Prompt:

consider the business case scenario and give simple question for employee list. with stream api, how one can filter the name that starts with M. Give expected output. similarly give 5 important questions on the stream api showing terminal operation, shortcircuit without terminal and intermediate operation.

Here’s a simple business case scenario involving an employee list and related questions demonstrating various Stream API operations in Java:

**Scenario:**

You are developing an employee management system for a company. The system contains a list of employees, each with their name, department, and salary. Using the Java Stream API, you need to perform various operations on this list, such as filtering, mapping, and performing terminal operations.

**Employee Class:**

java

Copy code

public class Employee {

private String name;

private String department;

private double salary;

// Constructor, Getters, Setters, toString() method

}

**1. Simple Question: Filter Employee Names that Start with "M"**

**Problem:**

Using the Stream API, how can you filter and collect the names of employees that start with the letter "M"?

**Solution:**

java

Copy code

List<Employee> employees = // your employee list

List<String> filteredNames = employees.stream()

.map(Employee::getName)

.filter(name -> name.startsWith("M"))

.collect(Collectors.toList());

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

**Expected Output (Example):**

If the employee list has names like "Mark", "John", "Molly":

css

Copy code

Mark

Molly

**2. Question: Demonstrate Terminal Operation (e.g., count())**

**Problem:**

How would you use the Stream API to count the number of employees who belong to the "IT" department?

**Solution:**

java

Copy code

long itEmployeeCount = employees.stream()

.filter(employee -> employee.getDepartment().equals("IT"))

.count();

System.out.println("Number of IT employees: " + itEmployeeCount);

**Explanation:**

* count() is a terminal operation that returns the count of elements in the stream that match the given predicate.

**3. Question: Show a Short-circuit Operation (e.g., findAny())**

**Problem:**

Using the Stream API, how can you find any employee who earns more than 100,000?

**Solution:**

java

Copy code

Optional<Employee> richEmployee = employees.stream()

.filter(employee -> employee.getSalary() > 100000)

.findAny();

richEmployee.ifPresent(System.out::println);

**Explanation:**

* findAny() is a short-circuit terminal operation. It stops as soon as it finds an element that matches the filter condition.

**4. Question: Demonstrate an Intermediate Operation (e.g., map())**

**Problem:**

How can you use the Stream API to get a list of all employee names in uppercase?

**Solution:**

java

Copy code

List<String> upperCaseNames = employees.stream()

.map(employee -> employee.getName().toUpperCase())

.collect(Collectors.toList());

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

**Explanation:**

* map() is an intermediate operation that transforms each element of the stream (in this case, converts names to uppercase).

**5. Question: Illustrate the Use of anyMatch() (Short-circuit Terminal Operation)**

**Problem:**

How can you check if there is any employee with a salary greater than 200,000 using the Stream API?

**Solution:**

java

Copy code

boolean hasHighEarner = employees.stream()

.anyMatch(employee -> employee.getSalary() > 200000);

System.out.println("Is there any employee with salary > 200000? " + hasHighEarner);

**Explanation:**

* anyMatch() is a short-circuit terminal operation that returns true if any element matches the given predicate.

**6. Question: Combining Intermediate Operations (e.g., filter(), map())**

**Problem:**

How can you use the Stream API to get the names of employees who are in the "HR" department and have a salary less than 50,000?

**Solution:**

java

Copy code

List<String> hrLowEarners = employees.stream()

.filter(employee -> employee.getDepartment().equals("HR") && employee.getSalary() < 50000)

.map(Employee::getName)

.collect(Collectors.toList());

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

**Explanation:**

* filter() is an intermediate operation that filters employees based on the condition. The result is further processed using map() to extract names.

These questions help you demonstrate the core functionalities of Java's Stream API, including terminal operations, intermediate operations, and short-circuiting behaviors.