Spring Boot & Microservices

Explain the flow of a REST API call from frontend to backend and back.

What is Spring Security?

What is JWT, and how is it used in microservices authentication?

How would you set up a discovery server like Eureka?

What is the default server in Spring Boot, and how to use another server?

What are Spring Boot profiles, and how do you use them?

How is bean injection handled in Spring?

How to inject a specific bean when multiple beans of the same type exist?

==========
🛠️ Spring Boot & Microservices

What is Spring Boot, and how does it differ from the traditional Spring framework?

Explain the concept of Dependency Injection in Spring.

What are Spring Boot Profiles, and how are they used?

Explain the role of @Component, @Service, and @Repository annotations in Spring.

What is Spring Security, and how is it implemented in a Spring Boot application?

Explain the concept of Spring Boot Actuator.

What is the role of @SpringBootApplication annotation?

How do you handle exception management in Spring Boot?

What is a Circuit Breaker, and how is it implemented in Spring Boot?

Explain the concept of Spring Boot Auto-Configuration.
==========
Spring Boot & Microservices

Spring Boot Overview: What is Spring Boot, and how does it differ from the traditional Spring framework?

Dependency Injection: Explain the concept of Dependency Injection in Spring.

Spring Boot Profiles: What are Spring Boot Profiles, and how are they used?

Annotations in Spring: Explain the role of @Component, @Service, and @Repository annotations in Spring.

Spring Security: What is Spring Security, and how is it implemented in a Spring Boot application?

Spring Boot Actuator: Explain the concept of Spring Boot Actuator.

Auto-Configuration: What is the concept of Spring Boot Auto-Configuration?

Exception Handling: How do you handle exception management in Spring Boot?

Circuit Breaker: How is a Circuit Breaker implemented in Spring Boot?

Microservices Authentication: How do microservices authenticate each other in a Spring Boot application?
==========
Spring Boot Profiles: What are Spring Boot profiles, and how are they used to externalize configurations?

@Primary Annotation: What is the use of the @Primary annotation in Spring?

Discovery Server: How do you set up a discovery server like Eureka in a Spring Boot application?

Microservices Authentication: How do microservices authenticate each other in a Spring Boot environment?

Default Server: Which server comes by default with Spring Boot, and how can you configure a different server?
==========
Spring Boot Overview: What is Spring Boot, and what are its advantages?

Spring Boot Actuator: What is Spring Boot Actuator, and what are its features?

Exception Handling in Spring Boot: How do you handle exceptions in Spring Boot?
==========
Spring Boot Profiles: What are Spring Boot profiles, and how are they used to externalize configurations?

@Primary Annotation: What is the use of the @Primary annotation in Spring?

Discovery Server: How do you set up a discovery server like Eureka in a Spring Boot application?

Microservices Authentication: How do microservices authenticate each other in a Spring Boot environment?

Default Server: Which server comes by default with Spring Boot, and how can you configure a different server?
==========
What is dependency injection in Spring?

Explain the difference between @Component, @Service, and @Repository.

How does Spring Boot simplify application development?

What are Spring profiles, and how are they used?

Explain the concept of microservices and how Spring Boot supports them.
==========
🔷 Spring Boot & Microservices

Spring Boot Basics: What is Spring Boot?

Profiles: What are profiles in Spring Boot and how are they used?

Bean Injection: How does bean injection work in Spring?

@Primary Annotation: What is the use of the @Primary annotation in Spring?

Microservices Architecture: Explain the microservices architecture and how

Spring Boot facilitates it.

JWT Authentication: How is JWT used for authentication in microservices?

Circuit Breaker: How is a circuit breaker implemented in Spring Boot?

Discovery Server: How would you set up a discovery server like Eureka?

==========

What is Spring Boot and what are its advantages?

What is Spring Boot Actuator and what are its features?

What is the use of the @Primary annotation in Spring Boot?

What is the use of @Profile in Spring Boot?

