Stream API

Stream api basics

```java
package corejava.suyash.stream;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class StreamDemo {
    public static void main(String[] args) {
        // feature introduced in Java 8
        // process collections of data in a functional and declarative manner
        // Simplify Data Processing
        // Embrace Functional Programming
        // Improve Readability and Maintainability
        // Enable Easy Parallelism

        //// What is stream ?
        // a sequence of elements supporting functional and declarative programing

        //// How to Use Streams ?
        // Source, intermediate operations & terminal operation

        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        System.out.println(numbers.stream().filter(x -> x % 2 == 0).count());

        //// Creating Streams
        // 1. From collections
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
        Stream<Integer> stream = list.stream();
        // 2. From Arrays
        String[] array = {"a", "b", "c"};
        Stream<String> stream1 = Arrays.stream(array);
        // 3. Using Stream.of()
        Stream<String> stream2 = Stream.of("a", "b");
        // 4. Infinite streams
        Stream.generate(() -> 1);
        Stream.iterate(1, x -> x + 1);
    }
}
```

Intermediate operator

```java
package corejava.suyash.stream;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class IntermediateOps {
    public static void main(String[] args) {
        // Intermediate operations transform a stream into another stream
        // They are lazy, meaning they don't execute until a terminal operation is invoked.

        // 1. filter
        List<String> list = Arrays.asList("Akshit", "Ram", "Shyam", "Ghanshyam", "Akshit");
        Stream<String> filteredStream = list.stream().filter(x -> x.startsWith("A"));
        // no filtering at this point
        long res = list.stream().filter(x -> x.startsWith("A")).count();
        System.out.println(res);

        // 2.map
        Stream<String> stringStream = list.stream().map(x -> x.toUpperCase);

        // 3. sorted
        Stream<String> sortedStream = list.stream().sorted();
        Stream<String> sortedStreamUsingComparator = list.stream().sorted((a, b) -> a.length() - b.length());

        // 4. distinct -> removes duplicate
        System.out.println(list.stream().filter(x -> x.startsWith("A")).distinct().count());

        // 5. limit
        System.out.println(Stream.iterate(1, x -> x + 1)
                           .limit(100) // only 100
                           .count());

        // 6. skip
        System.out.println(Stream.iterate(1, x -> x + 1)
                           .skip(10) // skip first ten
                           .limit(100) // from 11 to 110, starts from 11
                           .count());

        // 7. peek
        // Performs an action on each element as it is consumed.
        Stream.iterate(1, x -> x + 1).skip(10).limit(100).peek(System.out::println).count();

        // 8. flatMap
        // Handle streams of collections, lists, or arrays where each element is itself a collection
        // Flatten nested structures (e.g., lists within lists) so that they can be processed as a single sequence of elements
        // Transform and flatten elements at the same time.
        List<List<String>> listOfLists = Arrays.asList(
                Arrays.asList("apple", "banana"),
                Arrays.asList("orange", "kiwi"),
                Arrays.asList("pear", "grape")
        );
        System.out.println(listOfLists.get(1).get(1));
        System.out.println(listOfLists.stream().flatMap(x -> x.stream()).map(String::toUpperCase).toList());
        List<String> sentences = Arrays.asList(
                "Hello world",
                "Java streams are powerful",
                "flatMap is useful"
        );
        System.out.println(sentences
                .stream()
                .flatMap(sentence -> Arrays.stream(sentence.split(" ")))
                .map(String::toUpperCase)
                .toList();


    }
}
```

Terminal operators

```java
package corejava.suyash.stream;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class TerminalOps {
    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3);

        // 1. collect
        list.stream().skip(1).collect(Collectors.toList());
        list.stream().skip(1).toList(); // in new java version

        // 2. forEach
        list.stream().forEach(x -> System.out.println(x));

        // 3. reduce : Combines elements to produce a single result
        Optional<Integer> optionalInteger = list.stream().reduce((x,y) ->  x+y));
      //optional may have value or may not
        System.out.println(optionalInteger.get());

        // 4. count
        long res = list.stream().filter(x -> x%2==0).count();
        System.out.println(res);

      // 5. anyMatch, allMatch, noneMatch
//Short circuit operation, if mind match they end loop
      //check if any number is even in list
        boolean b = list.stream().anyMatch(x -> x % 2 == 0);
        System.out.println(b);
      // check if all elements are greater than 0
        boolean b1 = list.stream().allMatch(x -> x > 0);
        System.out.println(b1);
      // check if no element is negative in list
        boolean b2 = list.stream().noneMatch(x -> x < 0);
        System.out.println(b2);

        // 6. findFirst, findAny
      //Short circuit operation, if mind match they end loop
        System.out.println(list.stream().findFirst().get());
        System.out.println(list.stream()
                           .findAny() // brings any element of list
                           .get());

        // 7. toArray()

        Object[] array = Stream.of(1, 2, 3).toArray();

        // 8. min / max
        System.out.println("max: " + Stream.of(2, 44, 69).max((o1, o2) -> o2 - o1));
        System.out.println("min: " + Stream.of(2, 44, 69).min(Comparator.naturalOrder()));

        // 9. forEachOrdered
        List<Integer> numbers0 = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        System.out.println("Using forEach with parallel stream:");
        numbers0.parallelStream().forEach(System.out::println);
        System.out.println("Using forEachOrdered with parallel stream:");
        numbers0.parallelStream().forEachOrdered(System.out::println);



        // Example: Filtering and Collecting Names
        List<String> names = Arrays.asList("Anna", "Bob", "Charlie", "David");
        System.out.println(names.stream().filter(x -> x.length() > 3).toList());

        // Example: Squaring and Sorting Numbers
        List<Integer> numbers = Arrays.asList(5, 2, 9, 1, 6);
        System.out.println(numbers.stream().map(x -> x * x)
                           .sorted() //ascending order
                           .toList());

        // Example: Summing Values
        List<Integer> integers = Arrays.asList(1, 2, 3, 4, 5);
        System.out.println(integers.stream().reduce((x,y) ->  x+y) ).get());

        // Example:  Counting Occurrences of a Character
        String sentence = "Hello world";
      //cahrs() is used as Arrays.stream do not take char array.
        System.out.println(sentence.chars().filter(x -> x == 'l').count());

        // Example
        // Streams cannot be reused after a terminal operation has been called
        Stream<String> stream = names.stream();
        stream.forEach(System.out::println);
//        List<String> list1 = stream.map(String::toUpperCase).toList(); // exception

        // stateful & stateless



    }
}
```

After that go with Employee collection list.