1. how to avoid circular dependency in spring boot
2. **Refactor code**:

The most effective way is to redesign the code to eliminate the circular dependency. This might involve merging classes, extracting shared logic into a new component, or rethinking the relationships between beans.

1. Use @Lazy annotation:

Spring's @Lazy annotation can be used to delay the initialization of one of the beans involved in the circular dependency. This breaks the cycle by allowing Spring to create the beans without immediately resolving the dependency.

@Component

public class BeanA {

private final BeanB beanB;

@Autowired

public BeanA(@Lazy BeanB beanB) {

this.beanB = beanB;

}

}

@Component

public class BeanB {

private final BeanA beanA;

@Autowired

public BeanB(BeanA beanA) {

this.beanA = beanA;

}

}

* **Use setter injection**: Instead of constructor injection, setter injection allows Spring to create the beans first and then set the dependencies.

@Component

public class BeanA {

private BeanB beanB;

@Autowired

public void setBeanB(BeanB beanB) {

this.beanB = beanB;

}

}

@Component

public class BeanB {

private BeanA beanA;

@Autowired

public void setBeanA(BeanA beanA) {

this.beanA = beanA;

}

}

2. multipartfile parameter to upload file in spring boot

To handle file uploads in Spring Boot, the MultipartFile parameter is used within a controller method. This parameter represents the uploaded file and provides methods to access its content, name, and other metadata.

To use MultipartFile, add it as a parameter annotated with @RequestParam to a controller method that handles a POST request. The @RequestParam annotation maps the file data from the request to the MultipartFile parameter.

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

@RestController

public class FileUploadController {

private static final String UPLOAD\_DIR = "uploads/";

@PostMapping("/upload")

public String handleFileUpload(@RequestParam("file") MultipartFile file) {

try {

if (file.isEmpty()) {

return "Please select a file to upload.";

}

// Create the upload directory if it doesn't exist

Path uploadPath = Paths.get(UPLOAD\_DIR);

if (!Files.exists(uploadPath)) {

Files.createDirectories(uploadPath);

}

// Save the file to the server

Path filePath = uploadPath.resolve(file.getOriginalFilename());

Files.copy(file.getInputStream(), filePath);

return "File uploaded successfully: " + file.getOriginalFilename();

} catch (IOException e) {

return "Failed to upload file: " + e.getMessage();

}

}

}  
  
  
To enable file uploading, ensure that the spring-boot-starter-web dependency is included in the project. Additionally, Spring Boot automatically configures a MultipartResolver, which handles the parsing of multipart requests.

It's also possible to configure file upload limits in application.properties or application.yml:

Code

spring.servlet.multipart.max-file-size=10MB  
spring.servlet.multipart.max-request-size=10MB

3. @preauthorize @postauthorize java code example  
  
import org.springframework.security.access.prepost.PreAuthorize;

import org.springframework.security.access.prepost.PostAuthorize;

import org.springframework.stereotype.Service;

@Service

public class MyService {

@PreAuthorize("hasRole('ADMIN')")

public String adminOnlyMethod() {

return "This method is only accessible to users with the ADMIN role.";

}

@PreAuthorize("hasAnyRole('ADMIN', 'USER')")

public String adminOrUserMethod() {

return "This method is accessible to users with either ADMIN or USER role.";

}

@PreAuthorize("#username == authentication.principal.username")

public String myData(String username) {

return "This is " + username + "'s data.";

}

@PostAuthorize("returnObject == authentication.principal.username")

public String getLoggedUsername() {

return "current logged username";

}

@PostAuthorize("returnObject.owner == authentication.name")

public Resource getResource(Long id) {

return new Resource("example", "user1");

}

}

class Resource {

private String name;

private String owner;

public Resource(String name, String owner) {

this.name = name;

this.owner = owner;

}

public String getName() {

return name;

}

public String getOwner() {

return owner;

}

}  
  
To enable method-level security, add @EnableMethodSecurity annotation to a configuration class:

Java

import org.springframework.context.annotation.Configuration;  
 import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;  
   