What is the use of the @Value annotation?

How does @Autowired work in Spring Boot?

What is the difference between @Bean and @Component?

What is the difference between @RequestMapping, @GetMapping, and @PostMapping?

What is the difference between @PathVariable and @RequestParam?

What is the use of @ExceptionHandler in Spring Boot?

==========

What is Spring Boot and what are its advantages?

Explain the concept of Dependency Injection in Spring.

What is the use of @Autowired annotation in Spring?

What is the difference between @Component, @Service, and @Repository annotations?

What is Spring Boot Actuator and what are its features?

Explain the concept of Spring Boot profiles.

What is the use of @Primary annotation in Spring?

How does Spring Boot handle exception handling?

What is the difference between @RequestMapping, @GetMapping, and @PostMapping?

What is the role of @PathVariable and @RequestParam in Spring MVC?

==========

Explain Spring Security.

What is JWT and how is it used in authentication?

Explain the flow of a REST API call from frontend to backend and response back to frontend.

What are profiles in Spring Boot and how to use them?

How to set up a discovery server like Eureka?

What is the default port of Spring Boot?

Explain bean injection in Spring.

How to inject a specific bean of the same type?

What is @Primary annotation?

How to externalize a microservice?=====
==========
==========
Infosys Spring Boot + Microservices – 100 Recent Questions
Spring Boot Basics (1–25)

What is Spring Boot and why is it used?

Difference between Spring and Spring Boot.

Explain @SpringBootApplication.

Difference between @Component, @Service, @Repository, and @Controller.

What is @Primary annotation?

Difference between @Component and @Bean.

How to run Spring Boot app on a specific port?

Difference between application.properties and application.yml.

How to handle exceptions globally?

Difference between @Controller and @RestController.

Explain dependency injection in Spring Boot.

How does @Autowired work?

Difference between @RequestParam and @PathVariable.

What are Spring Boot starters?

Explain @Value annotation.

Difference between @RequestMapping and @GetMapping/@PostMapping.

How to secure Spring Boot REST APIs?

How to implement validation with @Valid and @Validated.

Explain Spring Boot actuator and its endpoints.

How to enable CORS in Spring Boot?

Difference between lazy and eager initialization in Spring Boot.

How to implement logging in Spring Boot?

How to manage multiple environments (dev, test, prod)?

Difference between @Transactional propagation types.

How to implement caching in Spring Boot?

Spring Boot Advanced (26–50)

How to connect Spring Boot with multiple databases?

Difference between @RequestBody and @ModelAttribute.

How to use Spring Boot Profiles?

Explain Spring Boot AutoConfiguration.

How to schedule tasks in Spring Boot?

How to implement exception handling in REST API?

Explain the difference between ResponseEntity and returning POJO.

How to configure logging levels in Spring Boot?

Difference between @EnableAutoConfiguration and @SpringBootApplication.

How to implement health check endpoints?

How to implement property placeholder in Spring Boot?

How to externalize configuration?

Explain Spring Boot DevTools.

How to configure embedded Tomcat server?

How to implement file upload in Spring Boot?

How to implement file download in Spring Boot?

How to enable HTTPS in Spring Boot?

Difference between @ControllerAdvice and @RestControllerAdvice.

Explain Spring Boot metrics.

How to create custom starter in Spring Boot?

How to implement batch processing in Spring Boot?

Difference between CommandLineRunner and ApplicationRunner.

How to enable scheduling tasks using @Scheduled?

How to handle validation errors in Spring Boot?

How to implement pagination in Spring Boot REST APIs?

Microservices Basics (51–75)

What is microservices architecture?

Difference between monolithic and microservices.

Explain service discovery.

What is Eureka Server and Eureka Client?

How does load balancing work?

Explain API Gateway.

What is Spring Cloud Config?

Explain circuit breaker pattern.

What is Hystrix?

How to implement inter-service communication?

