Need Of Stream API

👉 Introduction:

Stream API introduced in Java 1.8 version. It is used for **processing collections of data in a functional way**. It allows performing operations like filtering, mapping, and reducing data with **less code** and **better performance**.

Example using a for loop:

1. Suppose we have a list of integers and want to process it step by step.
   - ➢ We initialize a list of integers using Arrays.asList().

```java
import java.util.Arrays;
import java.util.List;

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

        // Step 1: Create a list of integers
        List<Integer> nums = Arrays.asList(4, 5, 7, 3, 2, 6);

    }
}
```

2. Filter out only the even numbers and ignore the odd ones.
   - ➢ We iterate through the list and check if each number is even using if (n % 2 == 0).

```java
import java.util.Arrays;
import java.util.List;

public class Demo {
    public static void main(String[] args) {
        // Step 1: Create a list of integers
        List<Integer> nums = Arrays.asList(4, 5, 7, 3, 2, 6);

        // Step 2: Iterate through the list and filter even numbers
        for (int n : nums) {
            if (n % 2 == 0) {
                // Even number found, process it in the next steps
            }
        }
    }
}
```

3. Double each of the filtered numbers.
   - If the number is even, we multiply it by 2.

```java
import java.util.Arrays;
import java.util.List;

public class Demo {
    public static void main(String[] args) {
        // Step 1: Create a list of integers
        List<Integer> nums = Arrays.asList(4, 5, 7, 3, 2, 6);

        // Step 2: Iterate through the list and filter even numbers
        for (int n : nums) {
            if (n % 2 == 0) {
                // Step 3: Double the even number
                n = n * 2;
            }
        }
    }
}
```

4. Calculate the sum of all the modified numbers to get the final result.
   - We add the modified numbers to a sum variable and print the final result.

```java
import java.util.Arrays;
import java.util.List;

public class Demo {
    public static void main(String[] args) {
        // Step 1: Create a list of integers
        List<Integer> nums = Arrays.asList(4, 5, 7, 3, 2, 6);
        // Step 2: Initialize sum variable
        int sum = 0;
        // Step 3: Iterate through the list, filter even numbers, and modify them
        for (int n : nums) {
            if (n % 2 == 0) {
                // Step 3: Double the even number
                n = n * 2;
            }
            // Step 4: Add the modified number to the sum
            sum += n;
        }
        // Print the final sum
        System.out.println(sum);
    }
}
```

👉 Limitations of for loop & enhanced for loop:

- **Verbose Code** – Requires explicit iteration and condition checks, making the code longer and less readable.
- **Manual Filtering & Transformation** – Needs extra logic for filtering and modifying data, while Streams handle it directly.
- **No Built-in Reduction** – Summing or aggregating values requires manual accumulation.
- **Sequential Execution** – Loops run sequentially by default, whereas Streams support parallel execution easily.
- **No Internal Optimization** – Streams optimize execution with lazy evaluation, but loops process data immediately.

👉 Using Stream API:

```java
import java.util.Arrays;
import java.util.List;

public class Demo {
    public static void main(String[] args) {
        List<Integer> nums = Arrays.asList(4, 5, 7, 2);

        int sum = nums.stream()
            .filter(n -> n % 2 == 0)  // Select even numbers
            .map(n -> n * 2)          // Double the selected numbers
            .reduce(0, Integer::sum); // Sum up the modified values

        System.out.println(sum);
    }
}
```

👉 Explanation:

- **stream()** – Converts the list into a stream for processing.
- **filter(n -> n % 2 == 0)** – Filters out only even numbers.
- **map(n -> n * 2)** – Doubles each filtered number.
- **reduce(0, Integer::sum)** – Accumulates the transformed numbers into a sum.