**QUESTIONS AND RESPONSE:**

1. **What are the key advantages of using Java Streams, and in what scenarios do you think they are most beneficial?**

Ans : Key advantages of using streams in Java are:

* **Parallel processing –** Streams provide facility to do parallel processing of data, improving performance on multicore processors.
* **Lazy Evaluation-** Streams can be lazily evaluated, improving efficiency by only processing elements as needed.
* **Declarative Programming-** Supports a more declarative and functional programming style.
* **Data Pipelines-** Streams allow us to chain multiple operations together into a pipeline. This is useful when you have a sequence of data transformations or manipulations.

1. **Explain the difference between intermediate and terminal operations in Java Streams. Provide examples of each.**

* **Intermediate operations:**
* Methods are chained together.
* Intermediate operations transform a stream into another stream.
* It enables the concept of filtering where one method filters data and passes it to another method after processing.

**Example: map(),filter(),sorted()**

List<String> words = Arrays.asList("Hello", "World");

Stream<String> upperCaseStream = words.stream().map(String::toUpperCase);

* **Terminal operations:**
* Operations that return the result.
* These Operations are not processed further just return a final result value.
* It enables the concept of filtering where one method filters data and passes it to another method after processing.

**Example: collect(),foreach(),reduce()**

List<String> collected = upperCaseStream.collect(Collectors.toList());

1. **When should you prefer using Streams over traditional loops for data processing tasks?**

* Stream is designed to handle large list of data
* Streams are often more readable, concise, and can take advantage of parallel processing.
* Streams are easier to maintain.

1. **What are the fundamental differences between a Map and a List in Java? When would you choose one over the other for a specific use case?**

**Map:**

* It is a collection in java which stores data as key-value pair. Keys cannot be duplicate.
* There is no ordering of data in a map.

**List**:

* List in java stores data in ordered form and allow duplicates.

Choose a Map when you need to associate a value with a unique key, and a List when order or duplicates matter.

1. **Can you explain some common design patterns and scenarios where using a Map would be more appropriate than using a List?**

* Use a Map when you need to perform frequent lookups based on keys.
* Maps are suitable for scenarios where we need to associate values with unique keys.

1. **How do you ensure the thread safety of Maps and Lists in a multi-threaded environment?**

We need to use synchronized block or synchronized methods if code is accessed by multiple threads. Only one thread can enter synchronized method or block by acquiring lock on the object at a time and hence avoid race condition.

1. For map use Collections.synchronizedMap() or ConcurrentHashMap
2. For List use Collections.synchronizedList()
3. **What is the Single Responsibility Principle, and why is it important in software design?**

SRP states that a class should have only one reason to change.

It promotes maintainability, flexibility, and understandability in software design.

1. **What is Spring Boot, and what are its key features and advantages over traditional Spring Framework?**

Spring Boot is an open-source Java framework for building web apps. It is built on the Spring Framework, which is an open-source, Java-based web application framework.

Key Features:

* Simplifies the configuration of Spring applications.
* Embedded HTTP server support.

Advantages:

* Faster development and deployment.
* Simplified configuration and setup.
* Reduced need for boilerplate code.

1. **How do you configure logging in a Spring Boot application, and what logging libraries does Spring Boot support?**

* The default logging framework used by Spring Boot is Logback, but it can easily be replaced with others like Log4j2 or Java Util Logging.
* By default, Spring Boot uses the src/main/resources/application.properties or src/main/resources/application.yml file for configuration.

1. **Explain the concept of "auto-configuration" in Spring Boot. How can you disable it if needed?**

Auto-configuration is a key feature of Spring Boot that simplifies the setup and configuration of Spring applications. Automatically configures the application based on the dependencies present.

1. Disable with @SpringBootApplication(exclude = {ClassToExclude.class}) .
2. Write in properties file spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration
3. **Describe the key principles of micro services architecture.**

* Every service in microservice has single responsibility.
* Distributed Computing.
* Resilience and Fault Tolerance.
* Multiple team can work parallel on small services. Faster time to market

1. **What are the benefits of using micro services over a monolithic architecture in large-scale applications?**

* Decoupling of large monolithic into independent services
* Helps in maintenance
* Independent deployment
* Less downtime
* Distributed Computing.
* Improve scalability

1. How do you handle communication between micro services? Explain the pros and cons of different communication mechanisms (e.g., REST, messaging queues).

We can handle communication between microservice using below mechanisms:

**REST**:

**Pros**:

* Simplicity and ease of use , widely adoptable
* Uniform interface – standard http methods ; GET,POST etc
* Statelessness

Cons:

* REST is primarily synchronous, meaning the client sends a request and waits for a response. This may lead to latency issues in scenarios with high loads or long processing times.
* Overhead of HTTP:

RESTful APIs often use HTTP as the communication protocol, which can introduce overhead, especially in scenarios where a lightweight protocol might be more suitable.

**Messaging Queues**: Asynchronous communication, decouples services.

**Pros**:

* Asynchronous Communication.
* Reliability
* Loose Coupling: Components in message queue don’t need to know each other.

CONS:

**Complexity**:Implementing and managing a messaging queue system can be more complex compared to RESTful communication.

**Message Order**:Ensuring the order of messages can be challenging, as messages are processed asynchronously and might arrive out of order.

**Learning Curve**:Working with messaging queues may have a steeper learning curve for developers unfamiliar with the concepts of message brokers and asynchronous communication.