What is Feign Client?

How to implement fault tolerance?

What is distributed tracing?

How to handle distributed transactions?

Explain JWT authentication in microservices.

Difference between synchronous and asynchronous communication.

How to implement rate limiting?

How to handle retries in microservices?

How to implement caching in microservices?

How to secure microservices using OAuth2?

How to monitor microservices?

What is Zipkin / Sleuth?

Explain microservices logging strategy.

Difference between service registry and discovery server.

How to implement messaging with Kafka / RabbitMQ?

Advanced Microservices (76–100)

How to containerize microservices using Docker?

How to deploy microservices using Kubernetes?

How to implement API versioning?

How to implement backward compatibility in APIs?

How to handle rate limiting using Spring Cloud Gateway?

Explain client-side vs server-side load balancing.

How to implement authentication and authorization in microservices?

How to implement centralized configuration for microservices?

Difference between synchronous REST calls and asynchronous messaging.

How to implement service-to-service communication?

How to implement distributed caching?

Explain sidecar pattern in microservices.

How to handle partial failures?

How to implement canary deployment?

How to implement blue-green deployment?

How to implement health checks for microservices?

How to implement fallback methods with Hystrix / Resilience4j?

How to implement dynamic scaling for microservices?

How to implement API throttling?

How to implement event-driven microservices?

How to implement saga pattern for distributed transactions?

How to implement idempotent operations in microservices?

How to monitor microservices performance?

How to handle versioning of database schemas in microservices?

What are the best practices for building production-ready microservices?

==========Annotation==========
==========
1–25: Core Spring / Spring Boot

@SpringBootApplication – Main class + auto-configuration.

@Configuration – Marks a configuration class.

@Component – Generic Spring bean.

@Service – Marks service layer beans.

@Repository – Marks DAO layer beans.

@Controller – Marks MVC controller.

@RestController – Combines @Controller + @ResponseBody.

@Bean – Declares a bean.

@Autowired – Dependency injection.

@Qualifier – Resolves multiple bean ambiguity.

@Primary – Marks primary bean among multiple candidates.

@Value – Injects property values.

@PropertySource – Load external properties.

@DependsOn – Specifies bean initialization order.

@Lazy – Lazy initialization.

@Scope – Defines bean scope (singleton, prototype).

@PostConstruct – Method after bean creation.

@PreDestroy – Method before bean destruction.

@Import – Import configuration classes.

@Conditional – Conditional bean creation.

@Profile – Load bean based on active profile.

@Order – Defines bean loading order.

@EnableAutoConfiguration – Enable Spring Boot auto-configuration.

@EnableScheduling – Enable scheduling.

@EnableAspectJAutoProxy – Enable AOP proxy support.

26–50: Web / REST API

@RequestMapping – Map HTTP requests.

@GetMapping – GET request shortcut.

@PostMapping – POST request shortcut.

@PutMapping – PUT request shortcut.

@DeleteMapping – DELETE request shortcut.

@PatchMapping – PATCH request shortcut.

@PathVariable – Bind URI path variable.

@RequestParam – Bind query parameter.

@RequestBody – Bind request body to object.

@ResponseBody – Return object as HTTP response.

@CrossOrigin – Enable CORS.

@ExceptionHandler – Handle exceptions in controller.

@ControllerAdvice – Global exception handling.

@RestControllerAdvice – Global exception handling for REST.

@ModelAttribute – Bind form data to object.

@ResponseStatus – Set HTTP response status.

@CookieValue – Bind cookie value.

@RequestHeader – Bind header value.

@SessionAttributes – Store attributes in session.

@InitBinder – Customize data binding.

@PathPattern – Path pattern mapping.

@MatrixVariable – Extract matrix variables.

@RequestPart – For multipart requests.

@RestController – Already mentioned, important for API layer.

@ResponseBody – Already mentioned, converts response object to JSON.

51–75: Validation & Data Binding

@Valid – Trigger validation on object.

