Questions asked for Java, Spring boot, Microservices

1. How do you make micro service as fault tolerant?

   https://ubiminds.com/en-us/fault-tolerance/

   https://medium.com/cloud-native-daily/fault-tolerance-in-microservices-architecture-patterns-principles-and-techniques-explained-20cfa3d7f98f

2. How to make your spring boot application scalable, say process 10k request per second?

   In this case we can tell various optimization techniques like,

   - Implementing asynchronous programming - example utilizing asyc and completableFuture.
   - Reactive programming with webflux and mono
   - Enabling Db optimization like connection pooling & indexing
   - Caching using @cahable annotation or using Redis
   - Serialization optimization like replacing Jakson with Kryo
   - Load balancing and auto scaling for horizontal scaling (in kubernetes or cloud)
   - Load test to identify bottlenecks

3. How do you analyse memory leaks?

   https://medium.com/@AlexanderObregon/java-memory-leaks-detection-and-prevention-25a1c09eacbe

4. Git merge vs rebase?

Questions asked for Java, Spring boot, Microservices

1. How do you make micro service as fault tolerant?

   https://ubiminds.com/en-us/fault-tolerance/

   https://medium.com/cloud-native-daily/fault-tolerance-in-microservices-architecture-patterns-principles-and-techniques-explained-20cfa3d7f98f

2. How to make your spring boot application scalable, say process 10k request per second?

   In this case we can tell various optimization techniques like,

   - Implementing asynchronous programming - example utilizing asyc and completableFuture.
   - Reactive programming with webflux and mono
   - Enabling Db optimization like connection pooling & indexing
   - Caching using @cahable annotation or using Redis
   - Serialization optimization like replacing Jakson with Kryo
   - Load balancing and auto scaling for horizontal scaling (in kubernetes or cloud)
   - Load test to identify bottlenecks

3. How do you analyse memory leaks?

4. Git merge vs rebase?

5. Git cherry-picking
   Git cherry-picking is a command that allows you to apply specific commits from one branch to another. It is useful for selecting individual changes without merging the entire branch.

6. What is Pom
   POM stands for Project Object Model. It is an XML file used by Maven to manage project configuration, dependencies, and build instructions. The POM file contains information about the project, such as its group ID, artifact ID, version, and dependencies.

   The POM file is essential for Maven to understand how to build and manage the project. It defines the project's structure, dependencies, and build lifecycle, enabling Maven to automate the build process

7. Maven different scopes and uses?

   Maven uses dependency scopes to control the classpath and transitivity of dependencies. The six default scopes are:

   - **compile:** Default scope; dependencies are available in all classpaths.
   - **provided**: Dependencies are provided by the JDK or container at runtime.
   - **runtime**: Dependencies are not required for compilation but are needed during execution.
   - **test**: Dependencies are only available for testing.
   - **system**: Dependencies are provided by the JDK or container but are always available.
   - **import**: Dependencies are only available in the current POM and are not transitive.

8. How to resolve Maven dependency conflicts?

8. How to resolve Maven dependency conflicts?
   - **Identify Conflicts:** Use the mvn dependency:tree command to visualize the dependency tree and identify conflicts. The -Dverbose flag can provide more detailed information about conflicting dependencies.
   - **Exclude Transitive Dependencies:** Use the <exclusions> element in your POM file to exclude specific transitive dependencies that are causing conflicts.
   - **Dependency Management:** Define a <dependencyManagement> section in your root POM to specify the versions of dependencies to be used across the project, ensuring consistency and resolving conflicts.
   - **Tools**: Use tools like the Maven Enforcer Plugin to enforce rules and resolve conflicts automatically. The Maven Helper plugin in IntelliJ IDEA can also help visualize and resolve conflicts.
   - **Best Practices**: Regularly review and update your dependencies to avoid conflicts. Use the latest stable versions of libraries and ensure that your POM files are well-structured and maintainable

9. Given an array of country codes {"AU","NZ","CA","US","UK"}, print all country codes except AU using Java streams?

Answer :

```
import java.util.Arrays;

public class ExludeAnElementFromArray {

  public static void main(String[] args){

    String[] countryCodes={"AU","NZ","CA","US","UK"};
    String countryTobeExcluded="AU";

    Arrays.stream(countryCodes).toList().stream().filter(e->!e.equals(countryTobeExcluded))
```

10. Anagram" - An anagram is a word formed by rearranging the letters of a different word using all the original letters exactly once e.g. (silent -> listen).