 @Configuration  
 @EnableMethodSecurity  
 public class MethodSecurityConfig {

………..  
 }

1. what is sprin apache kafka  
    similar to rabbitmq which we are using in our project – it provides a high-level abstraction for sending and receiving messages  
   but rabbitmq is only post but kafka posts and listens  
   its mostly used in recenrt projects – see more in google
2. @transactional – db   
     
   **Invocations Within the Same Class**
3. **@Transactional** is rarely covered by enough tests, and this means that some problems are not visible at first glance. As a result, you can come across the following code:
4. Java
5. 1
6. public void registerAccount(Account acc) {
7. 2
8. createAccount(acc);
9. 3
10. ​
11. 4
12. notificationSrvc.sendVerificationEmail(acc);
13. 5
14. }
15. 6
16. ​
17. 7
18. @Transactional
19. 8
20. public void createAccount(Account acc) {
21. 9
22. accRepo.save(acc);
23. 10
24. teamRepo.createPersonalTeam(acc);
25. 11
26. }
27. In this case, when calling registerAccount(), saving the user and creating a team will not be performed in a common transaction. **@Transactional** is powered by [Aspect-Oriented Programming](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#aop). Therefore, processing occurs when a bean is called from another bean. In the example above, the method is called from the same class so that no proxies can be applied. The same is true for other annotations such as **@Cacheable**.

The problem can be solved in three basic ways:

Self-inject

Create another layer of abstraction – we have done this in iris project

Use **TransactionTemplate** in the registerAccount() method by wrapping createAccount()call

5. How arraylist internally works?

An ArrayList in Java is a dynamic, resizable array implementation of the List interface. It uses an internal array to store elements and automatically increases its capacity when needed.

When an ArrayList is created, it is initialized with a default capacity of 10 or a user-specified initial capacity. As elements are added using the add() method, the ArrayList checks if there is enough space in the internal array. If the array is full, a new array with a larger capacity (typically 1.5 times the original size) is created. The elements from the old array are then copied to the new array, and the new element is added.

The get(index) method provides constant-time access to elements because it directly accesses the element at the specified index in the internal array. The remove(index) method, however, requires shifting subsequent elements to fill the gap created by the removed element, resulting in a linear time complexity.

ArrayList is not thread-safe, meaning that concurrent access from multiple threads can lead to race conditions. If thread safety is required, it is recommended to use Collections.synchronizedList() or CopyOnWriteArrayList.

6. **@RequestParam vs @QueryParam vs @PathParam vs @PathVariable in REST**

**@PathParam & @QueryParam are NOT Spring Annotations. They are JAX-RS annotation. Whereas only @RequestParam & @PathVariable are SPRING Annotations!!**

whats jax-rs -> JAX-RS, which stands for Java API for RESTful Web Services, is a Java API specification that simplifies the development of RESTful web services. It provides a set of interfaces and annotations to make it easier to create and consume REST applications. Essentially, JAX-RS is a framework that enables Java developers to build RESTful APIs in a more intuitive and efficient way.   
  
[Difference between JAX-RS and Spring Rest](https://stackoverflow.com/questions/42944777/difference-between-jax-rs-and-spring-rest)

**235**

JAX-RS

JAX-RS is a [specification](http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-fr-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf) for implementing REST web services in Java, currently defined by the [JSR-370](https://jcp.org/en/jsr/detail?id=370). It is part of the [Java EE technologies](https://stackoverflow.com/a/37083274/1426227), currently defined by the [JSR 366](https://www.jcp.org/en/jsr/detail?id=366).

[Jersey](https://jersey.java.net/) (shipped with GlassFish and Payara) is the JAX-RS reference implementation, however there are other implementations such as [RESTEasy](http://resteasy.jboss.org/) (shipped with JBoss EAP and WildFly) and [Apache CXF](https://cxf.apache.org/) (shipped with TomEE and WebSphere).

Spring Framework

The [Spring Framework](http://projects.spring.io/spring-framework/) is a [full framework](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html) that allows you to create Java enterprise applications. The REST capabilities are provided by the [Spring MVC](https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html) module (same module that provides *model-view-controller* capabilities). It is not a JAX-RS implementation and can be seen as a Spring alternative to the JAX-RS standard.