@Validated – Advanced validation with groups.

@NotNull – Bean validation.

@Size – Validate collection or string size.

@Min / @Max – Validate numeric limits.

@Email – Validate email format.

@Pattern – Regex validation.

@Positive / @Negative – Validate positive/negative.

@Past / @Future – Date validation.

@AssertTrue / @AssertFalse – Boolean validation.

@DecimalMin / @DecimalMax – Validate decimal ranges.

@Digits – Numeric precision validation.

@NotEmpty – Validate non-empty strings or collections.

@NotBlank – Validate string is not blank.

@Email – Already listed, frequently used.

@JsonIgnore – Ignore field in JSON serialization.

@JsonProperty – Rename field in JSON.

@JsonFormat – Format date/time in JSON.

@JsonInclude – Include only non-null fields.

@JsonCreator – Custom deserialization constructor.

@JsonValue – Serialize enum/value.

@RequestParam – Already listed, validation can be applied.

@PathVariable – Already listed, can validate.

@ExceptionHandler – Already listed, can handle validation exceptions.

@ControllerAdvice – Already listed, global validation handling.

76–100: Microservices / Advanced

@EnableEurekaClient – Register service in Eureka.

@EnableDiscoveryClient – Service discovery.

@FeignClient – Declarative REST client.

@LoadBalanced – Ribbon load balancing.

@HystrixCommand – Circuit breaker.

@EnableCircuitBreaker – Enable circuit breaker.

@EnableConfigServer – Configuration server.

@RefreshScope – Refresh bean properties at runtime.

@Cacheable – Enable caching.

@CachePut – Update cache.

@CacheEvict – Remove cache entry.

@Transactional – Manage transactions.

@EnableTransactionManagement – Enable transaction management.

@KafkaListener – Kafka message listener.

@RabbitListener – RabbitMQ listener.

@EventListener – Application event listener.

@Async – Asynchronous method execution.

@EnableAsync – Enable async execution.

@Retryable – Retry failed operations.

@EnableRetry – Enable retry mechanism.

@ApiOperation – Swagger/OpenAPI annotation.

@ApiResponse – Swagger/OpenAPI response.

@OpenAPIDefinition – OpenAPI documentation.

@Tag – Group APIs in Swagger.

@SecurityRequirement – Specify security requirements in Swagger.

========== Java 8==========

Java 8 Features & Functional Programming

What are Lambda Expressions in Java 8?

Explain the Stream API in Java 8.

What is a Functional Interface? Provide an example.

How does the default keyword work in interfaces?

What is the CompletableFuture class in Java 8?

Explain the Optional class and its use cases.

What is the Method Reference feature in Java 8?

How does the DateTime API in Java 8 improve date and time handling?
==========
Java 8 Features & Functional Programming

Lambda Expressions: What are Lambda Expressions in Java 8?

Stream API: Explain the Stream API in Java 8.

Functional Interface: What is a Functional Interface? Provide an example.

Default Methods: How does the default keyword work in interfaces?

CompletableFuture: What is the CompletableFuture class in Java 8?

Optional Class: Explain the Optional class and its use cases.

Method Reference: What is the Method Reference feature in Java 8?

DateTime API: How does the DateTime API in Java 8 improve date and time handling?
==========
Functional Interface: What is a functional interface in Java? Provide an example.

Method References: Explain method references in Java 8.

Optional Class: What is the Optional class in Java 8, and how is it used to avoid NullPointerException?

Stream API: How does the Stream API in Java 8 facilitate functional-style operations on collections?

CompletableFuture: What is CompletableFuture in Java 8, and how does it differ from Future?
==========
map() vs. flatMap(): What is the difference between map() and flatMap() in Java Streams?

Optional Class: What is the Optional class in Java 8, and how is it used to avoid NullPointerException?
==========
Functional Interface: What is a functional interface in Java? Provide an example.