Given two strings s1 and s2, we need a method to find the minimum number of manipulations required to make two strings anagram without adding/deleting any character. You can only modify characters.

Inputs :
s1 = "aba"
s2 = "baa"
Output : 0

Explanation: Both String contains identical characters

s1 = "abb"
s2 = "baa"
Output : 1

Explanation : Here, we need to change one character in either of the strings to make them identical. We can change 'b' in s1 or 'a' in s2.

Inputs :
s1 = "ddcf"
s2 = "cdek"

s2 = “ab”

Output : -1

Explanation : Length of the strings is different and so they can never be anagrams.

The anagram strings have same set of characters, sequence of characters can be different.

Assumption : The string only contains alphabets and it’s not case-sensitive.

Answer:

```
import java.util.Arrays;

public class AnagramManipulations {

  public static int minManipulations(String s1, String s2) {
    // If lengths are different, they can't be anagrams
    if (s1.length() != s2.length()) {
      return -1;
    }

    // Convert strings to lowercase to make it case-insensitive
    s1 = s1.toLowerCase();
    s2 = s2.toLowerCase();

    // Create frequency arrays for both strings
    int[] freq1 = new int[26];
    int[] freq2 = new int[26];

    // Count frequency of each character in both strings
    for (char c : s1.toCharArray()) {
      freq1[c - 'a']++;
    }
```

11. How do HTTPS work?

HTTPS (HyperText Transfer Protocol Secure) is an extension of HTTP that is used for secure communication over a computer network. It works by using SSL/TLS protocols to encrypt the data transmitted between the client (usually a web browser) and the server. Here's a simplified explanation:

1. Handshake: The client and server establish a secure connection through a process called the SSL/TLS handshake.
2. Certificate Exchange: The server sends its SSL certificate to the client. The certificate contains the server's public key.
3. Verification: The client verifies the certificate with a Certificate Authority (CA).
4. Session Key Generation: The client generates a session key, encrypts it with the server's public key, and sends it to the server.
5. Encrypted Communication: Both the client and server use the session key to encrypt and decrypt the data transmitted during the session.

12. What design pattern have I worked on?

Design patterns are typical solutions to common problems in software design. Some common design patterns include:

Singleton: Ensures a class has only one instance.
Factory: Creates objects without specifying the exact class of object that will be created.
Observer: Defines a onetomany dependency between objects.
Strategy: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
Decorator: Adds responsibilities to objects dynamically.

Implements vs Extends

Implements: Used to implement an interface. A class can implement multiple interfaces.
Extends: Used to inherit from another class. A class can extend only one class due to Java's single inheritance model.

data transmitted during the session.

12. What design pattern have I worked on?

Design patterns are typical solutions to common problems in software design. Some common design patterns include:

Singleton: Ensures a class has only one instance.
Factory: Creates objects without specifying the exact class of object that will be created.
Observer: Defines a onetomany dependency between objects.
Strategy: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
Decorator: Adds responsibilities to objects dynamically.

Implements vs Extends
Implements: Used to implement an interface. A class can implement multiple interfaces.
Extends: Used to inherit from another class. A class can extend only one class due to Java's single inheritance model.

13. How does a Spring Boot application run?

A Spring Boot application runs by creating a standalone, productiongrade Springbased application. Here's a basic overview:

1. **Main Class:** The application starts from a main class annotated with @SpringBootApplication.
2. **Embedded Server:** Spring Boot embeds a server (like Tomcat) within the application.
3. **Auto Configuration:** Spring Boot automatically configures the application based on the dependencies you have.

14. Do we need to create a singleton in Spring Boot?

No, you typically do not need to create a singleton manually in Spring Boot. Spring's default scope for beans is singleton, meaning that only one instance of the bean is created and shared throughout the application context.