The Spring ecosystem also provides a [wide range of projects](https://spring.io/projects) for creating enterprise applications, covering persistence, security, integration with social networks, batch processing, etc.

Examples

Consider the following resource controller using the JAX-RS API:

@Path("/greetings")

public class JaxRsController {

@GET

@Path("/{name}")

@Produces(MediaType.TEXT\_PLAIN)

public Response greeting(@PathParam("name") String name) {

String greeting = "Hello " + name;

return Response.ok(greeting).build();

}

}

The equivalent implementation using the Spring MVC API would be:

@RestController

@RequestMapping("/greetings")

public class SpringRestController {

@RequestMapping(method = RequestMethod.GET,

value = "/{name}",

produces = MediaType.TEXT\_PLAIN\_VALUE)

public ResponseEntity<?> greeting(@PathVariable String name) {

String greeting = "Hello " + name;

return new ResponseEntity<>(greeting, HttpStatus.OK);

}

}

7. 1) merge 2 sorted arrays into 1 a={1 3 5} b={2 4 6} o/p: {1 2 3 4 5 6} using java 8  
int[] a = {1, 3, 5};  
 int[] b = {2, 4, 6};  
  
 int[] merged = IntStream.concat(Arrays.stream(a), Arrays.stream(b))  
 .sorted()  
 .toArray();  
  
 System.out.println(Arrays.toString(merged)); *// Output: [1, 2, 3, 4, 5, 6]*

8. sublist in list   
  
list<>….. l = [1,2,3,4,5]

L1= l.subList(1,4) //output -> 2,3,4

9. In Java, **"fail-fast"** and **"fail-safe"** refer to different behaviors of iterators when a collection is modified during iteration. Here's a simple explanation and code examples for both:

**✅ Fail-Fast Iterator**

* **Throws ConcurrentModificationException** if the collection is modified structurally while iterating (except through the iterator's own methods).
* Common in classes like ArrayList, HashMap, etc.

**Example:**

**import java.util.ArrayList;**

**import java.util.Iterator;**

**public class FailFastExample {**

**public static void main(String[] args) {**

**ArrayList list = new ArrayList<>();**

**list.add("Apple");**

**list.add("Banana");**

**list.add("Cherry");**

**Iterator iterator = list.iterator();**

**while (iterator.hasNext()) {**

**String fruit = iterator.next();**

**if (fruit.equals("Banana")) {**

**list.remove(fruit); // Structural modification during iteration**

**}**

**}**

**}**

**}**

**Output:**

Exception in thread "main" java.util.ConcurrentModificationException

**✅ Fail-Safe Iterator**

* **Does not throw ConcurrentModificationException**.
* Works on a **clone** of the collection, so changes to the original collection don’t affect the iteration.
* Found in classes like ConcurrentHashMap, CopyOnWriteArrayList.

**Example:** **import java.util.concurrent.CopyOnWriteArrayList;**

**public class FailSafeExample {**

**public static void main(String[] args) {**

**CopyOnWriteArrayList list = new CopyOnWriteArrayList<>();**

**list.add("Apple");**

**list.add("Banana");**

**list.add("Cherry");**

**for (String fruit : list) {**

**if (fruit.equals("Banana")) {**

**list.remove(fruit); // Safe modification**

**}**

**}**

**System.out.println("Final list: " + list);**

**}**

**}**

**Output:**

Final list: [Apple, Cherry]

10. a simple spring boot app (just for quick recap on strcuture)  
  
model :  
package com.example.demo.model;

import jakarta.persistence.\*;

@Entity

public class User {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String name;

    private String email;

    // Getters and Setters

}

user repository :   
package com.example.demo.repository;

import com.example.demo.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<user, long> {  
// we can add custom ones here by calling dao methods or by writing queries

}

service layer :   
package com.example.demo.service;

import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class UserService {

    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {

        this.userRepository = userRepository;

    }

    public List getAllUsers() {

        return userRepository.findAll();

    }

    User createUser(User user) {

        return userRepository.save(user);

    }

}

Controller layer ;   
package com.example.demo.controller;

import com.example.demo.model.User;

import com.example.demo.service.UserService;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/users")

public class UserController {

    private final UserService userService;

    publicService userService) {

        this.userService = userService;