Method References: Explain method references in Java 8.

Optional Class: What is the Optional class in Java 8, and how is it used to avoid NullPointerException?

Stream API: How does the Stream API in Java 8 facilitate functional-style operations on collections?

CompletableFuture: What is CompletableFuture in Java 8, and how does it differ from Future?
==========

==========
What is a functional interface? Provide an example.

Explain the use of Optional in Java 8.

What are the differences between map() and flatMap() in Streams?

How does Collectors.toMap() work?

Explain the purpose of CompletableFuture in Java 8.
==========
==========
Java 8 Features

What is a functional interface? Provide an example.

Explain the use of Optional in Java 8.

What are the differences between map() and flatMap() in Streams?

How does Collectors.toMap() work?

Explain the purpose of CompletableFuture in Java 8.

Spring Framework & Microservices

What is dependency injection in Spring?

Explain the difference between @Component, @Service, and @Repository.

How does Spring Boot simplify application development?

What are Spring profiles, and how are they used?

Explain the concept of microservices and how Spring Boot supports them.
==========
==========
What is a functional interface?

Can you provide an example of a functional interface in Java 8?

What is a lambda expression? Provide an example.

What is the Stream API in Java 8?

What is the difference between map() and flatMap() in Java 8 Streams?

What is the Optional class in Java 8?

What is the default keyword in interfaces?

What is the java.time package?

What is the difference between Comparable and Comparator?

What new methods were introduced in the Object class in Java 8?

What are the new features introduced in Java 8 for collections?

What is method reference in Java 8?

What is the difference between forEach() and map() in streams?

What are the types of streams in Java 8?

How is parallel stream different from a normal stream in Java 8?

==========

What is a functional interface?

Can you provide an example of a functional interface in Java 8?

What is a lambda expression? Provide an example.

What is the Stream API in Java 8?

What is the difference between map() and flatMap() in Java 8 Streams?

What is the Optional class in Java 8?

What is the default keyword in interfaces?

What is the java.time package?

What is the difference between Comparable and Comparator?

What new methods were introduced in the Object class in Java 8?

What are the new features introduced in Java 8 for collections?

What is method reference in Java 8?

What is the difference between forEach() and map() in streams?

What are the types of streams in Java 8?

How is parallel stream different from a normal stream in Java 8?

==========

How do you implement filtering using Java 8 streams?

What is the difference between intermediate and terminal operations in streams?

How do you sort a list using Java 8 streams?

How do you group elements of a collection using streams?

What is the difference between sequential and parallel streams?

How do you handle exceptions in lambda expressions?

Can you chain multiple stream operations? Give an example.

How do you convert a list of objects to a map using Java 8 streams?

What is the difference between collect(Collectors.toList()) and collect(Collectors.toSet())?

How do you find the maximum or minimum element in a collection using streams?

How do you remove duplicates from a list using streams?

What is the difference between peek() and map() in streams?

How do you perform reduction operations using streams (reduce() method)?

How do you flatten a collection of lists using flatMap()?

What are some common use cases of Optional in real projects?
==========
==========
How do you filter a list using multiple conditions in Java 8 streams?

How can you handle null values in streams safely?

Explain the difference between reduce() and collect() in streams.

How do you perform a group-by operation on a stream of objects?

How do you use Collectors.partitioningBy() in streams?

What is the difference between map() and mapToInt()/mapToDouble() in streams?

How do you sort a map by its values using Java 8 streams?

Explain the difference between anyMatch(), allMatch(), and noneMatch() in streams.

How do you implement a custom comparator using lambda expressions?

How do you flatten a nested list of objects using flatMap()?

How do you find duplicate elements in a collection using streams?

How do you find the first element matching a condition using streams?

What are the differences between Stream.of() and Arrays.stream()?

How do you combine multiple predicates using and() / or() in streams?

How do you measure the performance difference between sequential and parallel streams?
==========
==========
How do you create an immutable collection using Java 8 features?

