List<Employee> lst= emp.stream().filter(e->e.age()>50 && e.age()<70).collect(Collectors.toList());

List<String> names = Arrays.asList("Sam", "Steve", "John", "Sarah", "Mike", "susan");

// Filter names starting with "S" (case-sensitive)

List<String> filtered = names.stream() .filter(name -> name.startsWith("S"))

.collect(Collectors.toList());

//.filter(s -> s.startsWith("S"))

.map(String::toUpperCase)

double maxSalary = employees.stream()

.mapToDouble(Employee::getSalary)

.max()

.orElse(0.0); // default if list is empty

// Step 2: Filter employees having max salary

List<Employee> highestPaidEmployees = employees.stream()

.filter(emp -> emp.getSalary() == maxSalary)

.collect(Collectors.toList());

Find departmnert it

List<Employee> itEmployees = employees.stream()

.filter(emp -> "IT".equalsIgnoreCase(emp.getDepartment()))

.collect(Collectors.toList());

Optional<Double> secondHighestSalary = employees.stream()

.map(Employee::getSalary)

.distinct()

.sorted(Comparator.reverseOrder())

.skip(1)

.findFirst();

// 🔹 Step 2: Filter employees by that salary

if (secondHighestSalary.isPresent()) {

double salary = secondHighestSalary.get();

List<Employee> secondHighestEmployees = employees.stream()

.filter(emp -> emp.getSalary() == salary)

.collect(Collectors.toList());

AnyMatchExample

List<String> names = Arrays.asList("Steve", "John", "Mike", "Sarah");

boolean hasNameStartingWithS = names.stream().anyMatch(name -> name.startsWith("S"));

List<Integer> numbers = Arrays.asList(10, 20, 30, 40);

int sum = numbers.stream().reduce(0, Integer::sum); // (identity, accumulator)

List<String> words = Arrays.asList("Java", "Stream", "API");

String result = words.stream().reduce("", (a, b) -> a + " " + b);

------------------MAX and MIN SAL----------------------------------------

Optional<Employee> maxSalaryEmp = employees.stream()

.max(Comparator.comparing(Employee::getSalary));

Optional<Employee> minSalaryEmp = employees.stream()

.min(Comparator.comparing(Employee::getSalary));

-----------------------------FlatMapExample ------------------

List<List<String>> listOfLists = Arrays.asList(

Arrays.asList("Java", "Spring"),

Arrays.asList("Python", "Flask"),

Arrays.asList("C++", "Qt")

);

List<String> flatList = listOfLists.stream()

.flatMap(List::stream) // flatten inner lists

.collect(Collectors.toList());

**✅ 1. Transformation (Intermediate) Operations**

These **transform a stream** into another stream — they are **lazy** (i.e., nothing happens until a terminal operation is invoked).

**🔁 Common Intermediate Operations:**

| **Operation** | **Description** | **Example** |
| --- | --- | --- |
| filter() | Filters elements based on a predicate | .filter(e -> e.getSalary() > 50000) |
| map() | Transforms each element | .map(e -> e.getName()) |
| flatMap() | Flattens nested streams | .flatMap(list -> list.stream()) |
| distinct() | Removes duplicates | .distinct() |
| sorted() | Sorts the stream | .sorted(Comparator.comparing(Employee::getName)) |
| limit(n) | Limits the stream to n elements | .limit(5) |
| skip(n) | Skips first n elements | .skip(2) |
| peek() | Debug/log without modifying | .peek(System.out::println) |

**✅ 2. Terminal Operations**

These **trigger** the processing of the stream pipeline and **produce a result** (or side effect).

**🔁 Common Terminal Operations:**

| **Operation** | **Description** | **Example** |
| --- | --- | --- |
| collect() | Gathers the stream into a collection | .collect(Collectors.toList()) |
| forEach() | Performs action on each element | .forEach(System.out::println) |
| count() | Counts the elements | .count() |
| reduce() | Reduces to a single value | .reduce(0, Integer::sum) |
| min(), max() | Finds min/max element | .max(Comparator.comparing(Employee::getSalary)) |
| anyMatch() / allMatch() / noneMatch() | Match operations | .anyMatch(e -> e.getSalary() > 100000) |
| findFirst() / findAny() | Returns Optional | .findFirst() |

**🧠 Stream Lifecycle Example:**

java

CopyEdit

List<String> names = Arrays.asList("Steve", "Sam", "John", "Sarah");

List<String> result = names.stream() // source

.filter(name -> name.startsWith("S")) // transformation

.map(String::toUpperCase) // transformation

.sorted() // transformation

.collect(Collectors.toList()); // terminal

**📝 Summary:**

| **Type** | **Examples** | **Lazy?** |
| --- | --- | --- |
| Transformation | map(), filter(), sorted() | Yes |
| Terminal | collect(), forEach(), count() | No |