    }

    @GetMapping

    public List getAllUsers() {

        return userService.getAllUsers();

    }

    @PostMapping

    public User createUser(@RequestBody User user) {

        return userService.createUser(user);

    }

}

few application properties to define port or db config ..etc  
and main app   
  
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

    public static void main(String[] args) {

        SpringApplication.run(D.class, args);

    }

}

11. we are using jdbc in our iris – so hibernate or jpa version of one of that method is as below  
  
iris one - using jdbc -  
public void updateProcessingStatus(String orderId, ProcessingStatus processingStatus) {

    String updateQuery = UPDATE + TABLE\_NAME + " d " +

            JOIN + TypedOrderDao.TABLE\_NAME + " o ON d.typed\_order\_id = o.id " +

            "SET " + PROCESSING\_STATUS + "=?" + " " +

            "WHERE o." + TypedOrderDao.ORDER\_ID + "=?";

    getWriteJdbcTemplate().update(updateQuery, parameters(processingStatus.getDatabaseValue(), orderId));

}

to jpa :   
  
@Repository

public interface DeliveryRepository extends JpaRepository<delivery, long> {</delivery, long>

    @Modifying

    @Transactional

    @Query("UPDATE Delivery d SET d.processingStatus = :status WHERE d.typedOrder.orderId = :orderId")

    void updateProcessingStatusByOrderId(@Param("orderId") String orderId, @Param("status") String status);

}

12. Amazon EC2 is a cloud service, it can be considered a tool because it provides the capability to launch and manage virtual servers (instances) within the Amazon Web Services (AWS) cloud. This allows users to run applications, host websites, and perform various computing tasks without the need for physical hardware.

Here's a more detailed explanation:

* **EC2 as a Service:**

EC2 is a web-based service offered by AWS, enabling users to access computing resources on demand.

* **Virtual Servers:**

EC2 allows users to launch and manage virtual servers (instances) within the AWS cloud.

* **Tool for Computing:**

These virtual servers (instances) can then be used as tools to host applications, websites, and other computing tasks.

* **Scalability and Flexibility:**

EC2 offers the flexibility to scale computing resources up or down based on needs, making it a useful tool for various workloads.

* **Cost-Effective:**

By using EC2, users can avoid the upfront costs of purchasing and maintaining physical hardware, making it a cost-effective tool for computing.

13. searched this in copilot  
spring boot communication between microservices example

give me a small boot app which had both sync communicaton and asycn of kafka along with eureka server

got this :   
   
spring-microservices-example/

├── eureka-server/ # Service registry

├── service-a/ # REST + Kafka producer, Eureka client

├── service-b/ # REST + Kafka consumer, Eureka client

└── common/ # Shared DTOs and Kafka config

**1. Eureka Server**

**Dependencies**: Spring Boot Web, Spring Cloud Netflix Eureka Server

**EurekaServerApplication.java**

@SpringBootApplication

@EnableEurekaServer

public class EurekaServerApplication {

    public static void main(String[] args) {

        SpringApplication.run(EurekaServerApplication.class, args);

    }

}

Appli.prop  
server:

  port: 8761

eureka:

  client:

    register-with-eureka: false

    fetch-registry: false

**2. Service A & B as Eureka Clients**

**Add to both service-a and service-b**:

**Dependencies**: Add spring-cloud-starter-netflix-eureka-client

**application.yml (Service A)**

server:

  port: 8081

spring:

  application:

    name: service-a

eureka:

  client:

    service-url:

      defaultZone: http://localhost:8761/eureka

server:

  port: 8082

spring:

  application:

    name: service-b

eureka:

  client:

    service-url:

      defaultZone: http://localhost:8761/eureka

**Enable Eureka Client in both apps**@SpringBootApplication

@EnableEurekaClient

public class ServiceAApplication {

    public static void main(String[] args) {

        SpringApplication.run(ServiceAApplication.class, args);

    }

}

3. **Update REST Call in Service A**

Replace hardcoded URL with service discovery:

@Autowired

private LoadBalancerClient loadBalancer;

@PostMapping("/send")

public ResponseEntity sendMessage(@RequestBody MessageDTO message) {

    ServiceInstance instance = loadBalancer.choose("service-b");