How do you combine two streams into one stream?

Explain Collectors.toMap() with an example.

How do you find the count of distinct elements in a collection using streams?

How do you convert a stream of objects into a map with multiple values per key?

How do you implement a custom collector in Java 8?

Explain the difference between findFirst() and findAny() in streams.

How do you use Collectors.joining() to concatenate strings from a list?

How do you sort elements of a stream based on multiple fields?

How do you remove null elements from a list using streams?

How do you perform parallel processing safely with shared mutable data?

How do you implement conditional filtering in streams (like if in filter)?

How do you use Collectors.partitioningBy() to split a collection into two groups?

How do you perform a cumulative sum or reduction on a stream?

How do you check if a stream contains any duplicates efficiently using Java 8?
==========
2. Java 8 & Functional Programming

Explain functional interfaces and provide an example.

Implement Runnable using lambda expressions.

Use Optional to handle null values.

Filter and sort a list of objects using streams.

Group elements of a list based on a property using streams.

Convert a list of strings to uppercase using streams.

Find the maximum element in a collection using streams.

Remove duplicates from a list using streams.

Partition a collection into two groups based on a predicate using streams.

Convert a list of objects to a map using Collectors.toMap().
==========

==========JAVA=
==========

🔧 Core Java & OOPs Concepts

What are the four major Object-Oriented Programming (OOP) concepts in Java?

The four major concepts of Object-Oriented Programming (OOP) in Java are encapsulation, inheritance, polymorphism, and abstraction. These "four pillars" are a foundational part of how Java is designed and used to create modular, reusable, and maintainable code.
1. Encapsulation
Encapsulation is the practice of bundling the data (variables) and the code that operates on that data (methods) into a single unit, which is the class.
Data hiding: The internal state of an object is hidden from the outside world by declaring variables as private.
Controlled access: Access to the private data is provided through public "getter" and "setter" methods. These methods can include validation logic to ensure data integrity.
Example: A BankAccount class might have a private balance variable. A public deposit() method would be the only way to modify the balance, and it could include checks to ensure that a negative amount cannot be deposited.

2. Inheritance
Inheritance is a mechanism that allows a new class to inherit fields and methods from an existing class, establishing a parent-child relationship.
Code reusability: It enables subclasses (child classes) to reuse code from a superclass (parent class), reducing code duplication.
The extends keyword: This keyword is used to create a new class that inherits from another. The new class is said to have an "is-a" relationship with the parent class (e.g., a Dog "is-a" Animal).
Example: A Car class and a Bicycle class can both inherit from a Vehicle superclass, inheriting common properties like speed and color

3. Polymorphism
Polymorphism literally means "many forms" and is the ability of an object to take on many forms. In Java, it allows the same method call to behave differently depending on the object that invokes it.
Method overloading (compile-time polymorphism): Multiple methods in the same class share the same name but have different parameters. The compiler decides which method to call based on the arguments provided.
Method overriding (runtime polymorphism): A subclass provides a specific implementation for a method that is already defined in its superclass. The decision of which method to execute is made at runtime.
Example: An Animal class can have a makeSound() method. A Dog subclass and a Cat subclass can each provide their own unique implementation of makeSound(), even though they are called by the same method name.

4. Abstraction
Abstraction is the process of hiding the complex implementation details and showing only the essential or relevant information to the user.
Abstract classes and interfaces: Abstraction is achieved in Java primarily through abstract classes and interfaces. Abstract classes can have both abstract (no implementation) and concrete (with implementation) methods. Interfaces can only define method signatures, achieving 100% abstraction.
Focus on "what," not "how": It focuses on what an object does rather than how it does it, simplifying the user's interaction with the program.
Example: A user operates a car by pressing the brake and gas pedals, but they don't need to know the complex internal workings of the engine or braking system. Similarly, a Shape abstract class can have an abstract calculateArea() method, and the user can call this method without knowing the specific formula used by a Circle or Rectangle subclass.

