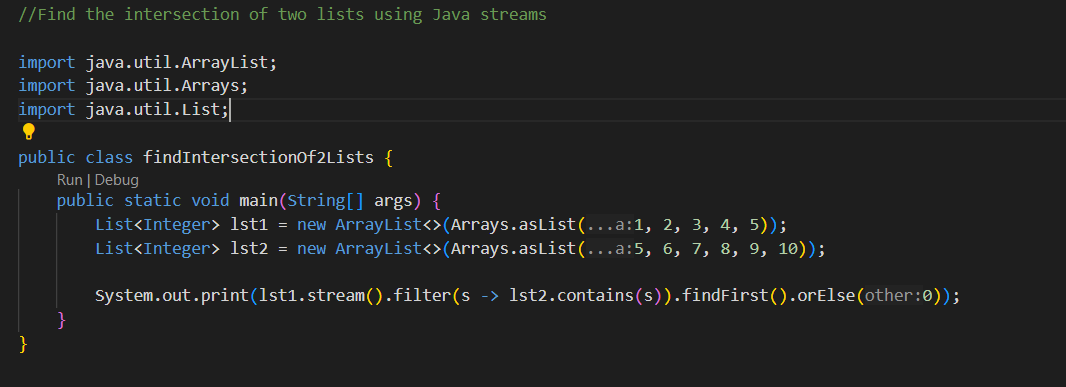
* First argument: classifier function (how to group)
* Second (optional) argument: downstream collector (e.g., counting(), mapping())

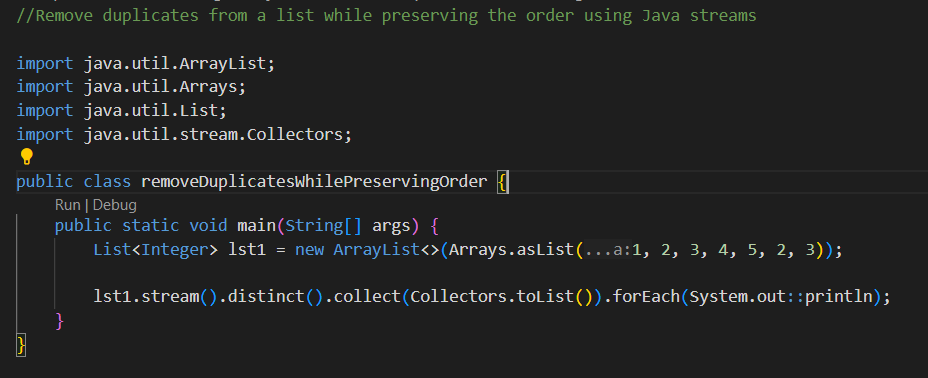












Given a list of transactions, find the sum of transaction amounts for each day using Java streams