    String url = instance.getUri().toString() + "/api/b/receive";

    restTemplate.postForObject(url, message, String.class);

    kafkaTemplate.send("demo-topic", message);

    return ResponseEntity.ok("Message sent via REST and Kafka");

}

**Summary**

* **Eureka Server** runs on port 8761
* **Service A** and **Service B** register with Eureka
* **Service A** uses LoadBalancerClient to discover service-b dynamically

--------------------------------from repo--------------------------------------------

1. **How to create a thread-safe singleton class in Java using double-checked locking?**

public class Singleton {

    private static volatile Singleton instance;

    private Singleton() {

        // private constructor

    }

    public static Singleton getInstance() {

        if (instance == null) {

            synchronized (Singleton.class) {

                if (instance == null) {

                    instance = new Singleton();

                }

            }

        }

        return instance;

    }

}

2. **Why is multiple inheritance not supported in Java?**

* + Java does not support multiple inheritance to avoid complexity and simplify the design. Multiple inheritance can lead to the "diamond problem," where a class inherits from two classes that have a common base class, causing ambiguity in method resolution.

3. **Can you override a static method in Java?**

* No, static methods cannot be overridden because they belong to the class, not instances of the class. However, they can be hidden by defining a static method with the same signature in a subclass.

Example below

class Animal {

public static void makeSound() {

System.out.println("Generic animal sound");

}

}

class Dog extends Animal {

public static void makeSound() {

System.out.println("Woof!");

}

}

public class Main {

public static void main(String[] args) {

Animal animal = new Animal();

Dog dog = new Dog();

Animal animalDog = new Dog();

animal.makeSound(); // Output: Generic animal sound

dog.makeSound(); // Output: Woof!

animalDog.makeSound(); // Output: Generic animal sound

}

}

In this example, Dog hides the makeSound method of Animal. When animalDog.makeSound() is called, the makeSound method of Animal is executed because animalDog is declared as an Animal, even though it refers to a Dog object. This demonstrates method hiding, not method overriding.

4. **Why does Java not support operator overloading?**

* Java does not support operator overloading to maintain code simplicity and readability. Operator overloading can make code harder to understand and maintain.

Instead of operator overloading, Java relies on method overloading to achieve similar functionality in a more controlled and explicit manner. Method overloading allows multiple methods with the same name but different parameters within the same class

class Calculator {  
 *// Method overloading for addition*  
 int add(int a, int b) {  
 return a + b;  
 }  
  
 int add(int a, int b, int c) {  
 return a + b + c;  
 }  
  
 double add(double a, double b) {  
 return a + b;  
 }  
  
 public static void main(String[] args) {  
 Calculator calc = new Calculator();  
 System.out.println(calc.add(2, 3)); *// Output: 5*  
 System.out.println(calc.add(2, 3, 4)); *// Output: 9*  
 System.out.println(calc.add(2.5, 3.5)); *// Output: 6.0*  
 }  
}

5 . **Why is String immutable in Java?**

* + Strings are immutable in Java to ensure security, thread safety, and performance. Immutable objects are inherently thread-safe, and they can be shared freely without synchronization.
  + We can make them create a new string obj using strungbuilder or buffer now these can be mutable

6. **Is String thread-safe in Java?**

* + Yes, since String is immutable, it is thread-safe. Multiple threads can safely share and use String instances without any synchronization.

**7. Write a program to reverse a string:**

public class ReverseString {

    public static void main(String[] args) {

        String original = "Hello, World!";

        String reversed = new StringBuilder(original).reverse().toString();

        System.out.println("Original: " + original);

        System.out.println("Reversed: " + reversed);

    }

}

8// skipped not needed

**9.What is String constant pool or string pool in Java?**

* + The String constant pool is a special memory region where Java stores string literals. When a string literal is created, Java checks the pool first; if the literal already exists, it returns the reference to the existing string, otherwise, it creates a new string in the pool.