Explain the differences between String and StringBuilder.

“String in Java is immutable, meaning any modification creates a new object. It is stored in the string pool and is thread-safe. StringBuilder is mutable, allowing modification of the same object using methods like append, insert, or delete. It is faster for frequent modifications but is not thread-safe. Use String when values rarely change and StringBuilder when frequent modifications are needed.”

What is the significance of immutability in String?

“Immutability in String ensures that once created, its value cannot be changed. This provides thread safety, security, memory efficiency via string pool, consistent hashcodes for keys in collections, and allows safe caching. It is a key reason why strings are heavily used in Java for sensitive data, maps, and performance-critical operations.”

Can you define a method in an interface?

“Yes, you can define methods in an interface. Traditionally, only abstract methods were allowed, which must be implemented by the implementing class. Since Java 8, default and static methods with a body are allowed, and since Java 9, private methods can be defined to help organize code inside the interface. Abstract methods define the contract, default methods provide default behavior, static methods belong to the interface itself, and private methods are for internal reuse.”

What is static overloading and dynamic overloading?

“Static or compile-time polymorphism is achieved via method overloading, where the compiler determines which method to call based on method parameters. Dynamic or run-time polymorphism is achieved via method overriding, where the JVM decides at runtime which overridden method of the subclass to execute. Static binding is faster, while dynamic binding is flexible and allows runtime behavior

modification.”

Explain the concept of functional interfaces.

“A Functional Interface is an interface in Java that contains exactly one abstract method, which allows it to be implemented using lambda expressions or method references. It can also have default, static, or private methods. Using @FunctionalInterface annotation is optional but helps the compiler enforce the rule of a single abstract method.”

What is the use of @Primary annotation in Spring?

“The @Primary annotation in Spring is used to indicate which bean should be considered as the default when multiple beans of the same type exist. It resolves autowiring ambiguity by selecting the bean marked with @Primary unless another bean is explicitly chosen using @Qualifier. It can be applied on classes annotated with @Component or on @Bean methods.”

What is an Optional class and its use?

“The Optional class in Java is a container that may or may not contain a non-null value. It is primarily used to avoid null checks and NullPointerExceptions, making the code safer and more readable. Optional provides methods like of, empty, ofNullable, ifPresent, orElse, and orElseThrow to handle values in a functional style.”

Difference between Optional.of() vs Optional.ofNullable().

“Optional.of() creates an Optional containing a non-null value and will throw NullPointerException if the value is null. Optional.ofNullable() creates an Optional that may contain a null value; if the value is null, it returns Optional.empty(). Use of() when you are certain the value is non-null, and ofNullable() when the value might be null.”

==========
Core Java & Object-Oriented Programming (OOP)

String Immutability: Why is the String class immutable in Java?

“The String class is immutable in Java to provide thread safety, security, efficient memory usage via the string pool, and consistent hashcodes for collections. Once created, a String’s value cannot be modified, which prevents accidental or malicious changes.”

StringBuilder vs. StringBuffer: What is the difference between StringBuilder and StringBuffer?

“StringBuilder is mutable and faster but not thread-safe, suitable for single-threaded operations. StringBuffer is also mutable but thread-safe, making it suitable for multi-threaded environments. Both provide methods like append, insert, delete, and reverse.”

Array vs. List: Can you explain the differences between an array and a list in Java?

“An array has a fixed size and can store primitives or objects. A List is part of the Collection framework, dynamically sized, object-only, and provides rich methods for manipulation. Use arrays for fixed-size data and Lists for dynamic collections.”

WeakHashMap: What is a WeakHashMap in Java, and how does it differ from a regular HashMap?

“WeakHashMap stores keys as weak references, allowing the garbage collector to

remove entries when keys are no longer in use. In contrast, HashMap stores keys strongly, so they remain in memory. WeakHashMap is useful for memory-sensitive caches.”

