# Java 8 Questions

# Explain Latest features of Java 8?

# What is the uses of Optional in java?

# What changes have been made in Hashmap in Java8?

# What is Stream API?

# What is benefit of using Stream API over normal for loops?

* 1. <https://www.onlinetutorialspoint.com/java/parallel-streams-java-8.html>
  2. Suppose let’s take a scenario of you having a list of employee objects and you have to count employees whose salary is above 15000. Generally, to solve this problem you will iterate over list going through each and every employee and checking if employees salary is above 15000. This takes O(N) time since you go sequentially.
  3. Streams provide us with the flexibility to iterate over the list in a parallel pattern and can give the aggregate in quick fashion.
  4. Stream implementation in Java is by default sequential unless until it is explicitly mentioned for parallel. When a stream executes in parallel, the Java runtime partitions the stream into multiple substreams. Aggregate operations iterate over and process these sub-streams in parallel and then combine the results.
  5. The only thing to keep in mind to create parallel stream is to call parallelStream() on the collection else by default sequential stream gets returned by stream().
  6. **Example on Parallel Streams :**

import java.util.ArrayList;

import java.util.List;

public class ParallelStream {

public static void main(String[] args) {

List < Employee > empList = new ArrayList < Employee > ();

for (int i = 0; i < 100; i++) {

empList.add(new Employee("A", 20000));

empList.add(new Employee("B", 3000));

empList.add(new Employee("C", 15002));

empList.add(new Employee("D", 7856));

empList.add(new Employee("E", 200));

empList.add(new Employee("F", 50000));

}

long t1 = System.currentTimeMillis();

System.out.println("Sequential Stream count: " + empList.stream().filter(e -> e.getSalary() > 15000).count());

long t2 = System.currentTimeMillis();

System.out.println("Sequential Stream Time taken:" + (t2 - t1));

t1 = System.currentTimeMillis();

System.out.println("Parallel Stream count: " + empList.parallelStream().filter(e -> e.getSalary() > 15000).count());

t2 = System.currentTimeMillis();

System.out.println("Parallel Stream Time taken:" + (t2 - t1));

}

}

**Employee.java**

class Employee {

private int salary;

private String name;

Employee(String name, int salary) {

this.name = name;

this.salary = salary;

}

public int getSalary() {

return salary;

}

public void setSalary(int salary) {

this.salary = salary;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

* 1. We have created a list of 600 employees out of which there are 300 employees whose salary is above 15000.
  2. Creating a sequential stream and filtering elements it took above 40 milliseconds, whereas the parallel stream only took 4 milliseconds.
  3. Performance Implications:
  4. Parallel Stream has equal performance impacts as like its advantages. Since each substream is a single thread running and acting on the data, it has overhead compared to sequential stream. Inter-thread communication is dangerous and takes time for coordination
  5. When to use Parallel Streams:
  6. They should be used when the output of the operation is not needed to be dependent on the order of elements present in source collection (on which stream is created).
  7. Parallel Streams can be used in case of aggregate functions.
  8. Iterate over large sized collections.
  9. If you have performance implications with sequential streams.
  10. If your environment is not multi threaded, because parallel stream creates thread and can effect new requests coming in.

https://coderwall.com/p/21hetg/java-8-stream-api-vs-for-each-loop

<https://java2blog.com/java-8-parallel-stream/>

The concept of parallel stream to do parallel processing. As we have more number of cpu cores nowadays due to cheap hardware costs, parallel processing can be used to perform operation faster.

Let’s understand with help of simple example.

In case of Parallel stream,4 threads are spawned simultaneously and it internally using Fork and Join pool to create and manage threads.Parallel streams create **ForkJoinPool** instance via static ForkJoinPool.commonPool() method

Parallel Stream has much higher overhead than sequential Stream and it takes good amount of time to coordinate between threads.

You need to consider parallel Stream if and only if:

Simplest formula for measuring parallelism is "NQ" model as provided by Brian Goetz in his presentation.

NQ Model:N x Q > 10000 where,

N = number of items in dataset

Q = amount of work per item

It means if you have a large number of datasets and less work per item(For example: Sum), parallelism might help you run program faster and vice versa is also true. So if you have less number of datasets and more work per item(doing some computational work), then also parallelism might help you in achieving results faster

1. What is Parallel Stream? How it internally works?
2. **ForkJoinPool ?**

# What improvements have been made in Date Time Api?

# What is default method why it has been introduced in Java8?

# What is Functional Interface?

# Explain about different built in Functional interfaces?

# What is Lambda Expression? what are its usage?

# What is UnaryOperator, Supplier, Consumer, Predicate?

# Does Lambda contains return statement?

# Java 8 Stream findFirst() vs findAny()?

# Remove element from a List that satisfies a given Predicate?

# Stream without terminal Operation?