Eg:

public class StringPoolExample {

    public static void main(String[] args) {

        // Creating string literals

        String str1 = "Hello";

        String str2 = "Hello";

        // Creating strings using the new keyword

        String str3 = new String("Hello");

        String str4 = new String("Hello");

        // Comparing string literals (references are the same)

        System.out.println("str1 == str2: " + (str1 == str2)); // true

        // Comparing strings created with new keyword (references are different)

        System.out.println("str3 == str4: " + (str3 == str4)); // false

        // Comparing string literal with string created using new keyword

        System.out.println("str1 == str3: " + (str1 == str3)); // false

        // Using intern() method to add str3 to the string pool

        String str5 = str3.intern();

        // Comparing interned string with string literal

        System.out.println("str1 == str5: " + (str1 == str5)); // true

    }

}

**10.Write a program to reverse a string:**

public class ReverseString {

    public static void main(String[] args) {

        String original = "Hello, World!";

        String reversed = new StringBuilder(original).reverse().toString();

        System.out.println("Original: " + original);

        System.out.println("Reversed: " + reversed);

    }

}

11. **List of different strings, how to concatenate all strings and create a new string?**

List strings = Arrays.asList("Hello", " ", "World", "!");

String concatenated = String.join("", strings);

System.out.println(concatenated); // Output: Hello World!

12. **What is the difference between StringBuilder and StringBuffer in Java?**

* + StringBuilder is not synchronized and is faster, while StringBuffer is synchronized and thread-safe.

13. **What happens if your Serializable class contains a member which is not serializable? How do you fix it?**

* + A NotSerializableException will be thrown. To fix it, mark the non-serializable member as transient.

14. **How to create an immutable class in Java?**

**Use final class and don’t add setters**

public **final** class ImmutableClass {

    private final int value;

    public ImmutableClass(int value) {

        this.value = value;

    }

    public int getValue() {

        return value;

    }

}

15. **What is the use of the intern() method in Java?**

* + The intern() method returns a canonical representation for the string object, ensuring that all identical string literals share the same reference.
  + See question 9th code example

1. 16. **How do you break a singleton pattern?**
   * Using reflection, serialization, or cloning.
2. 17. **How do you prevent a singleton pattern from breaking?**
   * Use enum for singleton, implement readResolve method, and throw exceptions in clone and reflection.

18 .whats metaspace which is introduced in java8

Here, the heap area is one of the most important memory areas of JVM. Here, all the [java objects](https://www.geeksforgeeks.org/classes-objects-java/) are stored. The heap is created when the JVM starts. The heap is generally divided into two parts. That is: 

1. **Young Generation(Nursery):** All the new objects are allocated in this memory. Whenever this memory gets filled, the [garbage collection](https://www.geeksforgeeks.org/garbage-collection-java/) is performed. This is called as *Minor Garbage Collection*.
2. **Old Generation:** All the long lived objects which have survived many rounds of minor garbage collection is stored in this area. Whenever this memory gets filled, the garbage collection is performed. This is called as *Major Garbage Collection*.
3. Apart from the heap memory, JVM also contains another type of memory which is called as Permanent Generation or “PermGen”.
4. **PermGen Memory:** This is a special space in java heap which is separated from the main memory where all the [static content is stored](https://www.geeksforgeeks.org/understanding-storage-of-static-methods-and-static-variables-in-java/) in this section. Apart from that, this memory also stores the application metadata required by the JVM. Metadata is a data which is used to describe the data. Here, garbage collection also happens like any other part of the memory. String pool was also part of this memory before Java 7. Method Area is a part of space in the PermGen and it is used to store the class structure and the code for methods and constructors. The biggest disadvantage of PermGen is that it contains a limited size which leads to an [OutOfMemoryError](https://www.geeksforgeeks.org/understanding-outofmemoryerror-exception-java/). The default size of PermGen memory is 64 MB on 32-bit JVM and 82 MB on the 64-bit version. Due to this, JVM had to change the size of this memory by frequently performing Garbage collection which is a costly operation. Java also allows to manually change the size of the PermGen memory. However, the PermGen space cannot be made to auto increase. So, it is difficult to tune it. And also, the garbage collector is not efficient enough to clean the memory.
5. Due to the above problems, PermGen has been completely removed in Java 8. In the place of PermGen, a new feature called Meta Space has been introduced. MetaSpace grows automatically by default. Here, the garbage collection is automatically triggered when the class metadata usage reaches its maximum metaspace size.
6. **Can you tell me 3 terminal operators used in streams?**
   * forEach(), collect(), reduce().