15. What are immutable objects in Java and what are their benefits?

Immutable objects are objects whose state cannot be modified after they are created. In Java, immutable objects are achieved by:

Making the class final.
Making all fields private and final.
Initializing all fields via a constructor.
Not providing setter methods.

Benefits:

Thread Safety: Immutable objects are inherently threadsafe.
Predictability: The state of an immutable object is predictable and consistent.
Easier to Cache: Immutable objects can be easily cached.
Simpler Design: Immutable objects simplify the design and reduce the risk of unintended side effects.

16. Why is String immutable in Java?

The String class in Java is immutable for several reasons:

Security: Immutable strings are more secure, especially when used in multithreaded environments.
Thread Safety: Immutable objects are inherently threadsafe.
Caching: Immutable strings can be cached and reused, which improves performance.
Hash Code Caching: The hash code of an immutable string can be cached, making it more efficient for use in hash based collections.

17. How will you develop a microservice using Spring Boot?

18. Why and how will you use Spring Boot Actuator?

Spring Boot Actuator provides production ready features to monitor and manage your application. It includes endpoints to check the health, metrics, and other details of the application.

Why Use Actuator:

Monitoring: Provides endpoints to monitor the application's health and performance.

Management: Allows for remote management and configuration of the application.

Auditing: Provides auditing capabilities to track changes and access.

How to Use Actuator:

1. Add the Actuator dependency to your pom.xml or build.gradle.
2. Enable the Actuator endpoints in the application.properties or application.yml.
3. Access the endpoints via HTTP (e.g., /actuator/health, /actuator/info).

19. Authentication and Authorization in Spring Security

Spring Security provides comprehensive security services for Java EEbased enterprise software applications.

Authentication: The process of verifying the identity of a user.

Authorization: The process of determining whether a user has permission to access a resource.

Implementation:

1. Add Spring Security dependency.
2. Configure security settings in a configuration class annotated with @EnableWebSecurity.
3. Define authentication and authorization rules using methods like configure(HttpSecurity http).
4. Use annotations like @PreAuthorize and @Secured to control access to methods.

20. How do you work with a Git repository & strategy?

Working with a Git repository involves several common commands and strategies:

Clone: git clone <repository-url> to copy a repository to your local machine

21. What do I look for in a code while doing a PR review?
When reviewing a pull request (PR), look for the following:

Code Quality: Ensure the code is clean, well structured, and follows coding standards.
Functionality: Verify that the code implements the desired functionality correctly.
Test Coverage: Check that there are adequate tests covering the new code.
Performance: Ensure the code does not introduce performance issues.
Security: Look for potential security vulnerabilities.

Documentation: Verify that the code is well documented.
Consistency: Ensure the code is consistent with the existing codebase.

22. Override Annotation situation basis

The @Override annotation in Java is used to indicate that a method is intended to override a method in a superclass. It helps catch errors at compile time if the method signature does not match the overridden method.

Situations to Use @Override:
When you want to ensure that a method in a subclass correctly overrides a method in a superclass.
When implementing methods from an interface.
When you want to make the code more readable and maintainable by clearly indicating the intention to override a method.

Example:

The @Override annotation in Java is used to indicate that a method is intended to override a method in a superclass. It helps catch errors at compile time if the method signature does not match the overridden method.

Situations to Use @Override:

When you want to ensure that a method in a subclass correctly overrides a method in a superclass.
When implementing methods from an interface.
When you want to make the code more readable and maintainable by clearly indicating the intention to override a method.

Example:

```
class Animal {
  void makeSound() {
    System.out.println("Animal sound");
  }
}

class Cat extends Animal {
  @Override
  void makeSound() {
    System.out.println("Meow");
  }
}
```

In this example, the @Override annotation ensures that the makeSound method in the Cat class correctly overrides the makeSound method in the Animal class.

23. Design patterns you have worked on?
24. Unit testing?
25. Tell me your achievements that you are proud of from current or last project?
26. Why should we hire you?