Inheritance Types: What types of inheritance are supported in Java?

“Java supports single, multilevel, and hierarchical inheritance. Multiple class inheritance is not allowed, but multiple interface inheritance is supported to avoid ambiguity.”

Entity Relationships: How does Java support entity relationships like one-to-one, one-to-many, and many-to-many?

“Java supports entity relationships using JPA annotations: @OneToOne, @OneToMany / @ManyToOne, and @ManyToMany. These relationships help model real-world associations between entities in the database.”

==========
String vs. StringBuilder vs. CharSequence: What are the key differences between String, CharSequence, and StringBuilder?

“String is immutable and stored in the string pool, suitable for fixed text. StringBuilder is mutable and faster for frequent modifications but not thread-safe. CharSequence is a general interface implemented by both String and StringBuilder to provide common methods for sequence manipulation.”

NullPointerException Prevention: How do you prevent NullPointerException in Java effectively?

“To prevent NullPointerException, use Optional for nullable values, always perform null checks, initialize objects before use, provide default values, and avoid returning null from methods. Using these practices ensures safer and more readable code.”

Method Hiding: What is method hiding in Java?

“Method hiding occurs when a static method in a subclass has the same signature as a static method in the superclass. Unlike overriding, method hiding is resolved at compile-time based on the reference type, not the object type.”

Access Modifiers: What is the difference between default and public access modifiers?

“Default access modifier provides package-level visibility, meaning classes or members are accessible only within the same package. Public access modifier allows global visibility, meaning the class or member can be accessed from any package.”

Serialization: What is serialization in Java, and how is it achieved?

“Serialization in Java is the process of converting an object into a byte stream for storage or transmission. It is achieved by implementing the Serializable interface and using ObjectOutputStream and ObjectInputStream. The transient keyword can be used to exclude fields from serialization.”

==========
String vs. StringBuilder vs. CharSequence: What are the key differences between String, CharSequence, and StringBuilder?

Preventing NullPointerException: How do you prevent NullPointerExcepti=on in Java effectively?

Method Hiding: What is method hiding in Java?

Access Modifiers: What is the difference between default and public access modifiers?

Serialization: What is serialization in Java, and how is it achieved?

==========

Explain the concept of OOP and its advantages.

What is the difference between String, StringBuilder, and StringBuffer?

How does garbage collection work in Java?

What are the different types of inheritance supported in Java?

Explain the concept of method overloading and method overriding.

==========

Explain the concept of OOP and its advantages.

What is the difference between String, StringBuilder, and StringBuffer?

How does garbage collection work in Java?

What are the different types of inheritance supported in Java?

Explain the concept of method overloading and method overriding.

==========

OOP Principles: Explain the four major OOP concepts in Java.

String Immutability: What is string immutability?

String vs StringBuffer: Difference between String and StringBuffer.

Inheritance Types: What types of inheritance are supported in Java?

Functional Interfaces: What is a functional interface? Provide an example.

Overloading: Explain static overloading and dynamic overloading.

==========

What is the difference between String, StringBuilder, and StringBuffer?

What is a Functional Interface?

What is the difference between map() and flatMap() in Java Streams?

What is the difference between Method Overloading and Method Overriding?

What is the difference between Optional.of() and Optional.ofNullable()?

What is the difference between JVM, JRE, and JDK?

What is the difference between == and .equals() in Java?

What is the use of final, finally, and finalize() in Java?

==========

What are the four major OOP concepts in Java?

Explain the difference between == and .equals() in Java.

What is method overloading and method overriding?

What is the significance of immutability in String?

Explain the concept of functional interfaces.

What is the difference between String, StringBuilder, and StringBuffer?

What is the use of super and this keywords in Java?

What is the purpose of final, finally, and finalize() in Java?

Explain the concept of serialization and deserialization.

What is the difference between ArrayList and LinkedList?

==========
==========