**import** **java.util.\***;

1. **class** **GFG** {
2. **public** **static** void main(String[] args) {
3. *// Creating a list of Strings*
4. List<String> words = Arrays.asList("GFG", "Geeks", "for", "GeeksQuiz", "GeeksforGeeks");
5. *// Using reduce to find the longest string in the list*
6. Optional<String> longestString = words.stream()
7. .reduce((word1, word2) -> word1.length() > word2.length() ? word1 : word2);
8. *// Displaying the longest String*
9. longestString.ifPresent(System.out::println);
10. }

import java.util.Arrays;  
import java.util.List;  
  
public class ReduceStringExample {  
 public static void main(String[] args) {  
 List<String> words = Arrays.asList("Hello", "World", "!");  
  
 String sentence = words.stream()  
 .reduce("", (a, b) -> a + " " + b);  
  
 System.out.println("Sentence: " + sentence.trim()); *// Output: Sentence: Hello World !*  
 }  
}

import java.util.Arrays;  
import java.util.List;  
  
public class ReduceExample {  
 public static void main(String[] args) {  
 List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);  
  
 int sum = numbers.stream()  
 .reduce(0, (a, b) -> a + b);  
  
 System.out.println("Sum: " + sum); *// Output: Sum: 15*  
 }  
}

1. 20. **Can you tell me how HashMap internally works?**
   * HashMap uses an array of buckets, and each bucket is a linked list or tree. It uses the hash code of the key to determine the bucket and handles collisions using chaining or tree structure.
2. 21. **How does the load factor work in HashMap?**
   * The load factor determines when to resize the HashMap. When the number of entries exceeds the product of the load factor and the current capacity, the HashMap is resized.
3. 22. **What is a Hashtable?**
   * A synchronized map implementation that does not allow null keys or values.
4. 23. **How does a blocking queue work?**
   * A blocking queue supports operations that wait for the queue to become non-empty when retrieving an element and wait for space to become available when adding an element.

// in my diary notes – we have some more info

1. 24. **Difference between sleep() and wait()?**
   * sleep() is a method of Thread class that pauses the current thread for a specified period. wait() is a method of Object class that causes the current thread to wait until another thread invokes notify() or notifyAll() on the same object.
2. 25. **In try-catch-finally, if we return a value, which block has priority?**
   * The finally block will execute after the try or catch block, even if there is a return statement. The value returned will be from the finally block if it contains a return statement.
3. 26. **What is Inversion of Control?**
   * Inversion of Control (IoC) is a design principle where the control of object creation and management is transferred from the application code to a container or framework, such as Spring.
4. 27. **How does garbage collection work in Java?**
   * The garbage collector identifies and discards objects that are no longer reachable from any references in the program. It uses algorithms like mark-and-sweep, generational garbage collection, and reference counting.

28. **Which scenario have you used the Optional class and how is it helpful?**

* The Optional class is used to represent a value that may or may not be present. It helps avoid NullPointerException and provides methods to handle the presence or absence of a value.

29, **What is Docker, Docker image, and Docker container?**

* + **Docker**: A platform for developing, shipping, and running applications inside containers.
  + **Docker Image**: A lightweight, standalone, and executable software package that includes everything needed to run a piece of software.
  + **Docker Container**: A runtime instance of a Docker image.

1. **Can you tell me a real-world example of encapsulation?**
   * A real-world example of encapsulation is a class representing a bank account. The class hides the account balance and provides methods to deposit and withdraw money, ensuring that the balance cannot be directly accessed or modified from outside the class.

34. Reason for overriding equals and hashCode methods?  
 Overriding the equals() and hashCode() methods in Java is crucial when you want to define your own comparison logic for objects, particularly when using them in collections like HashSet or HashMap. It ensures that objects are compared based on their content (logical equality) rather than their memory address (reference equality), and it's essential for the correct behavior of hash-based collections.

35. **What is the use of CopyOnWriteArrayList?**

* + CopyOnWriteArrayList is a thread-safe variant of ArrayList where all mutative operations (add, set, remove) are implemented by making a fresh copy of the underlying array. It is useful in scenarios where reads are more frequent than writes.

36. **Can you tell me the difference between truncate and delete?**

* TRUNCATE removes all rows from a table without logging individual row deletions, making it faster. DELETE removes rows one at a time and logs each deletion, allowing for rollback

37 .I **am expecting an exception in one of my code, how can I write a test case for that?**

* Use JUnit's @Test(expected = Exception.class) annotation to write a test case that expects an exception.

38. create a table and basic select

CREATE TABLE emp (  
 emp\_no INT PRIMARY KEY,  
 name VARCHAR(50),  
 sal DECIMAL(10,2),  
 age INT  
);  
  
INSERT INTO emp (emp\_no, name, sal, age) VALUES  
(1, 'Aarav', 50000.00, 25),  
(2, 'Aditi', 60000.50, 30),  
(3, 'Aarav', 75000.75, 35),  
(4, 'Anjali', 45000.25, 28),  
(5, 'Chetan', 80000.00, 32),  
(6, 'Divya', 65000.00, 27),  
(7, 'Gaurav', 55000.50, 29),  
(8, 'Divya', 72000.75, 31),  
(9, 'Gaurav', 48000.25, 26),  
(10, 'Divya', 83000.00, 33);  
SELECT \* from emp;

39. **SQL queries with group by?**

* GROUP BY is used to group rows that have the same values in specified columns and allows aggregate functions to be applied to each group.

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

// so in table if we have multiple rows wit Aus then we will get count , Aus

40. **What are triggers in the database?**

* Triggers are database objects that automatically execute a specified action in response to certain events on a table, such as INSERT, UPDATE, or DELET

41. **What is the Callable interface in threads?**

* The Callable interface is similar to Runnable but can return a result and throw a checked exception
* Eg: Callable<Integer> task = () -> {
* return 123;
* };

42. **How does auto-configuration work in Spring Boot?**

* + Auto-configuration in Spring Boot works by scanning the classpath for certain classes and beans, and automatically configuring them based on the dependencies and properties present.

43. **Do you know why we use @Qualifier annotation?**

* + @Qualifier is used to resolve ambiguity when multiple beans of the same type are present. It specifies which bean should be injected.

Eg: interface MessageService {  
 String sendMessage();  
}  
  
@Component("emailService")  
class EmailService implements MessageService {  
 @Override  
 public String sendMessage() {  
 return "Sending email";  
 }  
}  
  
@Component("smsService")  
class SMSService implements MessageService {  
 @Override  
 public String sendMessage() {  
 return "Sending SMS";  
 }  
}  
  
@Component  
class NotificationService {  
 private final MessageService messageService;  
  
 @Autowired  
 public NotificationService(@Qualifier("smsService") MessageService messageService) {  
 this.messageService = messageService;  
 }  
  
 public void sendNotification() {  
 System.out.println(messageService.sendMessage());  
 }  
}

44. whats bean in spring?

In the Spring Framework, a bean is an object that is managed by the Spring IoC (Inversion of Control) container. It is an instance of a class that is configured and controlled by Spring.

1. 45. **Which scenarios will we use @Component vs @Bean in Spring Boot?**
   * Use @Component for auto-detectable components during classpath scanning. Use @Bean for defining beans in configuration classes explicitly.
2. 46. **What does @SpringBootApplication do?**
   * @SpringBootApplication is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. It marks the main class of a Spring Boot application.
3. 47. **What are scopes of Spring bean?**
   * Common scopes include singleton, prototype, request, session, and globalSession.
4. 48. **@Controller vs @RestController in Spring?**
   * @Controller is used for defining web controllers that return views. @RestController is a specialized version of @Controller that returns JSON or XML responses directly.
5. 49. **Difference between JDK, JRE, and JVM?** - **JDK (Java Development Kit)**: A software development kit used to develop Java applications. It includes the JRE and development tools. - **JRE (Java Runtime Environment)**: Provides the libraries, Java Virtual Machine (JVM), and other components to run Java applications. - **JVM (Java Virtual Machine)**: An abstract machine that enables a computer to run Java programs by converting bytecode into machine code.